Deciding Joinability Modulo Ground Equations In Operational Type Theory

Adam Petcher and Aaron Stump

Operational Type Theory (OpTT) can be used to construct and check proofs related to programs, but the development of these proofs can be somewhat tedious. An algorithm is presented that can be used to automatically generate proofs of equality in OpTT. The algorithm takes as input a set of ground equations and two terms that should be tested for joinability modulo the supplied ground equations. The algorithm will equate the terms if and only if there exists an OpTT proof that can equate the two terms using only the proof rules related to evaluation under the operational semantics, symmetry,... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research
Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
Deciding Joinability Modulo Ground Equations In Operational Type Theory

Adam Petcher and Aaron Stump

Complete Abstract:

Operational Type Theory (OpTT) can be used to construct and check proofs related to programs, but the development of these proofs can be somewhat tedious. An algorithm is presented that can be used to automatically generate proofs of equality in OpTT. The algorithm takes as input a set of ground equations and two terms that should be tested for joinability modulo the supplied ground equations. The algorithm will equate the terms if and only if there exists an OpTT proof that can equate the two terms using only the proof rules related to evaluation under the operational semantics, symmetry, transitivity, and congruence with respect to the supplied ground equations. The description of this algorithm is accompanied by a proof that the algorithm is partially correct.
Deciding Joinability Modulo Ground Equations In Operational Type Theory

Authors: Adam Petcher, Aaron Stump

Abstract: Operational Type Theory (OpTT) can be used to construct and check proofs related to programs, but the development of these proofs can be somewhat tedious. An algorithm is presented that can be used to automatically generate proofs of equality in OpTT. The algorithm takes as input a set of ground equations and two terms that should be tested for joinability modulo the supplied ground equations. The algorithm will equate the terms if and only if there exists an OpTT proof that can equate the two terms using only the proof rules related to evaluation under the operational semantics, symmetry, transitivity, and congruence with respect to the supplied ground equations. The description of this algorithm is accompanied by a proof that the algorithm is partially correct.
DECIDING JOINABILITY MODULO GROUND EQUATIONS IN
OPERATIONAL TYPE THEORY

by
Adam Petcher

Prepared under the direction of Professor Aaron Stump

A thesis presented to the School of Engineering of
Washington University in partial fulfillment of the
requirements for the degree of
MASTER OF SCIENCE

May 2008
Saint Louis, Missouri
Operational Type Theory (OpTT) can be used to construct and check proofs related to programs, but the development of these proofs can be somewhat tedious. An algorithm is presented that can be used to automatically generate proofs of equality in OpTT. The algorithm takes as input a set of ground equations and two terms that should be tested for joinability modulo the supplied ground equations. The algorithm will equate the terms if and only if there exists an OpTT proof that can equate the two terms using only the proof rules related to evaluation under the operational semantics, symmetry, transitivity, and congruence with respect to the supplied ground equations. The description of this algorithm is accompanied by a proof that the algorithm is partially correct.
# Table of Contents

List of Figures............................................................................................................ iii

1 Introduction ...........................................................................................................1
   1.1 Operational Type Theory.....................................................................................1
   1.2 Proof Complexity in OpTT..................................................................................2
   1.3 Related Work .....................................................................................................4

2 Problem Description..............................................................................................6
   2.1 Problem Definition.............................................................................................6
   2.2 Inherent Difficulties.........................................................................................11
   2.3 Desired Algorithm Qualities.............................................................................12
   2.4 Additional Notation.........................................................................................13
   2.5 Example Input ..................................................................................................14

3 Algorithm Description .........................................................................................21
   3.1 Algorithm Overview.........................................................................................21
   3.2 Algorithm Design Considerations......................................................................21
   3.3 Algorithm Definition.......................................................................................22
      3.3.1 Evaluation Modulo U................................................................................22
      3.3.2 Consistency..............................................................................................24
      3.3.3 Normalization of U................................................................................25
      3.3.4 The Hypjoin Algorithm...........................................................................25
   3.4 Algorithm Implementation Notes......................................................................26
      3.4.1 Equivalence Relations...............................................................................26
      3.4.2 Deciding Equivalence.............................................................................26
      3.4.3 Deciding Consistency.............................................................................26
      3.4.4 Evaluation Modulo U.............................................................................27

4 Proof of Correctness.............................................................................................28
   4.1 Soundness of Hypjoin Algorithm......................................................................28
   4.2 Completeness of Hypjoin Algorithm.................................................................37

5 Conclusion and Future Work...............................................................................76
   5.1 Termination......................................................................................................76
   5.2 Clash/Contra ..................................................................................................76
   5.3 Injectivity.........................................................................................................76

References..................................................................................................................78

Vita.............................................................................................................................79
List of Figures

Figure 1-1. Example Definitions.................................................................2
Figure 1-2. Standard Proof of Reflexivity of ge..............................................3
Figure 1-3. Proof of Reflexivity of ge using hypjoin........................................4
Figure 2-1. Definitions of Contexts .............................................................7
Figure 2-2. Definition of Bound .................................................................7
Figure 2-3. Recursive Definition of Bound ...............................................8
Figure 2-4. Inductive Definition of Operational Semantics of OpTT ..........8
Figure 2-5. Inductive Definition of Capture-Avoiding Substitution ..........9
Figure 2-6. Bound Variable-Preserving Evaluation ..................................9
Figure 2-7. Equality Modulo Ground Equations .....................................10
Figure 2-8. Inductive Definition of Single Step Joinability Modulo Ground Equations .........................................................10
Figure 2-9. Inductive Definition of n-step Joinability Modulo Ground Equations .................................................................10
Figure 2-10. Simple hypjoin Input ...............................................................14
Figure 2-11. Definition of gcZ .................................................................15
Figure 2-12. Standard Definition of geTrans ..........................................17
Figure 2-13. Hypjoin Definition of geTrans .............................................18
Figure 2-14. Standard Definition of plus_assoc ........................................19
Figure 2-15. Hypjoin Definition of plus_assoc .........................................20
Figure 2-16 Hypjoin of Fun Terms ............................................................20
Figure 3-1. Restrictive Equality Modulo Ground Equations ....................23
Figure 3-2. Evaluation Modulo U ..............................................................23
Figure 3-3. Consistency Rules .................................................................24
Figure 3-4. Evaluation of U ................................................................. 25
Figure 3-5. The Hypjoin Algorithm ....................................................... 25
Figure 4-1. Structure of Soundness Proof ............................................. 29
Figure 4-2. Structure of Completeness Proof ........................................ 39
1 Introduction

Operational Type Theory (OpTT) [8] is a powerful and convenient system for developing programs and proofs related to those programs. In some cases, however, the development of proofs in OpTT can be excessively tedious. This paper describes a class of proofs that could be automatically generated by an algorithm that is capable of deciding the “joinability modulo ground equations” problem. Such an algorithm would significantly reduce the amount of effort required to develop proofs in OpTT.

The primary contribution of this paper is an algorithm which attempts to decide the joinability modulo ground equations problem. Also included is a proof that the algorithm is correct when it terminates and certain “consistency” conditions are satisfied. An analysis of the termination properties of the algorithm is not included, but it is conjectured that the algorithm will terminate if recursion is finite in the supplied program terms and in all terms that are derived by the algorithm.

This chapter describes OpTT and introduces the joinability modulo ground equations problem.

1.1 Operational Type Theory

Operational Type Theory is a system in which programs and proofs have distinct type systems, but can nevertheless be combined such that a program can contain a proof, and a proof related to a program can be documented and checked. The primary motivation for this distinction is to allow for the independence of the semantics of program evaluation, proof normalization, and definitional equality. As a result of this independence, proof complexity is not dependent on program complexity, and it is possible to develop simple proofs related to complex programs.
1.2 Proof Complexity in OpTT

In many practical cases, proofs in OpTT tend to be mostly composed of tedious sub-proofs that could easily be generated using simple automated reasoning. These proofs are usually composed of instances of a handful of common patterns, and they tend to have relatively large amounts of redundant information. If the need to manually develop such trivial proofs was removed, then proof development in OpTT would be significantly less time consuming.

A simple example will be used to further describe the problem. Additional examples are used to provide more detail in chapter 2. Several examples in this paper assume the definitions in figure 1-1 which define natural numbers, Booleans, and \( \geq \) for natural numbers, respectively.

\[
\text{Inductive } \mathsf{nat} : \text{type} := \\
\quad \mathsf{Z} : \mathsf{nat} \\
| \quad \mathsf{S} : \text{Fun}(x:\mathsf{nat}).\mathsf{nat}.
\]

\[
\text{Inductive } \mathsf{bool} : \text{type} := \\
\quad \mathsf{T} : \mathsf{bool} \\
| \quad \mathsf{F} : \mathsf{bool}.
\]

Define \( \text{ge} : \text{Fun}(n \text{ m} : \mathsf{nat}).\mathsf{bool} := \)

\[
\begin{align*}
\text{fun} & (n \text{ m} : \mathsf{nat}):\mathsf{bool}. \\
& \text{match m by x y return bool with} \\
& \quad \mathsf{Z} => \mathsf{T} \\
& | \quad \mathsf{S} m' => \text{match n by x1 y1 return bool with} \\
& \quad \quad \mathsf{Z} => \mathsf{F} \\
& \quad | \quad \mathsf{S} n' => (\text{ge} n' m') \\
& \text{end} \\
& \text{end.}
\end{align*}
\]

Figure 1-1. Example Definitions
One might prove that \( \text{ge} \) is reflexive, that is, for all \( n \), \( (\text{ge} \ n \ n) = \text{T} \), as follows. Proof by induction on \( n \): In the base case, \( n = \text{Z} \). Because \( n = \text{Z} \) and congruence of terms w.r.t. equality of subterms, \( (\text{ge} \ n \ n) = (\text{ge} \ \text{Z} \ \text{Z}) \). Due to the definition of \( \text{ge} \), \( (\text{ge} \ \text{Z} \ \text{Z}) = \text{T} \). From transitivity of equality, \( (\text{ge} \ n \ n) = \text{T} \). In the step case, \( n = (\text{S} \ n') \) and there is an induction hypothesis that says \( (\text{ge} \ n' \ n') = \text{T} \). Because \( n = (\text{S} \ n') \) and congruence of terms, \( (\text{ge} \ n \ n) = (\text{ge} \ (\text{S} \ n') \ (\text{S} \ n')) \). Due to the definition of \( \text{ge} \), \( (\text{ge} \ (\text{S} \ n') \ (\text{S} \ n')) = (\text{ge} \ n' \ n') \). Due to the induction hypothesis and transitivity of equality, \( (\text{ge} \ n \ n) = \text{T} \). This proof in Op$\text{TT}$ syntax is shown in figure 1-2. Note that trans, cong, and join are the proof rules for transitivity of equality, congruence of terms, and evaluation under the operational semantics, respectively.

Define \( \text{geRefl} : \text{Forall}(n:\text{nat}). \ (\text{ge} \ n \ n) = \text{T} \) :=

\[
\text{induction}(n:\text{nat}) \text{ by } x1 \ x2 \ \text{IH} \ \text{return} \ \{(\text{ge} \ n \ n) = \text{T}\} \text{ with} \\
Z \Rightarrow \text{trans} \\
\qquad \text{cong} \ (\text{ge} \ * \ *) \ x1 \\
\qquad \text{join} \ (\text{ge} \ \text{Z} \ \text{Z}) \ \text{T} \\
| \ \text{S} \ n' \Rightarrow \text{trans} \\
\qquad \text{trans} \\
\qquad \text{cong} \ (\text{ge} \ * \ *) \ x1 \\
\qquad \text{join} \ (\text{ge} \ (\text{S} \ n') \ (\text{S} \ n')) \ (\text{ge} \ n' \ n') \\
[\text{IH} \ n']
\]

end.

Figure 1-2. Standard Proof of Reflexivity of \( \text{ge} \)

Each case of the induction is little more than a series of cong and join rules linked together by trans rules. It is common for proofs in Op$\text{TT}$ to be composed of such large sub-proofs that contain only cong, join, symm, and trans rules. The cong proof rule also includes a ground equation as an argument, so this class of Op$\text{TT}$ proof will be referred to as a joinability modulo ground equations proof, and the problem of determining the existence of such proofs is called the joinability modulo ground equations problem. Because the need for this sort of proof is so common, an algorithm which can decide the joinability modulo
ground equations problem would greatly simplify proofs in many cases. For example, it would be desirable to simply write the proof from figure 1-2 as shown in figure 1-3.

\[
\text{Define } \text{geRefl} : \forall (n : \text{nat}). \{ (\text{ge } n \ n) = T \} := \\
\text{induction}(n : \text{nat}) \text{ by } x1 \ x2 \ \text{IH return } \{ (\text{ge } n \ n) = T \} \text{ with} \\
\ Z \Rightarrow \text{hypjoin } (\text{ge } n \ n) \ T \text{ by } x1 \ \text{end} \\
\ | \ S \ n' \Rightarrow \text{hypjoin } (\text{ge } n \ n) \ T \text{ by } x1 \ [\text{IH } n'] \ \text{end} \\
\text{end.}
\]

Figure 1-3. Proof of Reflexivity of ge using hypjoin

Note that hypjoin is the name given to the new proof rule in OpTT that can algorithmically decide the joinability modulo ground equations problem. The “hyp” in hypjoin stands for hypothesis, and the name “hypjoin” represents the fact that we are attempting to join two terms given a set of user-provided hypotheses. This problem is described in greater detail in chapter 2.

Though the example in figure 1-2 was very simple, the purpose of hypjoin is to take large, complicated proofs and collapse them into a single hypjoin statement. More complex examples are provided in section 2.5. The goal of this paper is to define an algorithm that implements hypjoin and to prove that the algorithm is correct to the extent that it can be used in practical cases. The algorithm is defined in chapter 3, and the proof of correctness is provided in chapter 4.

1.3 Related Work

Congruence closure[2] is a decision procedure that can be used to decide the equivalence of two terms given a finite set of ground equations \( E \) that defines a set of allowed substitutions. This problem is known as the ground word problem for \( E[1] \). Congruence closure is a significant part of the hypjoin algorithm, but using it to solve the entire problem is
challenging because it is difficult to effectively represent the operational semantics as a finite set of ground equations.

The Calculus of Congruent Inductive Constructions (CCIC)\cite{3}\cite{4} is an extension to the Calculus of Inductive Constructions (CIC) that incorporates decision procedures for first-order theories into the CIC conversion rule. This system is very powerful in that the user-supplied hypotheses can be equations containing quantified variables, and as such it can simplify many types of complex proofs. Though hypjoin is less general than CCIC, the fact that it is limited to reasoning on ground equations provides some significant benefits when compared to CCIC. The primary benefit is that the computational nature of the hypjoin algorithm will probably make it more understandable to the programmer. If hypjoin fails, an error message will be presented that can help the programmer understand why the failure occurred. If conversion fails, however, it may not be possible to generate such an error message. Another benefit of hypjoin is that it can be seen as an optional extension to OpTT, whereas the modified conversion rule is a core part CCIC. Due to this structure, the theoretical properties of OpTT (cut elimination, type soundness) are not affected by hypjoin.
2 Problem Description

This section contains a discussion of the issues surrounding the joinability modulo ground equations problem. Section 2.1 contains a definition of the problem, and a description of the difficulties inherent to solving the problem is presented in section 2.2. Section 2.3 presents a set of achievable goals that would allow an algorithm that partially decides the joinability modulo ground equations problem to be useful in practical cases.

2.1 Problem Definition

The joinability modulo ground equations problem can be described informally as follows. Given two terms, s and t, and a set of ground equations U, is there a path from s to t using evaluation under the operational semantics and substitutions from U? This paper is concerned with solving this problem using the operational semantics and constraints of OpTT. The operational semantics of OpTT can be defined as follows.

The definitions of contexts \( E \) and \( E^+ \) are found in [8] and reproduced in figure 2-1 for convenience. The definition of \( E^+ \) excludes contexts that are types since it is assumed that all type annotations have been dropped in the terms and equations that are provided to hypjoin. It is often helpful to consider contexts that are exactly one level deep. These contexts are described by the symbols \( E_i \) and \( E_i^+ \) which are identical to \( E \) and \( E^+ \), respectively, except the first set of contexts are one level deep. For example, \( E_i^+[a] \) could describe \((a\ b)\), but not \(((a\ b)\ c)\). Another difference is that the one-level-deep contexts could have an empty set of holes, as opposed to traditional contexts that must have at least one hole, but the context itself could be empty. The context \( E_i^* \) is identical to \( E_i^+ \) except it has an arbitrary number of (zero or more) holes. Notation such as \( E_i^*[\overline{a}] \) refers to a context and the list of terms \( \overline{a} \), in the holes of that context. Any time this notation is used, there is an element in \( \overline{a} \) for every possible hole in \( E_i^* \) based on the form of the term. For example, if \( E_i^*[\overline{a}] \) refers to a let term, then there must be exactly two elements in \( \overline{a} \).
E ::= * || (E X) || (I E) || 
  let x = E by y in y ||
  match E by x y return T with
  \[ c_i x_i \Rightarrow t_1 | ... | c_n x_n \Rightarrow t_n \] end

\[ E^+ ::= * || X || (E^+ E^+) || 
  \text{fun } r(x:E^+) : E^+.E^+ ||
  \text{let x = } E^+ \text{ by y in } E^+ ||
  \text{match } E^+ \text{ by x y return } E^+ \text{ with}
  \[ c_i x_i \Rightarrow E^+ 1 | ... | c_n x_n \Rightarrow E^+ \] end

Figure 2-1. Definitions of Contexts

When using one-level-deep contexts, one can easily extract the bound variables defined in that context. Later definitions will refer to a function called Bound that is used to produce a set of bound variables for a given one-level deep context. Bound is defined in figure 2-2.

\[
\begin{align*}
\text{Bound}((a_1 a_2), 1) &= \emptyset \\
\text{Bound}((a_1 a_2), 2) &= \emptyset \\
\text{Bound}((\text{fun } r(x!:a), 1) &= \{ x \} \\
\text{Bound}(\text{let } x = a_1 \text{ in } a_2, 1) &= \emptyset \\
\text{Bound}(\text{let } x = a_1 \text{ in } a_2, 2) &= \{ x \} \\
\text{Bound}(\text{match } s \text{ return } ! \text{ with } c_i x_i \Rightarrow a_i | ... | c_n x_n \Rightarrow a_n \text{ end}, 1) &= \emptyset \\
\text{Bound}(\text{match } s \text{ return } ! \text{ with } c_i x_i \Rightarrow a_i | ... | c_n x_n \Rightarrow a_n \text{ end}, m) &= \{ x_{m-1} \}
\end{align*}
\]

Figure 2-2. Definition of Bound

For any context E with only one hole, Bound(E) is shorthand for Bound(E, k) where k is the position of the only hole in E. We can also apply Bound to an evaluation context with only one hole but arbitrary depth. In which case the function is defined recursively as shown in figure 2-3.
The definition of the evaluation relation is provided in figures 2-4, 2-5, and 2-6. The definition provided uses the one level deep context defined earlier, but is nevertheless equivalent to the operational semantics defined in [8]. Note that \( \rightarrow \) is the symbol used to represent a single step of evaluation.

The relation is defined using a set of rules which combine to form an inductive definition. That is, any time two terms are related by \( \rightarrow \), there must exist a derivation composed of these rules which relate the terms. The notation \( k : a R b \) is used to represent the fact that \( a \) is related to \( b \) in inductively-defined relation \( R \) by a derivation of depth \( k \). This convention is used any time a relation is defined inductively in this paper.

\[
\begin{align*}
  f = \text{fun} \ r(x:!) :! b \\
  (f \ I) \rightarrow b[I/x][f/r]
\end{align*}
\]

\[
\begin{align*}
  \text{match}(c, T) \text{ return } \text{ with } c_1 x_1 \Rightarrow s_1 | \ldots | c_n x_n \Rightarrow s_n \text{ end } & \rightarrow s_i [T/x_i]
\end{align*}
\]

\[
\begin{align*}
  \text{let } x = I \text{ in } t & \rightarrow t[I/x]
\end{align*}
\]

\[
\begin{align*}
  E_i[\text{abort }] & \rightarrow \text{abort }!
\end{align*}
\]

\[
\begin{align*}
  t \rightarrow t' \\
  E_i[t] & \rightarrow E_i[t']
\end{align*}
\]

Figure 2-4. Inductive Definition of Operational Semantics of OpTT

Note that \( \Gamma \) is a set of bound variables and, in general, \( \Gamma \triangleright a R b \) means that \( a \) and \( b \) are related in \( R \) and no bound variables, as defined by \( \Gamma \), are introduced or eliminated when \( a \) is replaced with \( b \).
\[
\begin{array}{c|c|c}
\Gamma \cap (\text{Vars}(s_n) \cup \text{Vars}(t_n)) = \emptyset & \Gamma \triangleright t_n[\overline{s}/\overline{T}] = s_n & \text{CAS-BASE} \\
\\hline
\forall n \quad \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n[\overline{T}/\overline{s}] = b_n & \Gamma \triangleright (E^*_i[\overline{a}])[\overline{T}/\overline{s}] = E^*_i[\overline{b}] & \text{CAS-IND} \\
\end{array}
\]

Figure 2-5. Inductive Definition of Capture-Avoiding Substitution

\[
\frac{\text{Vars}(s) \cup \text{Vars}(t) \cap \Gamma = \emptyset}{\Gamma \triangleright s \rightarrow t}
\]

Figure 2-6. Bound Variable-Preserving Evaluation

The set of ground equations will be referred to as \(U\) and has the following form: \(\{u_1 = u_1', u_2 = u_2', ..., u_n = u_n'\}\). Two terms, \(s\) and \(t\), are equal modulo a set of ground equations \(U\) if there exists a sequence of substitutions defined by \(U\) that can be applied to \(s\) and \(t\) to produce two definitionally equal terms. This equality is defined in figure 2-7.

It is assumed that the variables in \(U\) only refer to free variables in \(s\) and \(t\). If there happens to be a bound variable in \(s\) or \(t\) with the same name as a variable in \(U\), then these should be treated as different variables. Techniques such as de Bruijn indices\(^5\) can be utilized to prevent inadvertent variable name collisions. As a result of this assumption, equality modulo \(U\) is not constrained by any context.

Note that in the EQ-REFL case, two terms are considered to be the same term if they are definitionally equal. It should be apparent that \(\pi_U\) is essentially the ground word problem for \(U[1]\).

The symbol \(s \leftrightarrow^*_U t\) is used to represent the notion that \(s\) and \(t\) are joinable modulo \(U\). This relation is defined in figures 2-8 and 2-9.
The symbol $\leftrightarrow$ represents the symmetric closure of $\Rightarrow$. Using the definitions above, $\Gamma \triangleright s \leftrightarrow_U t$ iff $\exists k, \Gamma \triangleright s \leftrightarrow_U t$. This convention is used for all transitive relations in this
paper. The joinability modulo ground equations problem can now be defined as follows. Given \( s, t, \) and \( U \), is it the case that \( \emptyset \triangleright s \Leftrightarrow_{U} t \) ? Note that \( \emptyset \) refers to the empty context.

In terms of OpTT, \( \emptyset \triangleright s \Leftrightarrow_{U} t \) means that there exists a proof \( P \) of \( \{s=t\} \) where \( P \) only uses the symm, trans, cong, and join proof rules. Furthermore, the cong proofs in \( P \) only reference the equations in \( U \). So an algorithm that can solve this problem would be able to determine the existence of such a proof in OpTT.

2.2 Inherent Difficulties

Unfortunately, there are some issues which will make it difficult to develop an algorithm which solves the joinability modulo ground equations problem. These difficulties are described in this section.

The first issue is that the problem, as described in the previous section, is undecidable. A special case of this problem is the one in which \( U = \emptyset \) and one of the terms does not terminate with respect to the operational semantics. In this case, the problem is equivalent to the problem of determining whether the other term terminates. Since termination is undecidable, joinability modulo ground equations must also be undecidable.

However, if it is possible to design an algorithm that terminates in practical cases and is guaranteed to be correct when it terminates, then that algorithm could still be immensely useful. In the spirit of this practicality, the theorems in this paper will often assume the termination of various algorithms. Likewise, a proof of termination of the parts of the algorithm that are unrelated to evaluation under the operational semantics is beyond the scope of this paper.
2.3 Desired Algorithm Qualities

With the issues described in the previous section in mind, this section attempts to define qualities that the algorithm must possess in order to be practically useful.

First and foremost, the algorithm should terminate in most practical cases, and it should be correct when it terminates. The algorithm presented in this paper is sound and complete with respect to joinability modulo ground equations when it terminates and certain “consistency” requirements for U are met. The algorithm is also conjectured to terminate when all of the terms encountered by the algorithm terminate with respect to evaluation under the operational semantics, but a proof of this conjecture is beyond the scope of this paper.

Next, the algorithm should be efficient. Efficiency is not explicitly defined, and an analysis of the computational complexity of the algorithm is not presented in this paper. However, the algorithm is designed in such a way that searching is minimized or eliminated. The definition of the algorithm allows for a large amount of flexibility in implementation, and efficient implementation suggestions will be provided where appropriate.

Additionally, the algorithm should support the creation of an OpTT proof that equates the supplied terms using the join, symm, trans, and cong proof rules. This means that any hypjoin proof can be replaced with the generated proof of equality, and that proof can then be checked. As a result of this requirement, there is no need to trust the implementation of hypjoin, and hypjoin has no affect on the theoretical properties of OpTT. This goal is accomplished by breaking up the hypjoin algorithm into small steps, each of which results in an equality that can be obtained from symm, join, or cong. These steps are then combined into a single proof using the trans rule.

Finally, in the event that hypjoin fails, an implementation of the algorithm should be able to present some information to the programmer in order to help him determine why hypjoin failed and what must be changed in order for hypjoin to succeed. An algorithm that is based
on searching would probably not satisfy this requirement since the only information that the algorithm could provide is that the search space was exhausted and no suitable match was found. The algorithm presented in this paper is based on normalizing all forms and then testing the normal forms for equality. These normalized terms can then be presented to the programmer, who will use the output to determine why the requested hypjoin operation was not successful.

The qualities in this section describe a sort of loose specification for an algorithm. This specification is referred to as hypjoin. The actual algorithm that is defined in this paper which satisfies this specification will be referred to as the “hypjoin algorithm.”

### 2.4 Additional Notation

This section describes some additional notation that is used in this paper and not explained elsewhere.

Notation such as $\overline{a}$ is used to denote a (possibly empty) list of items $a_1, a_2, \ldots, a_n$ where $n$ is the number of items in $\overline{a}$. A relation on list of items will be used to denote relations on the individual items: $\overline{a}R\overline{b}$ means $\forall n \ a_nRb_n$.

The notation $\Gamma \triangleright a \overline{f}$ indicates that $a$ is normal with respect to relation $R$ in context $\Gamma$. The notation $\Gamma \triangleright T(a, R)$ indicates that a terminates with respect to relation $R$ in context $\Gamma$. For these notational conventions, $\Gamma$ is omitted if the relation is not constrained by the context.

The naming conventions of [8] are followed, where possible. For example, $s$ and $t$ usually refer to terms, $I$ refers to an inactive term, and $x$ refers to a variable.
2.5 Example Input

This section contains some example input and the expected results of hypjoin. The purpose of these examples is to explain the expectations of the algorithm and the motivation behind the design of the algorithm.

The example from chapter 1 is a very simple example input set that will be discussed first. The proof using hypjoin is duplicated in figure 2-10 for convenience.

The most complicated hypjoin occurs in the S n’ case of the induction. In order to determine that (ge n n) is equivalent to T in this case, hypjoin could begin by realizing that n = (S n’) as given by x1, and then determining that (ge n n) = (ge (S n’) (S n’)). Then hypjoin would determine that (ge (S n’) (S n’)) = (ge n’ n’) due to the operational semantics and the definition of ge. Then by transitivity, (ge n n) = (ge n’ n’). In these steps, hypjoin simply found substitutions in U that allow the terms to take a step of evaluation in the operational semantics. This “evaluation modulo U” is the basis of the hypjoin algorithm. The term (ge n’ n’) is considered to be normal because there are no substitutions that would allow it to take a step of evaluation. The same can be said for the term T. From here, hypjoin can simply test the terms (ge n’ n’) and T for equivalence given substitutions from U. In this case, one of the user-provided equations is \{ (ge n’ n’) = T \} so the normalized terms are equivalent and hypjoin succeeds.

Define geRefl : Forall(n:nat). \{ (ge n n) = T \} :=
induction(n:nat) by x1 x2 IH return \{ (ge n n) = T \} with
  | Z => hypjoin (ge n n) T by x1 end
  | S n' => hypjoin (ge n n) T by x1 [IH n'] end
end.

Figure 2-10. Simple hypjoin Input

Actually, (ge n’ n’) is not normal w.r.t. the operational semantics – that term will take a step and be replaced with the definition of that function. So the user-provided equations must
also be normalized in order to successfully make the final comparison in the previous example. In fact, each user-provided equation must be evaluated modulo the other equations. The following example describes a case in which this additional evaluation is necessary. This example uses a lemma called \textit{geZ} which is described in figure 2-11.

The example is a proof of transitivity of \textit{ge} called \textit{geTrans}. Two versions of the proof are given: the first version, in figure 2-12, does not use hypjoin. The version in figure 2-13 uses hypjoin in order to simplify the proof.

\begin{verbatim}
Define geZ : Forall(n : nat)(a1 : {(ge Z n) = T}). {n = Z} :=
  induction(n:nat) by x1 x2 IH return Forall(a1:{(ge Z n) = T}). {n = Z} with
  Z => forall(a1 : {(ge Z n) = T}).x1
  | S n' => forall(a1 : {(ge Z n) = T}).
    contra trans trans trans trans
    trans cong (ge Z *) x1
    join (ge Z (S n')) F
    clash F T
  {n = Z}
end.

Figure 2-11. Definition of geZ
\end{verbatim}

The definition of \textit{geTrans} that uses hypjoin illustrates an additional part of this problem that must be addressed. The last use of hypjoin in figure 2-13 attempts to equate \((ge b' c')\) with \(T\), both of which are already normal, and the user-provided equations do not allow any substitutions that can equate these terms. In the standard version of this proof, the term \((ge b c)\) is used to bridge the gap from \((ge b' c')\) to \(T\). In order to do so, it must be determined that \((ge b c) = (ge (S b') (S c'))\), and then \((ge (S b') (S c'))\) can evaluate under the operational semantics to \((ge b' c')\). The hypjoin algorithm will solve this problem by attempting to normalize each term in the user-provided equations given substitutions allowed by all other equations. By doing so, hypjoin will conclude that \((ge b c)\) can evaluate to \((ge b' c')\). At this
point, the equation \{\text{ge b c} = T\} in U can be replaced with \{\text{ge b' c'} = T\} and hypjoin will succeed.

Define \text{geTrans} : \text{Forall}(a \ b \ c : \text{nat})

\[
\begin{align*}
(a1 : \{ (\text{ge a b}) = T \}) \\
(a2 : \{ (\text{ge b c}) = T \}). \ (\text{ge a c}) := \\
\end{align*}
\]

\text{induction}(a: \text{nat}) \text{ by } x1 \ x2 \ \text{IH return Forall}(b \ c : \text{nat})

\[
\begin{align*}
(a1 : \{ (\text{ge a b}) = T \}) \\
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

Z => foralli(b \ c : \text{nat})

\[
\begin{align*}
(a1 : \{ (\text{ge a b}) = T \}) \\
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

\text{trans cong} (\text{ge a *)} \ [\text{geZ c trans cong} (\text{ge * c}) \ \text{symm}

\[
\begin{align*}
g\text{eZ b trans cong} (\text{ge * b}) \ \text{symm x1 a1]}
\]

\text{a2]}

\text{join} (\text{ge a Z}) T

| S \ a' => induction(b: \text{nat}) \ \text{by} \ x1 \ x2 \ \text{IIH return Forall}(c : \text{nat})

\[
\begin{align*}
(a1 : \{ (\text{ge a b}) = T \}) \\
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

Z => foralli(c : \text{nat})(a1 : \{ (\text{ge a b}) = T \})

\[
\begin{align*}
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

\text{trans cong} (\text{ge a *)} \ [\text{geZ c trans cong} (\text{ge * c}) \ \text{symm ix1 a2]}

\text{join} (\text{ge a Z}) T

| S \ b' => induction(c: \text{nat}) \ \text{by} \ x1 \ x2 \ \text{IIIH return Forall}(a1 : \{ (\text{ge a b}) = T \})

\[
\begin{align*}
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

Z => foralli(a1 : \{ (\text{ge a b}) = T \})

\[
\begin{align*}
(a2 : \{ (\text{ge b c}) = T \}). \\
\end{align*}
\]

\text{trans cong} (\text{ge a *)} \ \text{ix1}
join (ge a Z) T
| S c' => forall(a1 : { (ge a b) = T})
  (a2 : { (ge b c) = T}).
trans trans trans cong (ge * c) x1
  cong (ge (S a') *) ix1
  join (ge (S a') (S c')) (ge a' c')
[IH a' b' c' trans symm trans cong (ge + b) x1
  cong (ge (S a') *) ix1
  join (ge (S a') (S b')) (ge a' b')
  a1
trans symm trans cong (ge * c) ix1
  cong (ge (S b') *) iix1
  join (ge (S b') (S c')) (ge b' c')
  a2]
end
end
end.

Figure 2-12. Standard Definition of geTrans

Define geTrans : Forall(a b c : nat)
  (a1 : { (ge a b) = T})
  (a2 : { (ge b c) = T}).
{ge a c = T} :=
induction(a:nat) by x1 x2 IH return Forall(b c : nat)
  (a1 : { (ge a b) = T})
  (a2 : { (ge b c) = T}).
  {ge a c = T} with
Z => forall(b c : nat)(a1 : { (ge a b) = T})
  (a2 : { (ge b c) = T}).
hypjoin (ge a c) T by
  [geZ c hypjoin (ge Z c) T by a2
\[ \text{hypjoin (ge a c) T by x1} \]
\[ \text{hypjoin (ge a c) T by x1 a1 end] end} \]
\[ \text{end} \]
\[ | S a' => \text{induction(b:nat) by ix1 ix2 IIH return} \]
\[ \text{Forall(c : nat)} \]
\[ (a1 : \{ (ge a b) = T \}) \]
\[ (a2 : \{ (ge b c) = T \}). \]
\[ \{(ge a c) = T\} \text{ with} \]
\[ Z => \text{forall(c : nat)(a1 : \{ (ge a b) = T \})} \]
\[ (a2 : \{ (ge b c) = T \}). \]
\[ \text{hypjoin (ge a c) T by} \]
\[ [\text{geZ c hypjoin (ge Z c) T by a2 ix1 end}] end \]
\[ | S b' => \text{induction(c:nat) by iix1 iix2 IIH return} \]
\[ \text{Forall(a1 : \{ (ge a b) = T \})} \]
\[ (a2 : \{ (ge b c) = T \}). \]
\[ \{(ge a c) = T\} \text{ with} \]
\[ Z => \text{forall(a1 : \{ (ge a b) = T \})} \]
\[ (a2 : \{ (ge b c) = T \}). \]
\[ \text{hypjoin (ge a c) T by iix1 end} \]
\[ | S c' => \text{forall(a1 : \{ (ge a b) = T \})} \]
\[ (a2 : \{ (ge b c) = T \}). \]
\[ \text{hypjoin (ge a c) T by x1 iix1} \]
\[ \text{[IH a' b' c'] hypjoin (ge a' b') T by x1 iix1 a1 end} \]
\[ \text{hypjoin (ge b' c') T by iix1 iix1 a2 end] end} \]
\[ \text{end} \]
\[ \text{end} \]
\[ \text{end}. \]

Figure 2-13. Hypjoin Definition of geTrans
The next example shows some of the practical benefit of hypjoin. This example is a proof that addition is associative. The definition of plus is not provided. The standard proof in figure 2-14 is fairly complicated, but each case of the induction uses only trans, symm, cong, and join proof rules. This proof becomes trivial when hypjoin is employed as shown in figure 2-15.

Define plus_assoc : Forall(x y z:nat). { (plus (plus x y) z) = (plus x (plus y z)) } :=
  induction(x:nat) by x1 x2 IH return
  Forall(y z : nat).
    { (plus (plus x y) z) = (plus x (plus y z)) }

with
  Z => forall(y z : nat).
    trans cong (plus (plus * y) z) x1
    trans join (plus (plus Z y) z) (plus y z)
    trans symm join (plus Z (plus y z)) (plus y z)
    cong (plus * (plus y z)) symm x1
| S x' => forall(y z : nat).
    trans cong (plus (plus * y) z) x1
    trans join (plus (plus (S x') y) z) (S (plus (plus x' y) z))
    trans cong (S *) [IH x' y z]
    trans symm join (plus (S x') (plus y z)) (S (plus x' (plus y z)))
    cong (plus * (plus y z)) symm x1
end.

Figure 2-14. Standard Definition of plus_assoc

The final example illustrates an unusual case that hypjoin is expected to handle correctly. Consider the case in which the programmer would like to hypjoin two terms that will not take a step of evaluation in the operational semantics. For example, the two terms could be fun terms with bodies that can be equated by hypjoin. In this example, ident is the identity function for natural numbers. A proof fragment illustrating this problem is provided in figure 2-16.
Define plus_assoc : Forall(x y z:nat). \{ (plus (plus x y) z) = (plus x (plus y z)) \} := induction(x:nat) by x1 x2 IH return

Forall(y z : nat).

\{ (plus (plus x y) z) = (plus x (plus y z)) \}

with

Z => forall(y z : nat).

hypjoin (plus (plus x y) z) (plus x (plus y z)) by x1 end

| S x' => forall(y z : nat).

hypjoin (plus (plus x y) z) (plus x (plus y z)) by x1 [IH x' y z] end

end.

Figure 2-15. Hypjoin Definition of plus_assoc

hypjoin fun r(a: nat):nat.(a n) fun r(b: nat):nat.(b (ident n)) by end

Figure 2-16 Hypjoin of Fun Terms

In order to join the terms in this example, hypjoin must evaluate within the bodies of these fun terms. If the bodies can be equated by hypjoin and the argument types and return type are equal, then the fun terms will be equated by hypjoin. The challenge presented by this example is that hypjoin must preserve the bound variables when evaluating the bodies of these terms. Since (ident n) hypjoins with n and these terms contain no bound variables, hypjoin should succeed.
3 Algorithm Description

This chapter contains a description of an algorithm that solves the problem described in chapter 2. The chapter begins with an overview of the elements of the algorithm and continues with some characteristics that are designed into the algorithm in order to make correctness easier to prove. The next section contains a formal definition of the algorithm, and implementation notes are provided in the final section.

3.1 Algorithm Overview

The algorithm is built around an operation referred to evaluation modulo $U$, which is represented as $\rightarrow_U$ where $U$ is a set of ground equations. This operation will, in essence, try to find a substitution in $U$ that allows the term to take a step of evaluation with respect to the operational semantics. In the first phase of the algorithm, all of the terms in the equations in $U$ are put into normal form with respect to $\rightarrow_U$. Any time $\rightarrow_U$ is used, it is important that the equations in $U$ are consistent - that is, substitutions in $U$ cannot allow some sort of contradiction. So this normalization of $U$ is structured in such a way that all $\rightarrow_U$ operations make use of normal, consistent $U$. In the next phase, the two supplied terms are put into normal form with respect to $\rightarrow_U$ using the normal, consistent $U$ that was calculated in the first phase. Finally, the algorithm results in a value of true if the normal forms of the supplied terms are related in $\approx_U$.

3.2 Algorithm Design Considerations

There are a few properties that are very valuable when attempting to prove that the algorithm is correct. The properties described in this section were designed in to the algorithm in order to make the proof of correctness simpler.

First, $\approx_U$ should be preserved by $\rightarrow_U$. That is, if $s \approx_U t$, $s \rightarrow_U s'$, and $t \rightarrow_U t'$, then $s' \approx_U t'$. Another way to view this property is to say that $\rightarrow_U$ is deterministic if the notion
of equality used is $\approx_U$. In order to achieve this determinism, it is necessary to evaluate the subterms of $s$ and $t$ in a specific order. The value of this determinism is that the proof that terms are joined by $\to_U$ can be a simple inductive proof that appeals to this preservation of $\approx_U$ in each step of evaluation. Without this property, it would be necessary to show that $\to_U$ is confluent[1], which would be more difficult.

It is also necessary to ensure that the ground equations are consistent in order to prevent a scenario in which the scrutinee of a match term is equivalent to the patterns of multiple cases in the term. For example, if the equations contain \{\(Z = (S \ n)\)\}, then a match on type \(\text{Nat}\) could take a step in $\to_U$ to the bodies of both cases. Because these bodies are not necessarily related by $\approx_U$, the determinism property described previously would be violated in this scenario.

Next, it is necessary for two terms to be related by $\downarrow_U$ if the terms differ only in their strict subterms and those subterms are related by $\downarrow_U$. In other words, $\overline{a} \downarrow_U \overline{b}$ implies $E_i[\overline{a}] \downarrow_U E_i[\overline{b}]$. This property is necessary for the completeness of the algorithm. In order to make this property easier to prove, $t \to_U$ will normalize all of the strict subterms of $t$ with respect to $\to_U$ before looking for a substitution in $U$ that allows $t$ to take a step in the operational semantics.

### 3.3 Algorithm Definition

This section provides a formal definition of the hypjoin algorithm.

#### 3.3.1 Evaluation Modulo U

The evaluation modulo U operation ($\to_U$) defined in figure 3-1 is used in several places in the algorithm. The equivalence relation used in $\to_U$ is more restrictive than $\approx_U$ as it only allows substitutions in evaluation contexts. This relation is denoted $\approx_U$ and is defined in figure 3-2. It is important to note that $\approx_U \subset \approx_U$. 
The definition of \( \to_U \) references a function called Holes which takes a one-level-deep extended context and returns the number of holes in that context.

Note that \( \to_U \) is considered to be a family of unary relations (that is, sets) indexed by the starting term. So \( t \to_U \) is a set of terms that can be reached in one step of \( \to_U \) starting at \( t \).

The definitions include a test that some term does not take a step in \( \to_U \) denoted \( a \not\to_U \). In terms of unary relations, \( a \not\to_U \) is defined as \( a \to_U = \emptyset \).

The relation \( \to_U \) is not considered to be a binary relation because that would result in impredicativity[6]. When considering whether \( s \to_U t \), it is necessary to test whether the
subterms of s are related to any other terms in $\rightarrow_U$. This test would be impredicative if $\rightarrow_U$ was a binary relation, because the definition would appeal to the entire relation $\rightarrow_U$ as it is being constructed. Because $\rightarrow_U$ is a family of unary relations, and because the definition of $t \rightarrow_U$ only appeals to the $\rightarrow_U$ of strict subterms of t, $t \rightarrow_U$ is predicative for all t.

### 3.3.2 Consistency

The correctness of the algorithm depends on the consistency of U. That is, U must not allow some sort of contradiction. The rules for consistency are provided in figure 3-3. All of the consistency rules must hold in order for a given U to be consistent. The abbreviation C(U) is used to describe the property that U is consistent.

\[
\begin{align*}
\text{fun}(x:!)i & \equiv_U \text{fun}(y:!).j \\
i & \equiv_U j[x/y] \\
C \approx_U D & \text{ (C and D are term constructors)}
\end{align*}
\]

\[
\begin{align*}
C & \not\approx_U F \\
\text{(C is a term constructor, F is a fun term)}
\end{align*}
\]

\[
\begin{align*}
(C\ a) & \approx_U (D\ b) \\
C & = D \\
\text{(C and D are term constructors, a and b are inactive terms)}
\end{align*}
\]

\[
\begin{align*}
(C\ a) & \not\approx_U D \\
\text{(C and D are term constructors, a is an inactive term)}
\end{align*}
\]

**Figure 3-3. Consistency Rules**

The first consistency rule related to fun term equivalence is very restrictive and essentially removes any benefit to including fun terms in U. In practice, this restriction is not very severe since the programmer can simply prove the equality of the bodies of the fun terms and place that equation in U.
3.3.3 Normalization of U

The first phase of the algorithm is the attempted normalization of the terms in the equations in U with respect to \( \rightarrow_U \). So it is necessary to define \( \rightarrow_U \) for sets of equations. This relation is defined inductively on the cardinality of U as shown in figure 3-4.

\[
\begin{align*}
0 : \emptyset & \rightarrow \emptyset \\
1 : U & \rightarrow U' \quad C(U') \\* & \rightarrow u \rightarrow_U u' \\
U & \cup \{u1 = u2\} \rightarrow U' \cup \{u1' = u2\} \\
1 & \rightarrow U' \quad \neg C(U') \\
U & \cup \{u1 = u2\} \rightarrow U' \cup \{u1' = u2\}
\end{align*}
\]

Figure 3-4. Evaluation of U

In the definition of evaluation of U, a disjoint union (\( \cup^* \)) is used to ensure that the size of U is predictable. In practice, duplicate equations can be removed at any time as they will have no impact on the outcome. Because \( m \) is always the size of U, \( m \) will often be omitted from the notation. Note that the \( \rightarrow \) symbol is overloaded. When applied to terms it refers to evaluation under the operational semantics. When it is applied to sets of ground equations, it refers to the “U evaluation” defined in this section.

3.3.4 The Hypjoin Algorithm

The algorithm which implements hypjoin is represented by the symbol \( \downarrow_U \). The definition of \( \downarrow_U \) is provided in figure 3-5.

\[
\begin{align*}
U & \rightarrow U' \quad C(U') \\
\Gamma \vdash s & \rightarrow_U s' \\
\Gamma \vdash t & \rightarrow_U t' \\
\Gamma \vdash s & \approx_U t'
\end{align*}
\]

Figure 3-5. The Hypjoin Algorithm
3.4 Algorithm Implementation Notes

The description of the hypjoin algorithm is intentionally vague in order to allow some amount of freedom when implementing it. This section provides some information that may be helpful when implementing the algorithm.

3.4.1 Equivalence Relations

The equivalence relation $=_v$ is used in the relation $\rightarrow_v$. This relation might be preferable to the more permissive equivalence relation $=_u$, because it reduces the size of the search space when trying to find equivalent terms that will take a step of evaluation. However, it would be equally correct to use $=_v$ in $\rightarrow_v$. This is because for all $s$ and $t$ $\frac{s =_v t \quad s \neq t \quad t \rightarrow}{s = t}$. In order to change a stuck term into a term that will take a step of evaluation, it is necessary to either replace the entire term or change something in an evaluation context.

3.4.2 Deciding Equivalence

It was noted earlier that $=_v$ is essentially the ground word problem for $U$. As a result, one could determine whether two terms are equivalent using congruence closure. It may also be possible to solve this problem using Knuth-Bendix completion[7] using a lexicographic path order[1] on the terms. The primary benefit derived from using completion to solve this problem is that it sets up a framework in which other implementation problems can be solved efficiently. These other problems are described in the next sections.

3.4.3 Deciding Consistency

The task of detecting inconsistency may seem daunting at first because it is necessary to consider all the terms that are equivalent to a given term. However, due to the nature of terms that can be related by $=_v$, it is only necessary to check for incompatible terms that are related by transitivity. This can be accomplished by iterating over all combinations of incompatible terms in the equations in $U$ and testing them for equivalence.
It may be possible to determine whether $U$ is consistent much more efficiently using Knuth-Bendix completion. Using completion, it is possible to orient the rewrite rules in such a way that all terms rewrite to inactive terms, if possible. Then it is possible to decide consistency simply by completing $U$ and examining the resulting rewrite rules. If $U$ is inconsistent, then there will be a rule that rewrites some term to an incompatible term.

### 3.4.4 Evaluation Modulo $U$

In order to determine if a term can take a step in $\rightarrow_U$, one might attempt to search through the entire substitution space until a redex is found. Unfortunately, this space may be infinite, so this approach will not work in general. In order to work around this problem, a limit could be specified in the implementation that will stop searching after a set amount of comparisons. Such a limit would still allow hypjoin to be sound, but it will no longer be complete. Also, this searching can be very inefficient, and it should be avoided if possible.

It may be possible to use Knuth-Bendix completion to solve this problem as well. The rewrite rules can be oriented in such a way that a term will rewrite to a redex, if possible. Then it is possible to find a redex that is equivalent to a given term, if one exists, by normalizing that term with respect to the rewrite system obtained by completing $U$. Note that this approach is compatible with the consistency test and the same order can be used in both.
4 Proof of Correctness

This chapter contains a proof that the algorithm described in the previous chapter is partially correct under certain stated assumptions. In essence, this chapter will show that the relations $\Leftrightarrow_U$ and $\downarrow_U$ are equal if $U$ is consistent. This proof is divided into two theorems.

Section 4.1 contains a theorem that states that $\downarrow_U$ is sound with respect to $\Leftrightarrow_U$ -- that is, $\downarrow_U \subseteq \Leftrightarrow_U$. Section 4.2 contains a theorem that states that $\downarrow_U$ is complete with respect to $\Leftrightarrow_U$ -- that is, $\downarrow_U \supseteq \Leftrightarrow_U$ -- if the algorithm deciding $\downarrow_U$ terminates and $U$ is consistent.

4.1 Soundness of Hypjoin Algorithm

This section contains a proof that the algorithm is sound as specified in theorem 4.1. The proof is structured as shown in the dependency graph in figure 4-1.

The most significant portion of this proof is lemma 4.1.2, which shows that if $U \rightarrow U'$ and $s \Leftrightarrow_U t$, then $s \Leftrightarrow_U t'$. That is, the algorithm does not gain any additional ability to join terms by putting $U$ in normal form.

Theorem 4.1:

$$\forall t1, \forall t2, \forall U, \forall \Gamma \quad \frac{\Gamma \rightarrow t1 \downarrow_U t2}{\Gamma \rightarrow t1 \Leftrightarrow_U t2}$$

That is,

$$\forall U', \forall t1', \forall t2' \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \rightarrow t1 \rightarrow_U t1' \quad \Gamma \rightarrow t2 \rightarrow_U t2' \quad t1' \approx_U t2'}{\Gamma \rightarrow t1 \Leftrightarrow_U t2}$$

Choose arbitrary $t1$, $t2$, $U$, $\Gamma$, $U'$, $t1'$, and $t2'$

Assume: $U \rightarrow U' \quad C(U') \quad \Gamma \rightarrow t1 \rightarrow_U t1' \quad \Gamma \rightarrow t2 \rightarrow_U t2' \quad t1' \approx_U t2'$

Derive: $\Gamma \rightarrow t1 \Leftrightarrow_U t2$

From lemma 4.1.1, $\Gamma \rightarrow t1 \Leftrightarrow_U t1'$ and $\Gamma \rightarrow t2 \Leftrightarrow_U t2'$

Because $t1' \approx_U t2'$ and JMU-EQU, $\Gamma \rightarrow t1' \Leftrightarrow_U t2'$
From transitivity, $\Gamma \triangleright t_1 \Leftrightarrow^* t_2$

From lemma 4.1.2, $\Gamma \triangleright t_1 \Leftrightarrow^*_u t_2$

![Figure 4-1. Structure of Soundness Proof](image)

Lemma 4.1.1:

$\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \frac{\Gamma \triangleright t_1 \rightarrow^*_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow^*_U t_2}$

That is: $\forall k, \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \frac{\Gamma \triangleright t_1 \rightarrow^*_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow^*_U t_2}$

Proof by induction on $k$: 
• **Base case (4.1.1.1):** \( \forall t_1, \forall t_2, \forall U, \forall \Gamma \)
\[
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

\[
\left\{ \begin{array}{c}
\forall t_1, \forall t_2, \forall U, \forall \Gamma \\
\Gamma \triangleright t_1 \rightarrow_U t_2 \\
\end{array} \right. 
\]

\[
\left\{ \begin{array}{c}
\forall t_1, \forall t_2, \forall U, \forall \Gamma \\
\Gamma \triangleright t_1 \Leftrightarrow_U t_2 \\
\end{array} \right.
\]

• **Step case (4.1.1.2):** \( \forall k \)

\[
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

\[
\left\{ \begin{array}{c}
\forall t_1, \forall t_2, \forall U, \forall \Gamma \\
\Gamma \triangleright t_1 \rightarrow_U t_2 \\
\end{array} \right. 
\]

\[
\left\{ \begin{array}{c}
\forall t_1, \forall t_2, \forall U, \forall \Gamma \\
\Gamma \triangleright t_1 \Leftrightarrow_U t_2 \\
\end{array} \right.
\]

**Lemma 4.1.1.1:**

\[
\forall t_1, \forall t_2, \forall U, \forall \Gamma 
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

Choose arbitrary \( t_1, t_2, U, \) and \( \Gamma \).

Assume: \( \Gamma \triangleright t_1 \rightarrow_U t_2 \)

Derive: \( \Gamma \triangleright t_1 \Leftrightarrow_U t_2 \)

Because \( \Gamma \triangleright t_1 \rightarrow_U t_2 \), \( t_1=t_2 \).

From EQ-REFL and JMU-EQU, \( \Gamma \triangleright t_1 \Leftrightarrow_U t_2 \)

**Lemma 4.1.1.2:**

\[
\forall k
\]

\[
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

Choose arbitrary \( k \).

Assume (A): \( \forall t_1, \forall t_2, \forall U, \forall \Gamma 
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

Derive: \( \forall t_1, \forall t_2, \forall U, \forall \Gamma 
\frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_U t_2}
\]

Choose arbitrary \( t_1, t_2, U, \) and \( \Gamma \).
Assume: $\Gamma \triangleright t_1 \rightarrow_U t_2$

Derive: $\Gamma \triangleright t_1 \leftrightarrow_U t_2$

Because, $\Gamma \triangleright t_1 \rightarrow_U t_2$, there exists $s$ such that $\Gamma \triangleright t_1 \rightarrow_U s$ and $\Gamma \triangleright s \rightarrow_U t_2$

From $A$, $\Gamma \triangleright t_1 \leftrightarrow_U s$

From lemma 4.1.1.3, $\Gamma \triangleright s \leftrightarrow_U t_2$

From transitivity, $\Gamma \triangleright t_1 \leftrightarrow_U t_2$

Lemma 4.1.1.3:
\[
\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \frac{\Gamma \triangleright t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
\]

That is: $\forall k, \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \frac{\Gamma \triangleright k : t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}$

Proof by induction on $k$:
- **Base case**: $t_1 = E_1^n[a]$  $\forall n \Gamma, Bound(E_1^n, n) \triangleright a_n \rightarrow_U$  $t_1 =_U t_3$  $\Gamma \triangleright t_3 \rightarrow t_2$
  - From JMU-EQU, $\Gamma \triangleright t_1 \rightarrow_U t_3$ and from JMU-EVAL, $\Gamma \triangleright t_3 \rightarrow_U t_2$
  - From transitivity, $\Gamma \triangleright t_1 \leftrightarrow_U t_2$
- **Step case**:
  \[
t_1 = E_1^n[a]\quad \forall n < m \quad \Gamma, Bound(E_1^n, n) \triangleright a_n \rightarrow_U \Gamma, Bound(E_1^m, m) \triangleright k : a_m \rightarrow_U a_m',
\]
  - Where the induction hypothesis (I.H.) is:
    \[
    \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \frac{\Gamma \triangleright k : t_1 \rightarrow_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
    \]
  - From EMU-IND, $t_2 = E_1^m[a_1 \ldots a_n, a_m', a_{m+1 \ldots \text{Holes}(E_1^n)\text{]}$]
  - From I.H. $\Gamma, Bound(E_1^m, m) \triangleright a_m \leftrightarrow_U a_m'$
  - From JMU_Context,
    \[
    \Gamma \triangleright E_1^n[a_1 \ldots a_n, a_m, a_{m+1 \ldots \text{Holes}(E_1^n)\text{]} \leftrightarrow_U \Gamma \triangleright E_1^n[a_1 \ldots a_n, a_m', a_{m+1 \ldots \text{Holes}(E_1^n)\text{]}\]
    - That is, $\Gamma \triangleright t_1 \leftrightarrow_U t_2$
Lemma 4.1.2:

\[ \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U} t_2} \]

That is: \( \forall j, \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{j} t_2} \)

Proof by induction on \( j \).

- **Base case:** (4.1.2.1) \( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{0} t_2} \)

- **Step case:** (4.1.2.2)

\[
\forall j \left( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{j} t_2} \right)
\]

Lemma 4.1.2.1:

\[ \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{0} t_2} \]

So \( t_1=t_2 \)

Then \( t_1 \leftrightarrow_{U} t_2 \)

Lemma 4.1.2.2:

\[
\forall j \left( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{j} t_2} \right)
\]

Choose arbitrary \( j \).

Assume: (A) \( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{j} t_2} \)

Derive: \( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{U \rightarrow U'}{\Gamma \vdash t_1 \leftrightarrow_{U}^{j+1} t_2} \)
Because $\Gamma \triangleright t_1 \leftrightarrow_{U,t2}$, there exists a $t_3$ such that $\Gamma \triangleright t_1 \leftrightarrow_{U,t3}$ and $\Gamma \triangleright t_3 \leftrightarrow_{U,t2}$.

From A, $\Gamma \triangleright t_1 \leftrightarrow_{U,t3}$.

From lemma 4.1.2.3, $\Gamma \triangleright t_3 \leftrightarrow_{U,t2}$.

From transitivity, $\Gamma \triangleright t_1 \leftrightarrow_{U,t2}$.

**Lemma 4.1.2.3:**

$$
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad U \xrightarrow{1} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2}
$$

That is: $\forall k, \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad U \xrightarrow{k} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2}$.

The notation $U \xrightarrow{k} U'$ means $U$ normalizes to $U'$ in $k$ steps.

**Proof by induction on $k$:**

- **Base case:** (4.1.2.4) $\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad U \xrightarrow{0} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2}$.

- **Step case:** (4.1.2.5)

  $$
  \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \left( \begin{array}{c}
  \forall k \quad U \xrightarrow{k} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \\
  \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad U \xrightarrow{k+1} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \\
  \end{array} \right)
  $$

**Lemma 4.1.2.4:**

$$
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad U \xrightarrow{0} U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2} \quad \Gamma \triangleright t_1 \leftrightarrow_{U,t2}
$$

So $U = U'$.

Then $\Gamma \triangleright t_1 \leftrightarrow_{U,t2}$ because $\Gamma \triangleright t_1 \leftrightarrow_{U,t2}$.
Lemma 4.1.2.5:

\[
\forall k \left( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \right)
\]

Choose arbitrary k.

Assume (A): \( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \)

Derive: \( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \)

Because \( U \rightarrow U' \), there exists a \( U'' \) such that \( U \rightarrow U'' \) and \( U'' \rightarrow_U U' \)

From lemma 4.1.2.6, \( \Gamma \triangleright t_1 \iff_U t_2 \)

From A, \( \Gamma \triangleright t_1 \iff_U t_2 \)

Lemma 4.1.2.6:

\[
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2}
\]

That is: \( \forall m, \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{m : U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \)

Proof by induction on m:

- Base case: (4.1.2.7)

\[
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{0 : U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2}
\]

- Step case: (4.1.2.8)

\[
\forall m \left( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{m : U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \right)
\]

\[
\forall m \left( \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \frac{m + 1 : U \rightarrow U' C(U')}{\Gamma \triangleright t_1 \iff_U t_2} \right)
\]
Lemma 4.1.2.7:
\[ \forall t_1, t_2, \forall U, \forall U', \forall \Gamma, \ 0 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2} \]

Because \( 0 : U \rightarrow U' \), \( U = U' = \emptyset \)

So \( \Gamma \triangleright t_1 \iff \Gamma \triangleright t_2 \), because \( \Gamma \triangleright t_1 \iff \Gamma \triangleright t_2 \)

Lemma 4.1.2.8:
\[ \forall m, \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \]
\[ \frac{m : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}}{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}} \]

Choose arbitrary \( m \).

Assume(A):
\[ \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \]
\[ \frac{m : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}}{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}} \]

Derive:
\[ \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \]
\[ \frac{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}}{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}} \]

Only the case where \( \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2 \) will be considered, the proof of the symmetric case is similar and omitted.

Derive:
\[ \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \]
\[ \frac{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}}{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}{\Gamma \triangleright t_1 \iff \Gamma \triangleright t_2}} \]

That is:
\[ \forall k, \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \]
\[ \frac{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2}{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2}}{m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2}{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2}} \]

Proof by induction on \( k \):
- Base case: \( t_1 =_{U \cdot t_2} \)
  - From lemma 4.1.2.9, \( \Gamma \triangleright t_1 \iff \Gamma \triangleright t_2 \)
- Base case: \( \Gamma \triangleright t_1 \rightarrow t_2 \)
  - From JMU-EVAL \( \Gamma \triangleright t_1 \iff \Gamma \triangleright t_2 \)
- Step case: \( t_1 = E^+_1[a_1] \quad t_2 = E^+_1[a_2] \quad \Gamma, \text{Bound}(E^+_1) \triangleright k : a_1 \Rightarrow_{U \cdot t_2} a_2 \)
  - I.H. is \( \forall t_1, t_2, \forall U, \forall U', \forall \Gamma \)
  - \( m + 1 : U \rightarrow U' \ C(U') \ \frac{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2}{\Gamma \triangleright k \iff \Gamma \triangleright t_1 \Rightarrow \Gamma \triangleright t_2} \)
Lemma 4.1.2.9:

\[
\begin{align*}
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma & \quad \frac{m : U \rightarrow U', \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2} \\
\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma & \quad \frac{m + 1 : U \rightarrow U', \quad C(U') \quad t_1 \approx_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
\end{align*}
\]

Assume (A): \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma

\[
\frac{m : U \rightarrow U', \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
\]

Derive: \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma

\[
\frac{m + 1 : U \rightarrow U', \quad C(U') \quad t_1 \approx_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
\]

That is: \forall k, \forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma

\[
\frac{m + 1 : U \rightarrow U', \quad C(U') \quad k : t_1 \approx_U t_2}{\Gamma \triangleright t_1 \leftrightarrow_U t_2}
\]

Because \( m + 1 : U \rightarrow U' \), there exist \( U_m, \ U_m', \ u_1, \ u_1', \) and \( u_2 \) such that

\( U = U_m \cup \{u_1 = u_2\}, \ u_1 \rightarrow_U u_1', \ u_2 \rightarrow_U u_2 \)

Proof by induction on \( k \):

- Base case: \( t_1 = t_2 \)
  - From JMU-EQU, \( \Gamma \triangleright t_1 \leftrightarrow_U t_2 \)
  - Base case: \( \{t_1 = t_2\} \in U' \)
    - Case \( \{t_1 = t_2\} \in U_m' \)
      - From A, \( \Gamma \triangleright t_1 \leftrightarrow_U t_2 \)
      - So \( \Gamma \triangleright t_1 \leftrightarrow_U t_2 \)
    - Case \( t_1 = u_1' \) and \( t_2 = u_2 \)
      - From lemma 4.1.1, \( \Gamma \triangleright u_1 \leftrightarrow_U u_1' \)
      - From A, \( \Gamma \triangleright u_1 \leftrightarrow_U u_1' \)
      - So \( \Gamma \triangleright u_1 \leftrightarrow_U u_1' \)
      - Because \( \{u_1 = u_2\} \in U, \ \Gamma \triangleright u_1 \leftrightarrow_U u_2 \) (EQ-SUBS, JMU-EQU)
- From transitivity, $\Gamma \triangleright u_1 \land u_2$
- That is, $\Gamma \triangleright t_1 \land t_2$

- Step case: $k : t_2 \equiv u_1 \land t_1$
  - I.H. (in all step cases) is
    $$\forall t_1, \forall t_2, \forall U, \forall U', \forall \Gamma \quad \frac{m+1 : U \rightarrow U' \quad C(U') \quad k : t_1 \equiv t_2}{\Gamma \triangleright t_1 \equiv t_2}$$
    - From I.H. $\Gamma \triangleright t_2 \equiv u_1 \land t_1$
    - From symmetry, $\Gamma \triangleright t_1 \equiv u_1 \land t_2$

- Step case: $k : t_1 \equiv u_1 \land t_3$ and $k : t_3 \equiv u_1 \land t_2$
  - From I.H. $\Gamma \triangleright t_1 \equiv u_1 \land t_3$
  - From I.H. $\Gamma \triangleright t_3 \equiv u_1 \land t_2$
  - From transitivity, $\Gamma \triangleright t_1 \equiv u_1 \land t_2$

- Step case: $t_1 = E_i^+[a_1] \quad t_2 = E_i^+[a_2] \quad k : a_1 \equiv u_1 \land a_2$
  - From I.H. $\Gamma, Bound(E_i^+) \triangleright a_1 \equiv u_1 \land a_2$
  - From JMU-CTXT, $\Gamma \triangleright t_1 \equiv u_1 \land t_2$

### 4.2 Completeness of Hypjoin Algorithm

This section contains a proof that the algorithm is complete, that is $s \equiv u \triangleright t$ implies $s \downarrow u \triangleright t$, if the algorithm terminates and $U$ does not allow inconsistency. This property is specified and proven in theorem 4.2. The structure of the proof is shown in figure 4-2.

Lemma 4.2.1 shows that if $U \rightarrow U'$ and $s \equiv u \triangleright t$, then $s \equiv u \triangleright t$. That is, joinability modulo $U$ is preserved by putting $U$ in normal form. With this lemma in place, the remainder of the proof can assume that $U$ is normal.

Lemma 4.2.3 shows that $s = u \triangleright t$ implies $s \downarrow u \triangleright t$ for normal, consistent $U$. This lemma is convenient and is therefore used frequently in the proof. Any time it is necessary to prove
that two terms are related in $\downarrow_U$, the proof can show that those two terms are related in $s \equiv_U t$.

The property that $\equiv_U$ is preserved by $\rightarrow_U$, which was designed into the hypjoin algorithm, is proven in lemma 4.2.5. This lemma is used in the proof of lemma 4.2.3.

Because $s \rightarrow t$ implies $s \Rightarrow_U t$, it is necessary to show that $s \rightarrow t$ implies $s \downarrow_U t$. This property is proven in lemma 4.2.7.

Lemma 4.2.2.6 shows that $\bar{a} \downarrow_U \bar{b}$ implies $E_{i}^*[\bar{a}] \downarrow_U E_{i}^*[\bar{b}]$. That is, two terms are related in $\downarrow_U$ if they are the same term at the top level and all of the subterms are related in $\downarrow_U$. This lemma is necessary because $\bar{a} \Rightarrow_U \bar{b}$ implies $E_{i}^*[\bar{a}] \Rightarrow_U E_{i}^*[\bar{b}]$ in the definition of $\Rightarrow_U$.

**Theorem 4.2:**

\[
\forall t_1, t_2, \forall U, \forall U', \forall \Gamma \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \Leftrightarrow_U t_2 \quad U \rightarrow_U \Gamma \leftarrow \Gamma \downarrow_U \quad \Gamma \triangleright t_1 \downarrow_U t_2
\]

Choose arbitrary $t_1, t_2, U, U'$, and $\Gamma$.

Assume: $\Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U)$

Assume: $\Gamma \triangleright t_1 \Leftrightarrow_U t_2 \quad U \rightarrow_U \Gamma \leftarrow \Gamma \downarrow_U \quad \Gamma \triangleright t_1 \downarrow_U t_2$

From Lemma 4.2.1, $\Gamma \triangleright t_1 \Leftrightarrow_U t_2$.

From Lemma 4.2.2:

\[
\forall t_1, t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \Leftrightarrow_U t_2 \quad U \not\rightarrow C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \downarrow_U t_2
\]

Because $U \not\rightarrow U'$, and $U \not\rightarrow$

From Lemma 4.2.2, $\Gamma \triangleright t_1 \downarrow_U t_2$

So $t_1 \downarrow_U t_2$ because $t_1 \downarrow_U t_2$ and $U \not\rightarrow U'$.
Figure 4-2. Structure of Completeness Proof
Lemma 4.2.1:
\[
\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2}}{\Gamma \triangleright t_1 \leftrightarrow_{U', t_2}}
\]
That is: \( \forall n, \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U^n \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2}}{\Gamma \triangleright t_1 \leftrightarrow_{U', t_2}} \)
Proof by induction on \( n \):
- Base case: (4.2.1.1) \( \forall n, \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U^n \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2}}{\Gamma \triangleright t_1 \leftrightarrow_{U', t_2}} \)
- Step case: (4.2.1.2)
\[
\forall n \quad \left( \frac{\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U^n \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2}}{\Gamma \triangleright t_1 \leftrightarrow_{U', t_2}} \right)
\]
Lemma 4.2.1.1:
\[
\forall n, \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U^{10} \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2}}{\Gamma \triangleright t_1 \leftrightarrow_{U', t_2}}
\]
Choose arbitrary \( n, U, U', t_1, t_2 \), and \( \Gamma \),
Assume: \( U^{10} \rightarrow U' \quad C(U') \quad \Gamma \triangleright t_1 \leftrightarrow_{U, t_2} \)
Derive: \( \Gamma \triangleright t_1 \leftrightarrow_{U', t_2} \)
Because \( U^{10} \rightarrow U', U=U' \)
So \( \Gamma \triangleright t_1 \leftrightarrow_{U', t_2} \) because \( \Gamma \triangleright t_1 \leftrightarrow_{U, t_2} \)

Lemma 4.2.1.2:
Choose arbitrary $n$.

Assume (A): $\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff t_2$

Derive: $\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff U \iff t_2$

Choose arbitrary $U, U', t_1, t_2, \text{ and } \Gamma$.

Assume: $U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff_U t_2$

Derive: $\Gamma \vdash t_1 \iff_U t_2$

Because $U \rightarrow U'$ and $C(U')$, there exists $U''$ such that $U \rightarrow U''$ and $U'' \rightarrow U'$ and $C(U'')$

From lemma 4.2.1.3, $\Gamma \vdash t_1 \iff_U t_2$

From A, $\Gamma \vdash t_1 \iff_U t_2$

Lemma 4.2.1.3:

$\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff_U t_2$

That is: $\forall k, \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff_U t_2$

Proof by induction on $k$

- Base case: (4.2.1.4) $\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff_U t_2$

- Step case: (4.2.1.5)

$$\forall k \begin{cases} \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \vdash t_1 \iff_U t_2 \\ \Gamma \vdash t_1 \iff_U t_2 \end{cases}$$
Lemma 4.2.1.4:
\[ \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2}{\Gamma \gg t_1 \leftrightarrow_U t_2} \]

Choose arbitrary \( U, U', t_1, t_2, \) and \( \Gamma \).

Assume: \( U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2 \)

Derive: \( \Gamma \gg t_1 \leftrightarrow_U t_2 \)

Because \( \Gamma \gg t_1 \leftrightarrow_U t_2, t_1 = t_2 \)

From EQ-REFL and JMU-EQU, \( \Gamma \gg t_1 \leftrightarrow_U t_2 \)

Lemma 4.2.1.5:
\[ \forall k \left( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2}{\Gamma \gg t_1 \leftrightarrow_U t_2} \right) \]

Choose arbitrary \( k \).

Assume(A): \( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2}{\Gamma \gg t_1 \leftrightarrow_U t_2} \)

Derive: \( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2}{\Gamma \gg t_1 \leftrightarrow_U t_2} \)

Choose arbitrary \( U, U', t_1, t_2, \) and \( \Gamma \).

Assume: \( U \rightarrow U' \quad C(U') \quad \Gamma \gg t_1 \leftrightarrow_U t_2 \)

Derive: \( \Gamma \gg t_1 \leftrightarrow_U t_2 \)

Because \( \Gamma \gg t_1 \leftrightarrow_U t_2, \) there exists \( t_3 \) such that \( \Gamma \gg t_1 \leftrightarrow_U t_3 \) and \( \Gamma \gg t_3 \leftrightarrow_U t_2 \)

From A, \( \Gamma \gg t_1 \leftrightarrow_U t_3 \)

From Lemma 4.2.1.6, \( \Gamma \gg t_3 \leftrightarrow_U t_2 \)

From transitivity, \( \Gamma \gg t_1 \leftrightarrow_U t_2 \)
Lemma 4.2.1.6:  
]\forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \triangleright t1 \iff_U t2 \\
\Gamma \triangleright t1 \iff_U t2 
This proposition will be proven for \( \Gamma \triangleright t1 \mapsto_U t2 \), the proof of \( \Gamma \triangleright t1 \iff_U t2 \) similar and omitted.

Proof by induction on the structure of \( \Gamma \triangleright t1 \mapsto_U t2 \)

- Case \( t1 \equiv_U t2 \)
  - From Lemma 4.2.1.7, \( \Gamma \triangleright t1 \iff_U t2 \)
- Case \( \Gamma \triangleright t1 \rightarrow t2 \)
  - From JMU-EVAL \( \Gamma \triangleright t1 \iff_U t2 \)
- Case \( t1 = E'_i[a1], t2 = E'_i[a2] \quad \Gamma, Bound(E'_i) \triangleright k : a1 \mapsto_U a2 \)
  - I.H. is \( \forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad \Gamma \triangleright k : t1 \mapsto_U t2 \\
\Gamma \triangleright t1 \iff_U t2 
  - From I.H. \( \Gamma, Bound(E'_i) \triangleright a1 \iff_U a2 \)
  - From JMU-CTXT \( \Gamma \triangleright t1 \iff_U t2 \)

Lemma 4.2.1.7:  
]\forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad U \rightarrow U' \quad C(U') \quad t1 \equiv_U t2 \\
\Gamma \triangleright t1 \iff_U t2 
That is, \( \forall m, \forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad m : U \rightarrow U' \quad C(U') \quad t1 \equiv_U t2 \\
\Gamma \triangleright t1 \iff_U t2 
Proof by induction on \( m \):

- Base case(4.2.1.8) \( \forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad 0 : U \rightarrow U' \quad C(U') \quad t1 \equiv_U t2 \\
\Gamma \triangleright t1 \iff_U t2 
- Step case(4.2.1.9)
  \[ \forall m \left( \forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad m : U \rightarrow U' \quad C(U') \quad t1 \equiv_U t2 \\
\Gamma \triangleright t1 \iff_U t2 \right) \]
  \[ \forall U, \forall U', \forall t1, \forall t2, \forall \Gamma \quad m + 1 : U \rightarrow U' \quad C(U') \quad t1 \equiv_U t2 \\
\Gamma \triangleright t1 \iff_U t2 \]
Lemma 4.2.1.8:
\[ \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad 0 : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2 \]
\[ \Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2 \]

Choose arbitrary U, U', t1, t2, and \(\Gamma\).
Assume: \(0 : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2\)
Derive: \(\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2\)

Because \(0 : U \rightarrow U', \ U = U' = \emptyset\)
Because \(U=U'\) and \(t_1 =_U t_2\), \(t_1 =_U t_2\)
From JMU-EQU, \(\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2\)

Lemma 4.2.1.9:
\[ \forall m\quad \left( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{m : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2} \right) \]
\[ \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad \frac{m+1 : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2}{\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2} \]

Choose arbitrary \(m\).
Assume(A): \(\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad m : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2\)
Derive: \(\forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \quad m+1 : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2\)

Choose arbitrary U, U', t1, t2, and \(\Gamma\).
Assume: \(m+1 : U \rightarrow U' \quad C(U') \quad t_1 =_U t_2\)
Derive: \(\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2\)

Because \(m+1 : U \rightarrow U'\) and \(C(U')\), there exists \(U_m, U_m', u_1, u_2, \) and \(u_1'\) such that
\[U = U_m \cup \{u_1 = u_2\}, \ m : U_m \rightarrow U_m', \ C(U_m'), \) and \(\emptyset \triangleright u_1 \rightarrow_{U_m} u_1'\)
So \(U' = U_m' \cup \{u_1' = u_2\}\)

Proof by induction on \(t_1 =_U t_2\)
- Case \(t_1 = t_2\)
  - From EQ-REFL and JMU_EQU, \(\Gamma \triangleright t_1 \Leftrightarrow_{U'} t_2\)
• Case \( \{t_1 = t_2\} \in U \)
  o Case \( \{t_1 = t_2\} \in U_m \)
    - From A, \( \Gamma \triangleright t_1 \leftrightarrow_{u_m} t_2 \)
    - So \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \)
  o Case \( t_1 = u_1 \) and \( t_2 = u_2 \)
    - Because \( \emptyset \triangleright u_1 \rightarrow_{U_m} u_1' \), and theorem 4.1, \( \emptyset \triangleright u_1 \leftrightarrow_{U_m} u_1' \), so \( \emptyset \triangleright u_1 \leftrightarrow_{U} u_1' \)
    - Because \( \{u_1' = u_2\} \in U_m, u_1' \equiv_U u_2 \) so \( \emptyset \triangleright u_1' \leftrightarrow_{U} u_2 \) from JMU-EQU
    - From transitivity, \( \emptyset \triangleright u_1 \leftrightarrow_{U} u_2 \), that is, \( \emptyset \triangleright t_1 \leftrightarrow_{U} t_2 \)
    - Because \( \emptyset \triangleright t_1 \leftrightarrow_{U} t_2 \) and \( (\text{Vars}(t_1) \cup \text{Vars}(t_2)) \cap \Gamma = \emptyset \), \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \)

• Case \( k : t_2 \equiv_U t_1 \)
  o I.H. is \( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \) \[ m : U \rightarrow U', C(U') \rightarrow \] \( k : t_1 \equiv_U t_2 \)
    \[ \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \]
  o From I.H. \( \Gamma \triangleright t_2 \leftrightarrow_{U} t_1 \)
  o From symmetry, \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \)

• Case \( k : t_1 \equiv_U t_3 \) and \( k : t_3 \equiv_U t_2 \)
  o I.H. is \( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \) \[ m : U \rightarrow U', C(U') \rightarrow \] \( k : t_1 \equiv_U t_2 \)
    \[ \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \]
  o From I.H. \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_3 \) and \( \Gamma \triangleright t_3 \leftrightarrow_{U} t_2 \)
  o From transitivity, \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \)

• Case \( t_1 = E^+_1 [a1] \) and \( t_2 = E^+_1 [a2] \) where \( k : a1 \equiv_U a2 \)
  o I.H. is \( \forall U, \forall U', \forall t_1, \forall t_2, \forall \Gamma \) \[ m : U \rightarrow U', C(U') \rightarrow \] \( k : t_1 \equiv_U t_2 \)
    \[ \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \]
  o From I.H. \( \text{Bound}(E^+_1) \triangleright a1 \leftrightarrow_{U} a2 \)
  o From JMU-CTX, \( \Gamma \triangleright t_1 \leftrightarrow_{U} t_2 \)
Lemma 4.2.2:
\[ \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]
That is:
\[ \forall k, \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]

Proof by induction on \(k\):
- Base case (4.2.2.1)
\[ \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]
- Step case (4.2.2.2)
\[ \forall k \quad \left( \begin{array}{c}
\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \\
\Gamma \triangleright t_1 \downarrow_U t_2 
\end{array} \right) \]
\[ \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]

Lemma 4.2.2.1:
\[ \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]

Choose arbitrary \(t_1, t_2, U\), and \(\Gamma\).

Assume: \( \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \)

Derive: \( \Gamma \triangleright t_1 \downarrow_U t_2 \)

So \(t_1=t_2\), so \( \Gamma \triangleright t_1 \equiv_U t_2 \) from EQ-REFL

From Lemma 4.2.3, \( \Gamma \triangleright t_1 \downarrow_U t_2 \)

Lemma 4.2.2.2
\[ \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \iff_U t_2 \quad U \not\vdash C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \]
\[ \Gamma \triangleright t_1 \downarrow_U t_2 \]

Choose arbitrary \(k\).
Assume(A):
$$\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \leftrightarrow_U t_2 \quad U \not\triangleright C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \downarrow_U t_2$$

Derive: $$\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \leftrightarrow_U t_2 \quad U \not\triangleright C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \downarrow_U t_2$$

Choose arbitrary $t_1, t_2, U,$ and $\Gamma$.
Assume: $$\Gamma \triangleright t_1 \leftrightarrow_U t_2 \quad U \not\triangleright C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U)$$
Derive: $$\Gamma \triangleright t_1 \downarrow_U t_2$$

Because $$\Gamma \triangleright t_1 \leftrightarrow_U t_2 \quad \exists t_3 \quad \Gamma \triangleright t_3 \leftrightarrow_U t_2$$
It is assumed that $$\Gamma \triangleright T(t_3, \rightarrow_U)$$
From A, $$\Gamma \triangleright t_1 \downarrow_U t_3$$
From lemma 4.2.2.3, $$\Gamma \triangleright t_3 \downarrow_U t_2$$
From lemma 4.2.8.7, $$\Gamma \triangleright t_1 \downarrow_U t_2$$

Lemma 4.2.2.3:
$$\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \leftrightarrow_U t_2 \quad U \not\triangleright C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \downarrow_U t_2$$

Case $$\Gamma \triangleright t_1 \Rightarrow_U t_2$$ (case $$\Gamma \triangleright t_1 \leftrightarrow_U t_2$$ is omitted because it is similar)
That is: $$\exists k \quad \Gamma \triangleright k : t_1 \Rightarrow_U t_2$$
Proof by induction on $k$.
- Base cases:
  - (4.2.3)
  - (4.2.7)
- Step case: (4.2.2.4)
$$\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad \Gamma \triangleright t_1 \rightarrow t_2 \quad U \not\triangleright C(U) \quad \Gamma \triangleright T(t_1, \rightarrow_U) \quad \Gamma \triangleright T(t_2, \rightarrow_U) \quad \Gamma \triangleright t_1 \downarrow_U t_2$$
Lemma 4.2.2.4:
\[
\begin{align*}
\forall k : & \quad \Gamma \vdash k : t1 \Rightarrow U, t2 \quad U \not\vdash C(U) \quad \Gamma \vdash T(t1, \rightarrow_U) \quad \Gamma \vdash T(t2, \rightarrow_U) \\
\forall U, \forall \Gamma : & \quad \Gamma \vdash t1 \downarrow_U t2 \\
\forall t1, \forall t2, \forall U, \forall \Gamma : & \quad \Gamma \vdash k + 1 : t1 \Rightarrow U, t2 \quad U \not\vdash C(U) \quad \Gamma \vdash T(t1, \rightarrow_U) \quad \Gamma \vdash T(t2, \rightarrow_U) \\
\forall U, \forall \Gamma : & \quad \Gamma \vdash t1 \downarrow_U t2
\end{align*}
\]

Choose arbitrary \( k \).

Assume (A): \( \forall t1, \forall t2, \forall U, \forall \Gamma : \Gamma \vdash k : t1 \Rightarrow U, t2 \quad U \not\vdash C(U) \quad \Gamma \vdash T(t1, \rightarrow_U) \quad \Gamma \vdash T(t2, \rightarrow_U) \)

Derive: \( \forall t1, \forall t2, \forall U, \forall \Gamma : \Gamma \vdash k + 1 : t1 \Rightarrow U, t2 \quad U \not\vdash C(U) \quad \Gamma \vdash T(t1, \rightarrow_U) \quad \Gamma \vdash T(t2, \rightarrow_U) \)

Choose arbitrary \( t1, t2, U, \) and \( \Gamma \)

Assume: \( \Gamma \vdash k + 1 : t1 \Rightarrow U, t2 \quad U \not\vdash C(U) \quad \Gamma \vdash T(t1, \rightarrow_U) \quad \Gamma \vdash T(t2, \rightarrow_U) \)

Derive: \( \Gamma \vdash t1 \downarrow_U t2 \)

Because \( \Gamma \vdash k + 1 : t1 \Rightarrow U, t2, \ t1 = E_1^+[a1] \quad t2 = E_1^+[a2] \quad \Gamma, Bound(E_1^+) \vdash k : a1 \Rightarrow_U a2 \)

From A, \( \Gamma, Bound(E_1^+) \vdash a1 \downarrow_U a2 \)

From lemma 4.2.2.6, \( \Gamma \vdash t1 \downarrow_U t2 \)

Lemma 4.2.2.6:
\[
\forall \bar{a}, \forall \bar{b}, \forall \Gamma, \forall E_1^+, \forall U : \quad \forall n \Gamma, Bound(E_1^+, n) \vdash a_n \downarrow_U b_n \quad U \not\vdash C(U) \\
\Gamma \vdash E_1^+ [\bar{a}] \downarrow_U E_1^+ [\bar{b}]
\]

Choose arbitrary \( \bar{a}, \bar{b}, \Gamma, E_1^+, \) and \( U \)

Assume: \( \forall n \Gamma, Bound(E_1^+, n) \vdash a_n \downarrow_U b_n \quad U \not\vdash C(U) \)

Derive: \( \Gamma \vdash E_1^+ [\bar{a}] \downarrow_U E_1^+ [\bar{b}] \)

Choose arbitrary \( \bar{a}' \) and \( \bar{b}' \) such that \( \forall n \Gamma, Bound(E_1^+, n) \vdash a_n \rightarrow_U a_n' \) and

\( \forall n \Gamma, Bound(E_1^+, n) \vdash b_n \rightarrow_U b_n' \) (termination is assumed).

Because \( \forall n \Gamma, Bound(E_1^+, n) \vdash a_n \downarrow_U b_n, \quad a_n' \equiv_U b_n' \)

From EQ-CTXT, \( E_1^+ [\bar{a}'] \equiv_U E_1^+ [\bar{b}'] \)

From lemma 4.2.3, \( \Gamma \vdash E_1^+ [\bar{a}'] \downarrow_U E_1^+ [\bar{b}'] \)

From lemma 4.2.2.7, \( \Gamma \vdash E_1^+ [\bar{a}] \downarrow_U E_1^+ [\bar{a}'] \) and \( \Gamma \vdash E_1^+ [\bar{b}] \downarrow_U E_1^+ [\bar{b}'] \)

From lemma 4.2.8.7, \( \Gamma \vdash E_1^+ [\bar{a}] \downarrow_U E_1^+ [\bar{b}] \)
Lemma 4.2.2.7

\[ \forall \alpha, \forall \beta, \forall \Gamma, \forall E_i^*, \forall U \quad \forall n \Gamma, \text{Bound}(E_i^*, n) \triangleright a_n \rightarrow_b U \nmid C(U) \]

That is:

\[ \forall j, \forall \alpha, \forall \beta, \forall \Gamma, \forall E_i^*, \forall U, \forall t, \forall t' \quad \forall n \Gamma, \text{Bound}(E_i^*, n) \triangleright a_n \rightarrow_b U \nmid C(U) \]

\[ \implies \left( \Gamma \triangleright E_i^*[\alpha] \rightarrow_U t \quad \Gamma \triangleright E_i^*[\beta] \rightarrow_U t' \right) \]

\[ t \equiv_U t' \]

Proof by induction on j

- **Base case:**
  \[ \forall \alpha, \forall \beta, \forall \Gamma, \forall E_i^*, \forall U, \forall t, \forall t' \quad \forall n \Gamma, \text{Bound}(E_i^*, n) \triangleright a_n \rightarrow_b U \nmid C(U) \]

  \[ \Gamma \triangleright E_i^*[\alpha] \rightarrow_U t \quad \Gamma \triangleright E_i^*[\beta] \rightarrow_U t' \]

  \[ t \equiv_U t' \]

  \[ \implies \]

  - Choose arbitrary \( \alpha, \beta, \Gamma, E_i^*, U, t, \) and \( t' \)
  - Assume \( \forall n \Gamma, \text{Bound}(E_i^*, n) \triangleright a_n \rightarrow_b U \nmid C(U) \)
  - Assume \( \alpha = \beta \) and therefore \( E_i^*[\alpha] = E_i^*[\beta] \)
  - From Eq-RefL, \( E_i^*[\alpha] = E_i^*[\beta] \)
  - From lemma 4.2.4, \( \Gamma \triangleright E_i^*[\beta] \rightarrow_U t' \)
  - Because \( \Gamma \triangleright E_i^*[\alpha] \rightarrow_U t \), \( E_i^*[\alpha] = t \)
  - Then \( t \equiv_U t' \) because \( E_i^*[\alpha] = t \)
• Step case: Prove (4.2.2.8)

\[
\forall \alpha, \forall \beta, \forall \Gamma, \forall E^i, \forall U, \forall t, \forall t', \forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b \ U \not\triangleright C(U)
\]

[Proof]

\[
\forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b, \ U \not\triangleright C(U)
\]

\[
\left( \Gamma \triangleright E^i_1[\alpha] \rightarrow_U t, \Gamma \triangleright E^i_1[\beta] \rightarrow_U t' \right)
\]

\[
t =_U t'
\]

Lemma 4.2.2.8

\[
\forall \alpha, \forall \beta, \forall \Gamma, \forall E^i, \forall U, \forall t, \forall t', \forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b \ U \not\triangleright C(U)
\]

[Proof]

\[
\forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b, \ U \not\triangleright C(U)
\]

\[
\left( \Gamma \triangleright E^i_1[\alpha] \rightarrow_U t, \Gamma \triangleright E^i_1[\beta] \rightarrow_U t' \right)
\]

\[
t =_U t'
\]

Choose arbitrary j.
Assume(A):

\[
\forall \alpha, \forall \beta, \forall \Gamma, \forall E^i, \forall U, \forall t, \forall t', \forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b \ U \not\triangleright C(U)
\]

[Proof]

\[
\forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b, \ U \not\triangleright C(U)
\]

\[
\left( \Gamma \triangleright E^i_1[\alpha] \rightarrow_U t, \Gamma \triangleright E^i_1[\beta] \rightarrow_U t' \right)
\]

\[
t =_U t'
\]

Derive:

\[
\forall \alpha, \forall \beta, \forall \Gamma, \forall E^i, \forall U, \forall t, \forall t', \forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b \ U \not\triangleright C(U)
\]

[Proof]

\[
\forall n \Gamma, \text{Bound}(E^i, n) \triangleright a \rightarrow_U b, \ U \not\triangleright C(U)
\]

\[
\left( \Gamma \triangleright E^i_1[\alpha] \rightarrow_U t, \Gamma \triangleright E^i_1[\beta] \rightarrow_U t' \right)
\]

\[
t =_U t'
\]
Choose arbitrary $\vec{a}, \vec{b}, \Gamma, E^*_i, U, t,$ and $t'$

Assume: $\forall n \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n \rightarrow^1 b_n \quad U \not
\top C(U)$

Assume: $\Gamma \triangleright E^*_i[\vec{a}] \rightarrow^1_U t \quad \Gamma \triangleright E^*_i[\vec{b}] \rightarrow^1_U t'$

Derive: $t \approx^U t'$

Because $\Gamma \triangleright E^*_i[\vec{a}] \rightarrow^1_U t$, there exists $w$ such that $\Gamma \triangleright E^*_i[\vec{a}] \rightarrow^1_U w$ and $\Gamma \triangleright w \rightarrow^1_U t$

Case split on the form of $\Gamma \triangleright E^*_i[\vec{a}] \rightarrow^1_U w$

- Case $\forall n \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n \rightarrow^1_U$ and $E^*_i[\vec{a}] = w$ $\quad \Gamma \triangleright w' \rightarrow w$
  
  o Because $\forall n \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n \rightarrow^1_U$, $\vec{a} = \vec{b}$
  
  o Then $E^*_i[\vec{a}] = E^*_i[\vec{b}]$
  
  o From lemma 4.2.3, $t \approx^U t'$

- Case $\forall n < m \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n \rightarrow^1_U \Gamma, \text{Bound}(E^*_i, m) \triangleright a_m \rightarrow^1_U a_m'$

  o Let $\vec{a}'$ be $a_{1..n}, a_{m'}, a_{m+1..\text{Holes}(E^*_i)}$

  o So $w = E^*_i[\vec{a}']$ where $\forall n \Gamma, \text{Bound}(E^*_i, n) \triangleright a_n \rightarrow^1_U b_n$

  o From A, $t \approx^U t'$

Lemma 4.2.3:

$\forall t_1, t_2, \forall U, \forall \Gamma \quad \frac{t_1 \approx^U t_2 \quad U \not \top C(U) \quad \Gamma \triangleright T(t_1, \rightarrow^1_U) \quad \Gamma \triangleright T(t_2, \rightarrow^1_U)}{\Gamma \triangleright t_1 \downarrow^1_U t_2}$

That is:

$\forall j, \forall k, \forall t_1, t_2, \forall t_1', t_2', \forall U, \forall \Gamma \quad \frac{t_1 \approx^U t_2 \quad U \not \top C(U) \quad \Gamma \triangleright t_1 \rightarrow^j_U t_1' \quad \Gamma \triangleright t_2 \rightarrow^k_U t_2'}{t_1' \approx^U t_2'}$

From Lemma 4.2.3.1, $j=k$

So it is sufficient to prove:

$\forall k, \forall t_1, t_2, \forall t_1', t_2', \forall U, \forall \Gamma \quad \frac{t_1 \approx^U t_2 \quad U \not \top C(U) \quad \Gamma \triangleright t_1 \rightarrow^k_U t_1' \quad \Gamma \triangleright t_2 \rightarrow^k_U t_2'}{t_1' \approx^U t_2'}$

Proof by induction on $k$.

- Base case (4.2.3.4)

$\forall t_1, t_2, \forall t_1', t_2', \forall U, \forall \Gamma \quad \frac{t_1 \approx^U t_2 \quad U \not \top C(U) \quad \Gamma \triangleright t_1 \rightarrow^0_U t_1' \quad \Gamma \triangleright t_2 \rightarrow^0_U t_2'}{t_1' \approx^U t_2'}$
Lemma 4.2.3.1:
\[ \forall j, \forall k, \forall U, \forall \Gamma \quad t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \]

Proof by induction on j:
- Base case (4.2.3.2):
  \[ \forall k, \forall U, \forall \Gamma \quad t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \]
  \[ k = 0 \]
- Step case (4.2.3.3):
  \[ \forall k, \forall U, \forall \Gamma \quad t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \]
  \[ j = k \]

Lemma 4.2.3.2:
\[ \forall k, \forall U, \forall \Gamma \quad t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \]
\[ k = 0 \]

Choose arbitrary k, U, \( \Gamma \), t1, t2, t1', and t2'
Assume: \( t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \)
Derive: k=0

Because \( \Gamma \triangleright t_1 \rightarrow_U t_1' \), \( \Gamma \triangleright t_1 \not\rightarrow_U \)
From contrapositive of lemma 4.2.4, \( \neg (t_1 \equiv_U t_2 \quad U \not\in C(U) \quad \exists t_2'' \quad \Gamma \triangleright t_2 \rightarrow_U t_2'') \)
Because \( t_1 \equiv_U t_2 \quad U \not\in C(U) \), \( \neg (\exists t_2'' \quad \Gamma \triangleright t_2 \rightarrow_U t_2'') \)
Therefore \( \Gamma \triangleright t_2 \rightarrow_U t_2' \)
So k=0
Lemma 4.2.3.3:
\[
\forall j, \forall k, \forall U, \forall \Gamma, \forall t_1, t_2, t_1', t_2' \\
\text{Assume: } t_1 \equiv_U t_2, U \not\models C(U) \text{ and } \Gamma \vdash t_1 \rightarrow_U t_1', \Gamma \vdash t_2 \rightarrow_U t_2'.
\]
\[
\text{Derive: } t_1 \equiv_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash t_1 \rightarrow_U t_1', \Gamma \vdash t_2 \rightarrow_U t_2'.
\]

Choose arbitrary j.
Assume (A):
\[
\forall k, \forall U, \forall \Gamma, \forall t_1, t_2, t_1', t_2' \\
\text{Assume: } t_1 \equiv_U t_2, U \not\models C(U) \text{ and } \Gamma \vdash t_1 \rightarrow_U t_1', \Gamma \vdash t_2 \rightarrow_U t_2'.
\]
\[
\text{Derive: } j + 1 = k
\]

Because \( \Gamma \vdash t_1 \rightarrow_U t_1' \), there exists \( t_1'' \) such that \( \Gamma \vdash t_1 \rightarrow_U t_1'' \) and \( \Gamma \vdash t_1'' \rightarrow_U t_1' \)

From lemma 4.2.4, \( \exists t_2'' \Gamma \vdash t_2 \rightarrow_U t_2'' \) and \( \Gamma \vdash t_2'' \rightarrow_U t_2' \)

From lemma 4.2.5, \( t_1'' \equiv_U t_2'' \)

From A, j=k-1
Then j+1=k

Lemma 4.2.3.4:
\[
\forall t_1, t_2, \forall U, \forall \Gamma \\
\text{Assume: } t_1 \equiv_U t_2, U \not\models C(U) \text{ and } \Gamma \vdash t_1 \rightarrow_U t_1', \Gamma \vdash t_2 \rightarrow_U t_2'.
\]
\[
\text{Derive: } t_1' \equiv_U t_2'
\]

Because \( \Gamma \vdash t_1 \rightarrow_U t_1' \), \( t_1'' \equiv_U t_1' \)

Because \( \Gamma \vdash t_2' \rightarrow_U t_2', t_2'' \equiv_U t_2' \)
Then \( t_1' \equiv_U t_2' \) because \( t_1 \equiv_U t_2 \)
Lemma 4.2.3.5:

\[
\forall k
\begin{cases}
    \forall t_1, t_2, t_1' \in U, \forall \Gamma, t_1 \approx \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2' \\
    t_1 \approx \Gamma \vdash t_2 \rightarrow_U t_2'
\end{cases}
\]

Choose arbitrary \( k \).

Assume (A):

\[
\forall t_1, t_2, t_1', t_2', U, \forall \Gamma, t_1 \approx \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2' \\
\]

Derive:

\[
\forall t_1, t_2, t_1', t_2', U, \forall \Gamma, t_1 \approx \Gamma \vdash t_2 \rightarrow_U t_2'
\]

Choose arbitrary \( t_1, t_2, t_1', t_2', U, \) and \( \Gamma \).

Assume: \( t_1 =_U t_2 \quad U \not\ni C(U) \quad \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2' \)

Derive: \( t_1' =_U t_2' \)

Because \( \Gamma \vdash t_1 \rightarrow_U t_1' \), there exists \( t_1'' \) such that \( \Gamma \vdash t_1 \rightarrow_U t_1'' \) and \( \Gamma \vdash t_1'' \rightarrow_U t_1' \)

Because \( \Gamma \vdash t_2 \rightarrow_U t_2' \), there exists \( t_2'' \) such that \( \Gamma \vdash t_2 \rightarrow_U t_2'' \) and \( \Gamma \vdash t_2'' \rightarrow_U t_2' \)

From lemma 4.2.5, \( t_1'' =_U t_2'' \)

From (A), \( t_1' =_U t_2' \)

Lemma 4.2.4:

\[
\forall t_1, t_2, U, \forall \Gamma, t_1 =_U t_2 \quad U \not\ni C(U) \quad \exists \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2'
\]

Choose arbitrary \( t_1, t_2, U, \) and \( \Gamma \)

Assume: \( t_1 =_U t_2 \quad U \not\ni C(U) \quad \exists \Gamma \vdash t_1 \rightarrow_U t_1' \)

Choose arbitrary \( t_1' \) such that \( \Gamma \vdash t_1 \rightarrow_U t_1' \)

Derive: \( \exists \Gamma \vdash t_2 \rightarrow_U t_2' \)

Proof by induction on the structure of \( \Gamma \vdash t_1 \rightarrow_U t_1' \)

- Case \( t_1 = E_i^n[a] \quad \forall n \Gamma, \text{Bound}(E_i^n, n) \not\ni a \not\ni U \quad t_1 =_U t_1'' \quad \Gamma \vdash t_1'' \rightarrow_U t_1' \)
  
  - From lemma 4.2.8.6, \( \exists t_2'', t_2'' =_U t_2'' \quad \Gamma \vdash t_2'' \rightarrow_U t_2'' \)
  
  - So \( \exists \Gamma \vdash t_2 \rightarrow_U t_2' \)
• Case $t_1 = E^n_i[a_{i1}, a_n, a_{n+1}]$  
  \( \Gamma, \text{Bound}(E^n_i, n) \vdash k : a_n \to_U d \)
  
  o I.H. is $\forall t, t', t, t', U, \forall \Gamma$  
   $t_1 =_U t_2$  
   $U \not\models C(U)$  
   $\Gamma \vdash t : t_1 \to_U t_1'$  
   $\exists t_2'$  
   $\Gamma \vdash t \to_U t_2'$
  
  o From lemma 4.2.8.5, $t_2 = E^n_i[a_{i1}, a_n', a_{n+1}]$ for some $\bar{a}'$
  
  o Because $t_1 =_U t_2$, $\forall n$  
   $a_n =_U a_n'$
  
  o From I.H., $\Gamma, \text{Bound}(E^n_i, n) \vdash a_n' \to_U d'$ for some $d'$
  
  o From EMU-IND, $\exists t_2'$  
   $\Gamma \vdash t \to_U t_2'$

**Lemma 4.2.5:**

$$\forall t_1, t_2, \forall t_1', t_2', \forall U, \forall \Gamma \quad t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash t_1 \to_U t_1' \quad \Gamma \vdash t_2 \to_U t_2' \quad t_1' =_U t_2'$$

That is,

$$\forall j, \forall t_1, t_2, \forall t_1', t_2', \forall U, \forall \Gamma \quad t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash j : t_1 \to_U t_1' \quad \Gamma \vdash t_2 \to_U t_2' \quad t_1' =_U t_2'$$

Proof by induction on $j$.

- Base case: (4.2.5.1)

  $$\forall t_1, \forall t_2, \forall t_1' \quad t_1 =_{E_i[\bar{a}]} \quad \forall n \Gamma, \text{Bound}(E^n_i, n) \vdash a_n \not\models_U \quad t_1 =_U t_1'' \quad \Gamma \vdash t_1'' \to t_1'$$

  $$\forall t_1', \forall U, \forall \Gamma \quad \left( t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash t_1' \to_U t_2' \right) \quad t_1' =_U t_2'$$

- Step case: (4.2.5.2)

  $$\forall j \quad \left( \forall t_1, \forall t_2, \forall t_1', \forall U, \forall \Gamma \quad t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash j : t_1 \to_U t_1' \quad \Gamma \vdash t_2 \to_U t_2' \right) \quad t_1' =_U t_2'$$

  $$\forall t_1', \forall U, \forall \Gamma \quad \left( \forall t_1, \forall t_2, \forall t_1', \forall U, \forall \Gamma \quad t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash j + 1 : t_1 \to_U t_1' \quad \Gamma \vdash t_2 \to_U t_2' \right) \quad t_1' =_U t_2'$$

**Lemma 4.2.5.1:**

$$\forall t_1, \forall t_2, \forall t_1' \quad t_1 =_{E_i[\bar{a}]} \quad \forall n \Gamma, \text{Bound}(E^n_i, n) \vdash a_n \not\models_U \quad t_1 =_U t_1'' \quad \Gamma \vdash t_1'' \to t_1'$$

$$\forall t_1', \forall U, \forall \Gamma \quad \left( t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash t_1' \to_U t_2' \right) \quad t_1' =_U t_2'$$

Choose arbitrary $t_1, t_2, t_1', t_2'$, $U$, and $\Gamma$.

Assume:

$t_1 =_{E_i[\bar{a}]} \quad \forall n \Gamma, \text{Bound}(E^n_i, n) \vdash a_n \not\models_U \quad t_1 =_U t_1'' \quad \Gamma \vdash t_1'' \to t_1'$

Assume: $t_1 =_U t_2 \quad U \not\models C(U) \quad \Gamma \vdash t_2 \to_U t_2'$

Derive: $t_1' =_U t_2'$

From lemma 4.2.8.5, $t_2 = E_i[\bar{b}]$ for some $\bar{b}$

Because $t_1 =_U t_2$, $\forall n$  
$a_n =_U b_n$
From the contrapositive of lemma 4.2.4, \( \forall n \Gamma, \text{Bound}(E^*_1, n) \triangleright b_n \rightarrow_U \)

From lemma 4.2.8.6, \( \exists t_2'' t_2 = t_2'' \Gamma \triangleright t_2'' \rightarrow t_2'' \)

Then \( \Gamma \triangleright t_2 \rightarrow_U t_2' \) by EMU-BASE only and \( t_2'' = t_2' \)

By transitivity, \( t_1'' = t_2'' \)

From lemma 4.2.8.1, \( t_1' = t_2' \)

**Lemma 4.2.5.2:**

\[
\begin{aligned}
\forall j \quad \left( \forall t_1, \forall t_2, \forall t_1', \forall U, \forall \Gamma, \ t_1 = t_2, \ U \not\triangleright C(U) \quad \Gamma \triangleright j : t_1 \rightarrow_U t_1', \ 
\Gamma \triangleright t_2 \rightarrow_U t_2' \right) \right)
\end{aligned}
\]

Choose arbitrary \( j \)

Assume (A):

\[
\left( \forall t_1, \forall t_2, \forall t_1', \forall U, \forall \Gamma, \ t_1 = t_2, \ U \not\triangleright C(U) \quad \Gamma \triangleright j : t_1 \rightarrow_U t_1', \ 
\Gamma \triangleright t_2 \rightarrow_U t_2' \right)
\]

Derive:

\[
\left( \forall t_1, \forall t_2, \forall t_1', \forall U, \forall \Gamma, \ t_1 = t_2, \ U \not\triangleright C(U) \quad \Gamma \triangleright j + 1 : t_1 \rightarrow_U t_1', \ 
\Gamma \triangleright t_2 \rightarrow_U t_2' \right)
\]

Choose arbitrary \( t_1, t_2, t_1', t_2', U \), and \( \Gamma \)

Assume: \( t_1 = t_2 \ U \not\triangleright C(U) \quad \Gamma \triangleright j : t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \)

Derive: \( t_1' = t_2' \)

Because \( \Gamma \triangleright j + 1 : t_1 \rightarrow_U t_1' \)

\( t_1 = E^*_1[a_{1...n-1}, a_n, a_{n+1...}] \) where

\( \forall m < n \Gamma, \text{Bound}(E^*_1, m) \triangleright a_m \not\rightarrow_U \) and \( \Gamma, \text{Bound}(E^*_1, n) \triangleright j : a_n \rightarrow_U d \) for some d.

So \( t_1' = E^*_1[a_{1...n-1}, d, a_{n+1...}] \)

From lemma 4.2.8.5, \( t_2 = E^*_1[\overline{a}] \) for some \( \overline{a} \)

Because \( t_1 = t_2 \), \( \forall k \quad a_k = a_k' \)

From contrapositive of lemma 4.2.4, \( \forall m < n \Gamma, \text{Bound}(E^*_1, m) \triangleright a_m' \not\rightarrow_U \)

From lemma 4.2.4, \( \Gamma, \text{Bound}(E^*_1, n) \triangleright a_n' \rightarrow_U d' \) for some d'

From A, \( d = d' \)

Because \( \forall m < n \Gamma, \text{Bound}(E^*_1, m) \triangleright a_m' \not\rightarrow_U \) and \( \Gamma, \text{Bound}(E^*_1, n) \triangleright a_n' \rightarrow_U d' \), and

EMU-IND, \( t_2' = E^*_1[a_{1...n-1}', d', a_{n+1...}] \)

From EQ-CTX, \( t_1' = t_2' \)
Lemma 4.2.7:
\[
\forall t_1, \forall t_1', \forall t_2, \forall t_2', \forall U, \forall \Gamma \quad \Gamma \vdash t_1 \rightarrow t_2 \quad U \not\triangleright C(U) \\
\left( \frac{\Gamma \vdash t_1 \rightarrow_{t_1'} t_1' \quad \Gamma \vdash t_2 \rightarrow_{t_2'} t_2'}{t_1' \approx_{t_1} t_2'} \right)
\]

Proof by induction on \( t_1 \rightarrow t_2 \)

- Base case: (4.2.7.0.1)
  \[
  \forall t_1, \forall t_1', \forall t_2, \forall t_2', \forall U, \forall \Gamma \\
  \left( \frac{\Gamma \vdash t_1 \rightarrow_{t_1'} t_1' \quad \Gamma \vdash t_2 \rightarrow_{t_2'} t_2'}{t_1' \approx_{t_1} t_2'} \right)
  \]

- Step case: (4.2.7.0.2)
  \[
  \forall n \left( \\
  \left( \frac{\Gamma \vdash n \cdot t_1 \rightarrow_{t_1'} t_1' \quad \Gamma \vdash t_2 \rightarrow_{t_2'} t_2'}{t_1' \approx_{t_1} t_2'} \right) \right)
  \]

Lemma 4.2.7.0.1:
\[
\forall t_1, \forall t_1', \forall t_2, \forall t_2', \forall U, \forall \Gamma \\
\left( \frac{\Gamma \vdash t_1 \rightarrow_{t_1'} t_1' \quad \Gamma \vdash t_2 \rightarrow_{t_2'} t_2'}{t_1' \approx_{t_1} t_2'} \right)
\]

That is:
\[
\forall j, \forall t_1, \forall t_1', \forall t_2, \forall t_2', \forall U, \forall \Gamma \\
\left( \frac{\Gamma \vdash t_1 \rightarrow_{t_1'} t_1' \quad \Gamma \vdash t_2 \rightarrow_{t_2'} t_2'}{t_1' \approx_{t_1} t_2'} \right)
\]
Proof by induction on $j$:

- **Base case:**
  \[
  \forall t_1, t_1', t_2, t_2', U, \forall \Gamma, \forall \Gamma'
  \frac{\Gamma \vdash t_1 \rightarrow t_2 \quad U \not
  \cup \quad C(U)}{t_1' \equiv_U t_2'}
  \]
  - That is $\Gamma \not \vdash t_1\not \rightarrow t_2$
  - The above contradicts $\Gamma \vdash t_1 \rightarrow t_2$
  - So conclude $t_1' \equiv_U t_2'$

- **Step case (4.2.7.1)**
  \[
  \forall j
  \frac{\Gamma \vdash t_1 \rightarrow t_2 \quad U \not
  \cup \quad C(U)}{t_1' \equiv_U t_2'}
  \]

**Lemma 4.2.7.0.2:**

\[
\forall n
\frac{\Gamma \vdash n : t_1 \rightarrow t_2 \quad U \not
  \cup \quad C(U)}{t_1' \equiv_U t_2'}
\]

Choose arbitrary $n$. 
Choose arbitrary \( t_1, t_1', t_2, t_2', U, \) and \( \Gamma \).

Assume: \( \Gamma \triangleright n+1 : t_1 \rightarrow t_2 \quad U \not\triangleright C(U) \)

Assume: \( \Gamma \triangleright t_1 \rightarrow_U t_1' \quad \Gamma \triangleright t_2 \rightarrow_U t_2' \)

Derive: \( t_1' \approx_U t_2' \)

Because \( \Gamma \triangleright n+1 : t_1 \rightarrow t_2 \), \( t_1 = E_i [r_1], t_2 = E_i [r_2], \quad \Gamma \triangleright n : r_1 \rightarrow r_2 \)

More generally, \( t_1 = E_i [a_{1..n-1}, r_1, a_{n+1...}] \) and \( t_2 = E_i [a_{1..n-1}, r_2, a_{n+1...}] \)

Because \( r_1 \) (which is in position \( n \)) is in the evaluation hole of \( E_i^+ \), \( \text{Bound}(E_i^+, n) = \emptyset \)

Consider the term \( t_3 = E_i [a_{1..n-1}', r_3, a_{n+1...}] \) where

\[
\forall m \neq n \quad \Gamma, \text{Bound}(E_i^+, m) \triangleright a_m \rightarrow_U a_m' \quad \text{and} \quad \Gamma \triangleright r_1 \rightarrow_U r_3
\]

Because \( \rightarrow_U \) normalizes all subterms first, \( \Gamma \triangleright t_1 \rightarrow_U t_3 \rightarrow_U t_1' \)

Consider the term \( t_4 = E_i [a_{1..n-1}', r_4, a_{n+1...}] \) where

\[
\forall m \neq n \quad \Gamma, \text{Bound}(E_i^+, m) \triangleright a_m \rightarrow_U a_m' \quad \text{and} \quad \Gamma \triangleright r_2 \rightarrow_U r_4
\]

Because \( \rightarrow_U \) normalizes all subterms first, \( \Gamma \triangleright t_2 \rightarrow_U t_4 \rightarrow_U t_2' \)

From A, \( r_3 \approx_U r_4 \)

From EQ-CTX, \( t_3 \approx_U t_4 \)

From lemma 4.2.3, \( t_1' \approx_U t_2' \)
Lemma 4.2.7.1:
\[
\forall t_1, t_1', t_2, t_2', U, \forall \Gamma \\
\Gamma \vdash 1: t_1 \rightarrow t_2 \quad U \not\vdash C(U) \\
\Gamma \vdash t_1 \rightarrow_U t_1', \quad \Gamma \vdash t_2 \rightarrow_U t_2' \\
\downarrow_{t_1'} \approx_U t_2'
\]

Choose arbitrary \( j \).

Assume (A):
\[
\forall t_1, t_1', t_2, t_2', U, \forall \Gamma \\
\Gamma \vdash 1: t_1 \rightarrow t_2 \quad U \not\vdash C(U) \\
\Gamma \vdash t_1 \rightarrow_U t_1', \quad \Gamma \vdash t_2 \rightarrow_U t_2' \\
\downarrow_{t_1'} \approx_U t_2'
\]

Derive:
\[
\forall t_1, t_1', t_2, t_2', U, \forall \Gamma \\
\Gamma \vdash 1: t_1 \rightarrow t_2 \quad U \not\vdash C(U) \\
\Gamma \vdash t_1 \rightarrow_U t_1', \quad \Gamma \vdash t_2 \rightarrow_U t_2' \\
\downarrow_{t_1'} \approx_U t_2'
\]

Choose arbitrary \( t_1, t_1', t_2, t_2', U, \) and \( \Gamma \).

Assume: \( \Gamma \vdash 1: t_1 \rightarrow t_2 \quad U \not\vdash C(U) \)

Assume: \( \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2' \)

Derive: \( t_1' \approx_U t_2' \)

Because \( \Gamma \vdash t_1 \rightarrow_U t_1' \), there exists \( t_3 \) such that \( \Gamma \vdash t_1 \rightarrow_U t_3 \) and \( \Gamma \vdash t_3 \rightarrow_U t_1' \)

Case split on the form of \( \Gamma \vdash t_1 \rightarrow_U t_3 \)

- EMU-BASE: \( t_1 =_U t_4 \) and \( \Gamma \vdash t_4 \rightarrow t_3 \)
  - From lemma 4.2.8.1, \( t_2 =_U t_3 \)
  - From lemma 4.2.3, \( \Gamma \vdash t_3 \downarrow_U t_3 \)
  - That is, \( \Gamma \vdash t_2 \rightarrow_U t_2' \), \( \Gamma \vdash t_3 \rightarrow_U t_1' \), and \( t_2' =_U t_1' \)
  - From EQ-SYMM, \( t_1' =_U t_2' \)
• EMU-IND: $t_1 = E_i^+ [a_1]$ where $\Gamma, \text{Bound}(E_i^+ ) \vdash a_1 \rightarrow_U a_3$
  o So $t_3 = E_i^+ [a_3]$
  o Note that if $a_1$ is inactive, then $a_3$ is the same sort of inactive term (from lemma 4.2.8.4)
  o So because $\Gamma \vdash 1 : t_1 \rightarrow t_2$, there exists $t_3'$ such that $\Gamma \vdash 1 : t_3 \rightarrow t_3'$
  o Choose arbitrary $t_3'$ such that $\Gamma \vdash 1 : t_3 \rightarrow t_3'$
  o Choose arbitrary $t_3'''$ such that $\Gamma \vdash t_3' \rightarrow_U t_3''$
  o From A, $\Gamma \vdash t_3'' \downarrow_U t_1'$
  o From 4.2.7.2, $\Gamma \vdash t_3'' \downarrow_U t_2'$
  o From lemma 4.2.8.7, $\Gamma \vdash t_1' \downarrow_U t_2'$
  o Because $N(U, \rightarrow), \Gamma \vdash N(t_1', \rightarrow_U),$ and $\Gamma \vdash N(t_2', \rightarrow_U),$ $\Gamma \vdash t_1' \downarrow_U t_2'$
    implies $t_1' \approx_U t_2'$
  o So $t_1' \approx_U t_2'$

**Lemma 4.2.7.2:**
\[
\forall E_i^+, \forall a_1, \forall a_1', \forall t_2, \forall t_2', \forall U, \forall \Gamma \quad \left( \Gamma \vdash 1 : E_i^+ [a_1] \rightarrow t_2 \quad U \not\rightarrow \quad C(U) \right) \quad \frac{\Gamma \vdash E_i^+ [a_1] \rightarrow_U E_i^+ [a_1'] \quad \Gamma \vdash E_i^+ [a_1'] \rightarrow t_2'}{\Gamma \vdash t_2 \downarrow_U t_2'}
\]

Choose arbitrary $E_i^+, a_1, a_1', t_2, t_2', U,$ and $\Gamma.$

Assume: $\Gamma \vdash 1 : E_i^+ [a_1] \rightarrow t_2 \quad U \not\rightarrow \quad C(U)$

Assume: $\Gamma \vdash E_i^+ [a_1] \rightarrow_U E_i^+ [a_1'] \quad \Gamma \vdash E_i^+ [a_1'] \rightarrow t_2'$

Derive: $\Gamma \vdash t_2 \downarrow_U t_2'$

Case split on the form of $\Gamma \vdash E_i^+ [a_1] \rightarrow^1 t_2$

• Case $E_i^+ [a_1] = \text{(fun r(x:!:):!.b I)}$
  o So $t_2 = b[I/x][\text{fun r(x:!:):!.b}/r]$
  o Case split on the form of $E_i^+ [a_1]$
    • Case $a_1 = \text{fun r(x:!:):!.b}$
      • So $a_1' = \text{fun r(x:!:):!.b'}$ where $\Gamma, x \vdash b \rightarrow_U b'$
      • So $t_2' = b'[I/x][\text{fun r(x:!:):!.b'}/r]$
      • From lemma 4.2.7.5, $\Gamma \vdash t_2 \downarrow_U t_2'$
    • Case $a_1 = I$
      • From lemma 4.2.8.4 $a_1' = I'$ where $I'$ is an inactive term
      • Then $\Gamma \vdash I \rightarrow_U I'$
      • Then $t_2' = b[I'/x][\text{fun r(x:!:):!.b'}/r]$
      • From lemma 4.2.7.4, $\Gamma \vdash t_2 \downarrow_U t_2'$
• Case \( E_1^+ [a1] =\text{match } c_n \overrightarrow{d} \text{ with } c_i \overrightarrow{x}_i \Rightarrow b_1 | ... | c_n \overrightarrow{x}_n \Rightarrow b_n | ... \text{end} \)
  
  o So \( t_2 = b_n [\overrightarrow{d} / \overrightarrow{x}_n] \)
  
  o Case split on the form of \( E_1^+ [a1] \)
    
    • Case \( a_1 = (c_n \overrightarrow{d}) \)
      
      • Because \( c_n \) has no holes, and from lemma 4.2.8.3
        
        \( \Gamma \triangleright c_n \not\triangleright_U \), then \( \exists d_n \in \overrightarrow{d}, \ \Gamma \triangleright d_n \Downarrow_U e_n \)
      
      • So let \( \overrightarrow{c} = d_{1..n-1}, e_n, d_{n+1..} \)
      
      • So \( \Gamma \triangleright \overrightarrow{d} \Downarrow_U \overrightarrow{c} \)
      
      • So \( a_1' = (c_n \overrightarrow{c}) \)
      
      • Then \( t_2'= b_n [\overrightarrow{c} / \overrightarrow{x}_n] \)
      
      • From lemma 4.2.7.4, \( \Gamma \triangleright t_2 \Downarrow_U t_2' \)
    
    • Case \( a_1 = b_n \)
      
      • So \( \Gamma, \overrightarrow{x} \triangleright b_n \Downarrow_U b_n' \)
      
      • So \( t_2' = b_n'[\overrightarrow{d} / \overrightarrow{x}_n] \)
      
      • From lemma 4.2.7.5, \( \Gamma \triangleright t_2 \Downarrow_U t_2' \)
    
    • Case \( a_1 = b_m \) where \( m \neq n \)
      
      • Then \( t_2' \) is \( b_n'[\overrightarrow{d} / \overrightarrow{x}_n] \)
      
      • Because \( t_2=t_2' \), \( \Gamma \triangleright t_2 \Downarrow_U t_2' \) from lemma 4.2.3
  
• Case \( E_1^+ [a1] =\text{let } x = I \text{ in } b \)
  
  o So \( t_2 = b[I/x] \)
  
  o Case split on the form of \( E_1^+ [a1] \)
    
    • Case \( a_1 = I \)
      
      • Then \( \Gamma \triangleright I \Downarrow_U I' \)
      
      • Then \( t_2' = b[I'/x] \)
      
      • From lemma 4.2.7.4, \( \Gamma \triangleright t_2 \Downarrow_U t_2' \)
    
    • Case \( a_1 = b \)
      
      • Then \( \Gamma, x \triangleright b \Downarrow_U b' \)
      
      • Then \( t_2' = b'[I/x] \)
      
      • From lemma 4.2.7.5, \( \Gamma \triangleright t_2 \Downarrow_U t_2' \)

Lemma 4.2.7.3:

\[
\forall a1, \forall a2, \forall U, \forall \Gamma, \forall E^+ \quad \Gamma, Bound(E^+) \triangleright a1 \Downarrow_U a2 \\
\Rightarrow \Gamma \triangleright E^+[a1] \Downarrow_U E^+[a2]
\]

(Note \( E^+ \) is the arbitrary depth context)
Choose arbitrary $a_1$, $a_2$, $U$, $E^+$, and $\Gamma$.

Assume: $\Gamma, \text{Bound}(E^+) \triangleright a_1 \downarrow_U a_2$

Derive: $\Gamma \triangleright E^+[a_1] \downarrow_U E^+[a_2]$

Proof by induction on the structure of $E^+$

- **Base case: $E^+$ is **
  - Then $\text{Bound}(E^+) = \{\}$
  - Then $\Gamma, \text{Bound}(E^+) \triangleright a_1 \downarrow_U a_2$ implies $\Gamma \triangleright E^+[a_1] \downarrow_U E^+[a_2]$
  - So $\Gamma \triangleright E^+[a_1] \downarrow_U E^+[a_2]$

- **Step case: $E^+ = E^+[E^+]$**
  - So $\Gamma, \text{Bound}(E^+), \text{Bound}(E^+^+) \triangleright a_1 \downarrow_U a_2$
  - From lemma 4.2.2.6, $\Gamma, \text{Bound}(E^+) \triangleright E^+_1[a_1] \downarrow_U E^+_1[a_2]$
  - From I.H., $\Gamma \triangleright E^+[E^+_1[a_1]] \downarrow_U E^+[E^+_1[a_2]]$
  - That is, $\Gamma \triangleright E^+[a_1] \downarrow_U E^+[a_2]$

**Lemma 4.2.7.4:**

$$\forall \overline{a}, \forall \overline{a}', \forall b, \forall \overline{x}, \forall \Gamma \quad \frac{\Gamma \triangleright \overline{a} \downarrow_U \overline{a}'}{\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b[\overline{a}' / \overline{x}]}$$

Proof by induction on $b[\overline{a} / \overline{x}]$

- **Base case $b = x_n$ and $(\text{Vars}(x_n) \cup \text{Vars}(a_n)) \cap \Gamma = \emptyset$**
  - So $b[\overline{a} / \overline{x}] = a_n$
  - Because $\Gamma \triangleright \overline{a} \downarrow_U \overline{a}'$, $\Gamma \triangleright a_n \downarrow_U a_n'$
  - Because $\Gamma \triangleright a_n \downarrow_U a_n'$ and $(\text{Vars}(x_n) \cup \text{Vars}(a_n)) \cap \Gamma = \emptyset$,
    $$(\text{Vars}(x_n) \cup \text{Vars}(a_n')) \cap \Gamma = \emptyset$$
  - So $b[\overline{a} / \overline{x}] = a_n'$
  - Because $\Gamma \triangleright a_n \downarrow_U a_n'$, $\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b[\overline{a}' / \overline{x}]$

- **Step case $b = E^+_1[\overline{c}]$** and $\forall n \quad \Gamma, \text{Bound}(E^+_1, n) \triangleright k : c_n[\overline{a} / \overline{x}] = d_n$
  - I.H. is $\forall \overline{a}, \forall \overline{a}', \forall b, \forall \overline{x}, \forall \Gamma \quad \frac{\Gamma \triangleright \overline{a} \downarrow_U \overline{a}'}{\Gamma \triangleright k : b[\overline{a} / \overline{x}] \downarrow_U b[\overline{a}' / \overline{x}]}$
  - So $b[\overline{a} / \overline{x}] = E^+_1[\overline{d}]$
  - Let $\overline{d}' = \overline{c}[\overline{a} / \overline{x}]$
  - Then $b[\overline{a} / \overline{x}] = E^+_1[\overline{d}']$ (Note: even if $b = x_n$, CAS-BASE cannot apply because $(\text{Vars}(x_n) \cup \text{Vars}(a_n)) \cap \Gamma \neq \emptyset$)
  - From I.H., $\forall n \quad \Gamma, \text{Bound}(E^+_1, n) \triangleright d_n \downarrow_U d_n'$
From lemma 4.2.2.6, $\Gamma \triangleright E_i'[\overline{d}] \downarrow_U E_i'[\overline{d}']$

That is, $\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b[\overline{a}' / \overline{x}]

Lemma 4.2.7.5:

$\forall \overline{a}, \forall b, \forall b', \forall \overline{x}, \forall \Gamma$

\[
\frac{\Gamma, \overline{x} \triangleright b \rightarrow_U b'}{
\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b'[\overline{a} / \overline{x}]
}\]

Choose arbitrary $\overline{a}, b, b', \overline{x}$, and $\Gamma$.

Assume: $\Gamma, \overline{x} \triangleright b \rightarrow_U b'$

Derive: $\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b'[\overline{a} / \overline{x}]

Then there exists $E^+, c$, and $c'$ such that $b = E^+[c]$ and $\Gamma, \overline{x}, Bound(E^+) \triangleright c \rightarrow_U c'$

Note that $E^+$ is the “deepest” possible context that fits the above description.

So $b' = E^+[c']$

In terms of $E^+[c]$, $b[\overline{a} / \overline{x}] = (E^+[c])[\overline{a} / \overline{x}]$

Because $\Gamma, \overline{x}, Bound(E^+) \triangleright c \rightarrow_U c'$ and $E^+$ is the deepest possible context, none of $\overline{x}$ appear in $c$ or $c'$

So $b[\overline{a} / \overline{x}] = (E^+[c])[\overline{a} / \overline{x}] = (E^+[\overline{a} / \overline{x})[c]$

By a similar argument, $b'[\overline{a} / \overline{x}] = (E^+[c'])[\overline{a} / \overline{x}] = (E^+[\overline{a} / \overline{x})[c']$

Substitution cannot affect the “binding” variables in a context, so $Bound(E^+) = Bound(E^+[\overline{a} / \overline{x})]$

Then $\Gamma, \overline{x}, Bound(E^+[\overline{a} / \overline{x})] \triangleright c \rightarrow_U c'$

Then $\Gamma, \overline{x}, Bound(E^+[\overline{a} / \overline{x})] \triangleright c \downarrow_U c'$

From lemma 4.2.7.3, $\Gamma, \overline{x} \triangleright (E^+[\overline{a} / \overline{x})[c] \downarrow_U (E^+[\overline{a} / \overline{x})[c']$

Because neither of the terms above contain any of the variables in $\overline{x}$,

$\Gamma \triangleright (E^+[\overline{a} / \overline{x})[c] \downarrow_U (E^+[\overline{a} / \overline{x})[c']$

That is, $\Gamma \triangleright b[\overline{a} / \overline{x}] \downarrow_U b'[\overline{a} / \overline{x}]$

Lemma 4.2.8.1

$\forall t_1, \forall t_2, \forall t_1', \forall t_2'

\frac{t_1 \equiv_U t_2 \quad \Gamma \triangleright t_1 \rightarrow t_1' \quad \Gamma \triangleright t_2 \rightarrow t_2'}{
\forall U, \forall \Gamma

\frac{t_1 \equiv_U t_2 \quad \Gamma \triangleright k : t_1 \rightarrow t_1' \quad \Gamma \triangleright t_2 \rightarrow t_2'}{U \not\leftrightarrow C(U)\quad \frac{t_1' \equiv_U t_2'}{U \not\leftrightarrow C(U)}}}$

Proof by induction on $\Gamma \triangleright t_1 \rightarrow t_1'$

- Case $t_1 = E_i[a_1]$ where $\Gamma \triangleright k : a_1 \rightarrow a_1'$
  - I.H. is

\[
\forall t_1, \forall t_2, \forall t_1', \forall t_2'

\frac{t_1 \equiv_U t_2 \quad \Gamma \triangleright k : t_1 \rightarrow t_1' \quad \Gamma \triangleright t_2 \rightarrow t_2'}{
\forall U, \forall \Gamma

\frac{t_1 \equiv_U t_2 \quad \Gamma \triangleright k : t_1 \rightarrow t_1' \quad \Gamma \triangleright t_2 \rightarrow t_2'}{U \not\leftrightarrow C(U)\quad \frac{t_1' \equiv_U t_2'}{U \not\leftrightarrow C(U)}}}$


From lemma 4.2.8.5, \( t_1 = E_1^*[b_1...b_{n-1}, a_1, b_{n+1...Holes(E_1^*)}] \) and \( t_2 = E_1^*[d_1...d_{n-1}, a_2, d_{n+1...Holes(E_1^*)}] \) for some \( \vec{b} \) and \( \vec{d} \)

Then \( a_1 \approx_U a_2 \)

Case split on the form of \( E_1^* \)
- Case \( t_1=(a_1 \, g_1) \) and \( t_2=(a_2 \, g_2) \)
  - So \( t_1' = (a_1' \, g_1) \)
  - Case split on the form of \( \Gamma \triangleright t_2 \rightarrow t_2' \)
    - Case \( \Gamma \triangleright a_2 \rightarrow a_2' \)
      - From I.H. \( a_1' \approx_U a_2' \)
      - So \( t_2' = (a_2' \, g_2) \)
      - From EQ-CTXT \( t_1' \approx_U t_2' \)
    - Case \( a_2 \in I \) and \( \Gamma \triangleright g_2 \rightarrow g_2' \)
      - From lemma 4.2.8.3,
        \( \Gamma \triangleright \neg (\exists a_1'' \approx_U a_1 \triangleright a_1'' \rightarrow) \)
      - The previous statement contradicts \( \Gamma \triangleright a_1 \triangleright a_1' \)
      - So conclude \( t_1' \approx_U t_2' \)
  - Case \( a_2=\text{fun} \, x(a!):!t \) and \( g_2 \in I \)
    - From lemma 4.2.8.3,
      \( \Gamma \triangleright \neg (\exists a_1'' \approx_U a_1 \triangleright a_1'' \rightarrow) \)
    - The previous statement contradicts \( \Gamma \triangleright a_1 \triangleright a_1' \)
    - So conclude \( t_1' \approx_U t_2' \)
- Case \( t_1=(f_1 \, a_1) \) and \( t_2=(f_2 \, a_2) \) where \( f_1 \in I \)
  - So \( t_1' = (f_1 \, a_1') \)
  - Case split on the form of \( \Gamma \triangleright t_2 \rightarrow t_2' \)
    - Case \( \Gamma \triangleright f_2 \rightarrow f_2' \)
      - From lemma 4.2.8.3,
        \( \Gamma \triangleright \neg (\exists f_2'' \approx_U f_2 \triangleright f_2'' \rightarrow) \)
      - The previous statement contradicts \( \Gamma \triangleright f_2 \triangleright f_2' \)
      - So conclude \( t_1' \approx_U t_2' \)
    - Case \( f_2 \in I \) and \( \Gamma \triangleright a_2 \rightarrow a_2' \)
      - From I.H. \( a_1' \approx_U a_2' \)
      - So \( t_2' = (f_2 \, a_2') \)
      - Then \( t_1' \approx_U t_2' \)
    - Case \( f_2=\text{fun} \, x(a!):!t \) and \( a_2 \in I \)
From lemma 4.2.8.3,
\[ \Gamma \vdash \neg (\exists a_l'' =_U a_l \ a_l'') \]
- The previous statement contradicts
\[ \Gamma \vdash a_l \to a_l' \]
- So conclude \( t_1' \approx_U t_2' \)

- Case \( t_1 = \text{match } a_1 \text{ with } c_i x_i \Rightarrow s_1 \ldots \ c_n x_n \Rightarrow s_n \text{ end} \)
  - So \( t_2 = \text{match } a_2 \text{ with } c_i y_i \Rightarrow v_1 \ldots \ c_n y_n \Rightarrow v_n \text{ end} \)
  - Then \( t_1' = \text{match } a_1' \text{ with } c_i x_i \Rightarrow s_1 \ldots \ c_n x_n \Rightarrow s_n \text{ end} \)
  - Case split on the form of \( \Gamma \vdash t_2 \to t_2' \)
    - Case \( \Gamma \vdash a_2 \to a_2' \)
      - From I.H. \( a_1' =_U a_2' \)
      - So \( t_2' = \text{match } a_2' \text{ with } c_i y_i \Rightarrow v_1 \ldots \ c_n y_n \Rightarrow v_n \text{ end} \)
      - From EQ-CTX \( t_1' \approx_U t_2' \)
    - Case \( a_2 = (c_i \overline{t}) \)
      - From lemma 4.2.8.3,
        \[ \Gamma \vdash \neg (\exists a_l'' =_U a_l \ a_l'') \]
      - The previous statement contradicts
        \[ \Gamma \vdash a_l \to a_l' \]
      - So conclude \( t_1' \approx_U t_2' \)

- Case \( t_1 = \text{let } x_1 = a_1 \text{ in } b_1 \)
  - So \( t_2 = \text{let } x_2 = a_2 \text{ in } b_2 \)
  - Then \( t_1' = \text{let } x_1 = a_1' \text{ in } b_1 \)
  - Case split on the form of \( \Gamma \vdash t_2 \to t_2' \)
    - Case \( \Gamma \vdash a_2 \to a_2' \)
      - From I.H. \( a_1' =_U a_2' \)
      - So \( t_2' = \text{let } x_2 = a_2' \text{ in } b_2 \)
      - From EQ-CTX \( t_1' \approx_U t_2' \)
    - Case \( a_2 \in I \)
      - From lemma 4.2.8.3,
        \[ \Gamma \vdash \neg (\exists a_l'' =_U a_l \ a_l'') \]
      - The previous statement contradicts
        \[ \Gamma \vdash a_l \to a_l' \]
      - So conclude \( t_1' \approx_U t_2' \)

- Case \( t_1 = (F \ I) \text{ where } F = \text{fun } x(a!):!t \)
  - So \( t_1' = t[I / a] [F/x] \)
  - From lemma 4.2.8.5, \( t_2 = (F' \ I') \) and \( F =_U F' \) and \( I =_U I' \)
  - From lemma 4.2.8.3, \( \Gamma \vdash F' \not\in I' \) and \( \Gamma \vdash I' \not\in F' \)
o Because $\Gamma \triangleright t_2 \rightarrow t_2'$ and all the sub-terms of $t_2$ don’t take a step in $\rightarrow$, $F'$ is a fun term

o Because $C(U)$, and $F \approx_U F'$, $F'$ = fun $x'(a'!):!t'$ where $t' \approx_U t[a'/a]

o So $t_2' = t'[I'/a'][F'/x']$

o Because $t' \approx_U t[a'/a]$, $I \approx_U I'$, and $F \approx_U F'$,
$t'[I'/a'][F'/x'] \approx_U t[I/a][F/x]$

o From EQ-SYMM, $t[I/a][F/x] \approx_U t'[I'/a'][F'/x']$

o That is, $t_1' \approx_U t_2'$

• Case $t_1 = $match $(c_i \overline{T})$ by $x$ return $!$ with
$c_i \overline{x}_i \rightarrow s_1 | \ldots | c_i \overline{x}_n \rightarrow s_n$. 

o So $t_1' = s_1[\overline{t} / \overline{x}_i]$

o From lemma 4.2.8.5, $t_2 = $match $a$ by $x$ return $!$ with
$c_i \overline{x}_i \rightarrow s_1 | \ldots | c_i \overline{x}_n \rightarrow s_n$. 
Where $a \approx_U (c_i \overline{T})$ and $\forall n \ s_n \approx_U s_n'$.

o Because $C(U)$ and $a \approx_U (c_i \overline{T})$, $\forall a' \approx_U a$ where $a' = (f \overline{t})$, $f = c_i$. From lemma 4.2.8.3, $\forall a'$ where $a' \approx_U a$, $\Gamma \triangleright a'$ does not take a step, so none of $\overline{t}$ will take a step in $\rightarrow$. Because $a' \approx_U a$, $\overline{t} \approx_U \overline{t}$

o So $t_2' = s_1'[\overline{t} / \overline{x}_i]$

o Because $\overline{t} \approx_U \overline{t}$, and $s_1 \approx_U s_1'$, $t_1' \approx_U t_2'$ by EQ-CTX

• Case $t_1 = $let $x = I$ by $y$ in $t$

o So $t_1' = t[I/x]$

o From lemma 4.2.8.5, $t_2 = $let $x' = I'$ by $y'$ in $t'$ where $I \approx_U I'$ and $t[x'/x] \approx_U t'$

o So $t_2' = t'[I'/x']$

o So $t_1' \approx_U t_2'$ by EQ-CTX

• Case $t_1 = E[abort !]$

o So $t_1' = abort !$

o Then $t_2 = E[abort !]$

o So $t_2' = abort !$

o So $t_1' \approx_U t_2'$

Lemma 4.2.8.2

\[\forall t_1, \forall t_2, \forall U, \forall \overline{a}, \forall \Gamma \quad t_1 \approx_U t_2 \quad t_1 = E_i'[\overline{a}] \\
\exists \overline{b} \quad t_2 = E_i'[\overline{b}]\]

Proof by induction on $t_1 \approx_U t_2$

• Case $t_1 = t_2$

o Let $\overline{b} = \overline{a}$
Then $t_2 = E_i^*[\bar{a}]$

- Case $\{t_1 = t_2\} \in U$
  - Let $\bar{b} = \bar{a}$
  - Then $t_2 =_U E_i^*[\bar{a}]$

- Case $k : t_2 =_U t_1$
  - I.H. is $\forall t_1, \forall t_2, \forall U, \forall \bar{a}, \forall \Gamma$
    
  \[
  \frac{k : t_1 =_U t_2 \quad t_1 = E_i^*[\bar{a}]}{\exists \bar{b} \quad t_2 =_U E_i^*[\bar{b}]}
  \]

  - From I.H., $\exists \bar{b} \quad t_2 =_U E_i^*[\bar{b}]$

- Case $\exists t_3 \quad k : t_1 =_U t_3 \quad k : t_3 =_U t_2$
  - I.H. is $\forall t_1, \forall t_2, \forall U, \forall \bar{a}, \forall \Gamma$
    
  \[
  \frac{k : t_1 =_U t_2 \quad t_1 = E_i^*[\bar{a}]}{\exists \bar{b} \quad t_2 =_U E_i^*[\bar{b}]}
  \]

  - From I.H., $\exists \bar{b} \quad t_2 =_U E_i^*[\bar{b}]$

  - Choose arbitrary $E_i^*[\bar{b}]$ where $t_3 =_U E_i^*[\bar{b}]$

  - From I.H., $\exists \bar{b}' \quad t_2 =_U E_i^*[\bar{b}']$

- Case $t_1 = E_i^*[a_1] \quad t_2 = E_i^*[a_2] \quad k : a_1 =_U a_2$
  - So $t_1 = E_i^*[\bar{c}, a_1, \bar{d}]$ and $t_2 = E_i^*[\bar{c}, a_2, \bar{d}]$

  - So let $\bar{b}$ be $\bar{c}, a_2, \bar{d}$

  - Then $t_2 =_U E_i^*[\bar{b}]$

**Lemma 4.2.8.2.1**

\[
\forall t_1, \forall t_2, \forall U, \forall \bar{a}, \forall \Gamma
\quad
\frac{t_1 =_U t_2 \quad t_1 = (c \bar{a})}{\exists \bar{b} \quad t_2 =_U (c \bar{b})}
\]

From lemma 4.2.8.2, $\exists d, \bar{d}$ such that $t_2 =_U (d \bar{d})$

So $d =_U c$

From lemma 4.2.8.2, $d =_U c$

So $t_2 =_U (c \bar{d})$

**Lemma 4.2.8.2.2**

\[
\forall t_1, \forall t_2, \forall U, \forall \Gamma
\quad
\frac{t_1 =_U t_2 \quad t_1 \in I}{\exists \bar{t}' \in I \quad t_2 =_U \bar{t}'}
\]

Note $I$ is the set of inactive terms

$t_1 = E_i^*[\bar{a}]$ for some $\bar{a}$

From lemma 4.2.8.2, $\exists \bar{b} \quad t_2 =_U E_i^*[\bar{b}]$

Proof by induction on $E_i^*[\bar{a}]$

- Case $t_1 = (C \bar{a})$ where $C$ is a constant
I.H. is \[ \forall n, \forall b, \forall U, \forall \Gamma \quad \frac{a_n \approx_U b \quad a_n \in I}{\exists b' \in I \quad b' =_U b} \]

- From lemma 4.2.8.2.1, \( \exists \overline{b} \) \( t_2 =_U (C \overline{b}) \)
- From I.H. \( \forall n \quad \exists b_n ' \in I \quad b_n ' =_u b_n \)
- So \( (C \overline{b}) \in I \), let \( t_2' = (C \overline{b}) \)

- **Case otherwise**
  - All other inactive \( E_1^* \) will be inactive regardless of their subterms
  - So \( \exists t_2' \in I \quad t_2' =_U t_2 \) from lemma 4.2.8.2

**Lemma 4.2.8.3**

\[
\forall t_1, \forall t_2, \forall U, \forall \Gamma \quad\begin{array}{c}
U \not\models C(U) \\
\vdash t_1 =_U t_2 \quad t_1 \in I \\
\vdash t_2' =_U t_2' \\
\Gamma \not\models t_2' \not\models
\end{array}
\]

Note I is the set of inactive terms

Choose arbitrary \( t_2' \) such that \( t_2' =_U t_2 \)

(Proof by contradiction)

- Assume \( \Gamma \not\models t_2' \not\models \)

Because \( t_2' =_U t_1 \) and lemma 4.2.8.5, \( t_1 = E_1^*[\overline{a}] \) and \( t_2' = E_1^*[\overline{b}] \) for some \( \overline{a}, \overline{b} \)

Proof by induction on \( E_1^*[\overline{a}] \)

- **Case** \( t_1 = (c_1 a_1) \) where \( c_1 \) is a constant
  - So \( t_2' = (f_2 a_2) \)
  - Case split on the form of \( \Gamma \not\models t_2' \not\models \)
    - **Case** \( \Gamma \not\models f_2 \not\models \)
      - From I.H. \( \Gamma \not\models f_2 \not\models \)
      - Contradiction, conclude \( \Gamma \not\models t_2' \not\models \)
    - **Case** \( f_2 \in I \) and \( a_2 \not\models \)
      - From I.H. \( \Gamma \not\models a_2 \not\models \)
      - Contradiction, conclude \( \Gamma \not\models t_2' \not\models \)
    - **Case** \( f_2 = \text{fun } x(y!):!.d \) and \( a_2 \in I \)
      - Contradicts \( C(U) \) because \( c_1 \approx_U \text{fun } x(y!):!.d \)
      - So conclude \( \Gamma \not\models t_2' \not\models \)

- **Case otherwise**
  - All other inactive \( E_1^* \) will be inactive regardless of their subterms
  - So \( t_2' \in I \)
  - Therefore \( \Gamma \not\models t_2' \not\models \)
In all cases, we get something that contradicts $\Gamma \triangleright t2' \rightarrow$
So $\Gamma \triangleright t2' \not\rightarrow$

Lemma 4.2.8.4:
$$\forall t1, \forall t2, \forall U, \forall \Gamma \quad U \not\triangleright C(U) \quad \Gamma \triangleright t1 \rightarrow t2 \quad t1 \in I \quad t2 \in I$$

Induction on $\Gamma \triangleright t1 \rightarrow t2$
- Case $t1 = E_i^+[a1], \: \Gamma, Bound(E_i^+) \triangleright k : a1 \rightarrow a2, \: t2 = E_i^+[a2]$
  - I.H. is $\forall t1, \forall t2, \forall U, \forall \Gamma \quad U \not\triangleright C(U) \quad \Gamma \triangleright k : t1 \rightarrow t2 \quad t1 \in I \quad t2 \in I$
  - Case split on the form of $E_i^+[a1]$
    - Case $E_i^+[a1] = (c \overline{I})$ where $c$ is a constant and $\overline{I}$ is a list of inactive terms
      - From lemma 4.2.8.3, and because $c$ has no holes, $c \not\triangleright U$
      - So some $I_n \in \overline{I}$ is $a1$
      - From I.H., $a2 \in \overline{I}$
      - So $t2 = (c \overline{I}')$ where $\overline{I}'$ is a list of inactive terms
      - So $t2 \in I$
    - Case otherwise
      - All other $E_i^+$ are inactive regardless of the subterms
      - So $t2 \in I$
- Case $t1 =_U t2$ and $\Gamma \triangleright t3 \rightarrow t2$
  - From lemma 4.2.8.3, this $t3$ cannot exist
  - This case results in a contradiction, conclude $t2 \in I$

Lemma 4.2.8.5
$$\forall t1, \forall t2, \forall U, \forall \Gamma \quad U \not\triangleright t1 \approx U t2 \quad \Gamma \triangleright t1 \rightarrow t2 \quad \exists E_i^+, \overline{b}, \overline{b}' \quad t1 = E_i^+[\overline{b}] \quad t2 = E_i^+[\overline{b}']$$

Proof by induction on $t1 \approx_U t2$:
- Case $t1 = t2$
  - Then trivially, $\exists E_i^+, \overline{b}, \overline{b}' \quad t1 = E_i^+[\overline{b}] \quad t2 = E_i^+[\overline{b}']$
- Case $\{t1 = t2\} \in U$
  - Contradicts $U \not\triangleright$
  - So conclude $\exists E_i^+, \overline{b}, \overline{b}' \quad t1 = E_i^+[\overline{b}] \quad t2 = E_i^+[\overline{b}']$
- Case $k : t2 =_U t1$
\[
\begin{align*}
&\text{• I.H. is } \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad N(U, \rightarrow) \quad k : t_1 \approx_U t_2 \quad \Gamma \triangleright t_1 \rightarrow_U t_2 \\
&\quad \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b') \\
&\quad \text{From I.H. } \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b') \\
&\quad \text{Case } \exists t_3 \quad k : t_1 \approx_U t_3 \quad k : t_3 \approx_U t_2 \\
&\quad \text{I.H. is } \forall t_1, \forall t_2, \forall U, \forall \Gamma \quad N(U, \rightarrow) \quad k : t_1 \approx_U t_2 \quad \Gamma \triangleright t_1 \rightarrow_U t_2 \\
&\quad \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b') \\
&\quad \text{From I.H. } \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b') \\
&\quad \text{From I.H. } \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b) \\
&\quad \text{Because } t_3 = E'(b) \text{ and } t_3 = E'(b) \text{, } E' = E' \\
&\quad \text{Let } b' = b' \\
&\quad \text{So } \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b) \\
&\quad \text{Case } t_1 = E'[a] \quad t_2 = E'[a] \quad a_1 \approx_U a_2 \\
&\quad \text{So } t_1 = E'[b, a_1, a] \text{ and } t_2 = E'[b, a_2, a] \\
&\quad \text{So } \exists E'(b, b') \quad t_1 = E'(b) \quad t_2 = E'(b) \\
\end{align*}
\]

Lemma 4.2.8.6:

\[
\forall t_1, \forall t_1', \forall t_2, \forall U, \forall \Gamma \quad U \not\vdash C(U) \quad t_1 \approx_U t_2 \quad t_1' \approx_U t_2' \quad \Gamma \triangleright t_1' \rightarrow t_2' \\
\exists t_2' \approx_U t_2 \quad \Gamma \triangleright t_2' 
\]

Proof by induction on } t_1 \approx_U t_2:

• Case } t_1 = t_2:
  • So } t_1 \approx_U t_2 \text{ and from transitivity, } t_1' \approx_U t_2 \\
  • Let } t_2' = t_1' \\
  • Then } t_2' \approx_U t_2' \quad \Gamma \triangleright t_2' \\

• Case } \{t_1 = t_2\} \in U:
  • So } t_1 \approx_U t_2 \text{ and from transitivity, } t_1' \approx_U t_2 \\
  • Let } t_2' = t_1' \\
  • Then } t_2' \approx_U t_2' \quad \Gamma \triangleright t_2' \\

• Case } k : t_2 \approx_U t_1:
  • I.H. is
    \[
    \forall t_1, \forall t_1', \forall t_2, \forall U, \forall \Gamma \quad U \not\vdash C(U) \quad k : t_1 \approx_U t_2 \quad t_1' \approx_U t_2' \quad \Gamma \triangleright t_1' \rightarrow t_2' \\
    \exists t_2' \approx_U t_2 \quad \Gamma \triangleright t_2' 
    \]
  • From I.H. } \exists t_2' \approx_U t_2 \quad \Gamma \triangleright t_2' \\

• Case } \exists t_3 \quad k : t_1 \approx_U t_3 \quad k : t_3 \approx_U t_2:
  • I.H. is
    \[
    \forall t_1, \forall t_1', \forall t_2, \forall U, \forall \Gamma \quad U \not\vdash C(U) \quad k : t_1 \approx_U t_2 \quad t_1' \approx_U t_2' \quad \Gamma \triangleright t_1' \rightarrow t_2' \\
    \exists t_2' \approx_U t_2 \quad \Gamma \triangleright t_2' 
    \]
From I.H. \[ \exists t^3 \vdash_t t^3 \] \[ \Gamma \vdash t^3 \rightarrow \]

- Choose arbitrary \( t^3 \) such that \( t^3 \vdash_t t^3 \) and \( \Gamma \vdash t^3 \rightarrow \)

- From I.H. \[ \exists t^2 =_t t^2 \] \[ \Gamma \vdash t^2 \rightarrow \]

- Case \( t_1 = E^+_1[a] \) \[ t_2 = E^+_1[a2] \] \[ k : a1 =_t a2 \]

- I.H. is

\[
\forall t_1, \forall t_1', \forall t_2, \forall U, \forall \Gamma.U \not\vdash C(U) \quad k : t_1 =_t t_2 \quad t_1 =_t t_1' \quad \Gamma \vdash t_1' \rightarrow \exists t_2 =_t t_2 \quad \Gamma \vdash t_2' \rightarrow
\]

- Case split on the form of \( t_1 \)
  - Case \( t_1 = (a_1 b) \)
    - Then \( t_2 = (a_2 b) \)
    - From lemma 4.2.8.5, \( t_1' = (a_1'' b_1'') \)
    - Case split on the form of \( \Gamma \vdash t_1' \rightarrow \)
      - Case \( \Gamma \vdash a_1'' \rightarrow a_1' \)
        - From I.H. \( \exists a_2'' =_t a_2 \) \[ \Gamma \vdash a_2'' \rightarrow \]
        - So, \( t_2 =_t (a_2'' b) \)
        - Let \( t_2' = (a_2'' b) \)
        - So \( \exists t_2' =_t t_2 \quad \Gamma \vdash t_2' \rightarrow \)
      - Case \( a_1'' \in I \) and \( \Gamma \vdash b_1'' \rightarrow b_1' \)
        - From lemma 4.2.8.2.2, \[ \exists a_2'' \in I \quad a_2'' =_t a_2 \]
        - So \( t_2 =_t (a_2'' b_1'') \)
        - Let \( t_2' = (a_2'' b_1'') \)
        - So \( \exists t_2' =_t t_2 \quad \Gamma \vdash t_2' \rightarrow \)
    - Case \( a_1'' = \text{fun } r(x!) : !y \) and \( b_1'' \in I \)
      - From lemma 4.2.8.2, \[ \exists a_2'' =_t a_2 \quad \text{where } \]
        \[ a_2'' = \text{fun } r(x!) : !y \]
      - So \( t_2 =_t (a_2'' b_1'') \)
      - Let \( t_2' = (a_2'' b_1'') \)
      - So \( \exists t_2' =_t t_2 \quad \Gamma \vdash t_2' \rightarrow \)
  - Case \( t_1 = (f a_1) \)
    - Then \( t_2 = (f a_2) \)
    - From lemma 4.2.8.5, \( t_1' = (f_1'' a_2'') \)
    - Case split on the form of \( \Gamma \vdash t_1' \rightarrow \)
      - Case \( \Gamma \vdash f_1'' \rightarrow f' \)
        - So \( t_2 =_t (f_1'' a_2) \)
        - Let \( t_2' = (f_1'' a_2) \)
        - So \( \exists t_2' =_t t_2 \quad \Gamma \vdash t_2' \rightarrow \)
      - Case \( f_1'' \in I \) and \( a_1'' \rightarrow a_1' \)
- From I.H. \( \exists a_2'' =_u a_2 \quad \Gamma \triangleright a_2'' \rightarrow \)
- So \( t_2 =_u (f' a_2'') \)
- Let \( t_2' = (f' a_2'') \)
- So \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
  - Case \( t_1'' = \text{fun } r(x!):!y \) and \( a_1'' \in I \)
    - From lemma 4.2.8.2, \( \exists a_2'' \in I \quad a_2'' =_u a_2 \)
    - So \( t_2 =_u (f' a_2'') \)
    - Let \( t_2' = (f' a_2'') \)
    - So \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
- Case \( t_1 = \text{fun } (x!):!a \)
  - From lemma 4.2.8.5, \( t_1' = \text{fun } (x!):!a_1'' \) for some \( a_1'' \)
  - Then \( \Gamma \triangleright t_1' \not\rightarrow \)
  - The above conclusion contradicts \( \Gamma \triangleright t_1' \rightarrow \)
  - Conclude \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
- Case \( t_1 = \text{let } x = a_1 \) by \( y \) in \( b \)
  - Then \( t_2 = \text{let } x = a_2 \) by \( y \) in \( b \)
  - From lemma 4.2.8.5, \( t_1' = \text{let } x = a_1'' \) by \( y \) in \( b_1'' \) for some \( a_1'' \) and \( b_1'' \)
  - Case split on the form of \( \Gamma \triangleright t_1' \rightarrow \)
    - Case \( \Gamma \triangleright a_1'' \rightarrow a_1' \)
      - From I.H. \( \exists a_2'' =_u a_2 \quad \Gamma \triangleright a_2'' \rightarrow \)
      - So \( t_2 =_u \text{let } x = a_2'' \) by \( y \) in \( b \)
      - Let \( t_2' = \text{let } x = a_2'' \) by \( y \) in \( b \)
      - So \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
    - Case \( a_1'' \in I \)
      - From lemma 4.2.8.2, \( \exists a_2'' \in I \quad a_2'' =_u a_2 \)
      - So \( t_2 =_u \text{let } x = a_2'' \) by \( y \) in \( b \)
      - Let \( t_2' = \text{let } x = a_2'' \) by \( y \) in \( b \)
      - So \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
- Case \( t_1 = \text{let } x = v \) by \( y \) in \( a_1 \)
  - Then \( t_2 = \text{let } x = v \) by \( y \) in \( a_2 \)
  - From lemma 4.2.8.5, \( t_1' = \text{let } x = v_1'' \) by \( y \) in \( a_1'' \) for some \( v_1'' \) and \( a_1'' \)
  - Case split on the form of \( \Gamma \triangleright t_1' \rightarrow \)
    - Case \( \Gamma \triangleright v_1'' \rightarrow v_1' \)
      - So \( t_2 =_u \text{let } x = v_1'' \) by \( y \) in \( a_2 \)
      - Let \( t_2' = \text{let } x = v_1'' \) by \( y \) in \( a_2 \)
      - So \( \exists t_2' =_u t_2 \quad \Gamma \triangleright t_2' \rightarrow \)
Case \( v_l'' \in I \)
- So \( t_2 =_U \) let \( x = v_l'' \) by \( y \) in \( a_2 \)
- Let \( t_2' = \) let \( x = v_l'' \) by \( y \) in \( a_2 \)
- So \( \exists t_2' =_U t_2 \) \( \Gamma \triangleright t_2' \rightarrow \)

Case \( t_1 = \) match \( a_1 \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1 \ldots l c_n\overline{x}_n = \) bnl...end
- Then \( t_2 = \) match \( a_2 \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1 \ldots l c_n\overline{x}_n = \) bnl...end
- From lemma 4.2.8.5, \( t_1' = \) match \( a_1''' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1''' \ldots l c_n\overline{x}_n = \) bn''...end for some \( a_1''' \) and \( b_1''' \ldots b_n''' \)

Case split on the form of \( \Gamma \triangleright t_1' \rightarrow \)
- Case \( \Gamma \triangleright a_1'' \rightarrow a_1' \)
  - From I.H. \( \exists a_2'' =_U a_2 \) \( \Gamma \triangleright a_2'' \rightarrow \)
  - So \( t_2 =_U \) match \( a_2'' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1'' \ldots l c_n\overline{x}_n = \) bn''...end
  - Let \( t_2' = \) match \( a_2'' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1'' \ldots l c_n\overline{x}_n = \) bn''...end
  - So \( \exists t_2' =_U t_2 \) \( \Gamma \triangleright t_2' \rightarrow \)

Case \( a_1''' = c_i\overline{x} \)
- From lemma 4.2.8.2.1, \( \exists a_2'' =_U a_2 \) where \( a_2'' = c_i\overline{x} \)
  - So \( t_2 =_U \) match \( a_2'' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1'' \ldots l c_n\overline{x}_n = \) bn''...end
  - Let \( t_2' = \) match \( a_2'' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = \) \( b_1'' \ldots l c_n\overline{x}_n = \) bn''...end
  - So \( \exists t_2' =_U t_2 \) \( \Gamma \triangleright t_2' \rightarrow \)

Case \( t_1 = \) match \( s \) by \( x \ y \) return ! with \( c_i\overline{x}_i = b_1 l \ldots l c_n\overline{x}_n = \) bnl...end
- Then \( t_2 = \) match \( s \) by \( x \ y \) return ! with \( c_i\overline{x}_i = b_1 l \ldots l c_n\overline{x}_n = \) bnl...end
- From lemma 4.2.8.5, \( t_1' = \) match \( s_1''' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = b_1''' \ldots l c_n\overline{x}_n = \) bn'''...end for some \( s_1''' \), \( a_1''' \) and \( b_1''' \ldots b_n''' \)

Case split on the form of \( \Gamma \triangleright t_1' \rightarrow \)
- Case \( \Gamma \triangleright s_1'' \rightarrow s_1' \)
  - So \( t_2 =_U \) match \( s_1''' \) by \( x \ y \) return ! with \( c_i\overline{x}_i = b_1 l \ldots l c_i\overline{x}_i = b_2 l \ldots l c_n\overline{x}_n = \) bnl...end
Let \( t_2' \) = match \( s_1'' \) by \( x \, y \) return ! with
\[
\begin{align*}
& \Rightarrow b_1 \ldots \Rightarrow c_i \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow \text{bnl}\ldots \text{end} \\
& \text{end}
\end{align*}
\]
So \( \exists t_2' =_u t_2 \quad \Gamma \vdash t_2' \to \)
  
  \text{Case } s_1'' = \overline{c_i} \overline{x}
  
  So \( \exists t_2 =_u \text{match } s_1'' \text{ by } x \, y \text{ return }! \text{ with} \)
\[
\begin{align*}
& \Rightarrow b_1 \ldots \Rightarrow c_i \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow \text{bnl}\ldots \text{end} \\
& \text{end}
\end{align*}
\]
Let \( t_2' = \text{match } s_1'' \text{ by } x \, y \text{ return }! \text{ with} \)
\[
\begin{align*}
& \Rightarrow b_1 \ldots \Rightarrow c_i \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow a_2 \ldots \Rightarrow b_n \Rightarrow \text{bnl}\ldots \text{end} \\
& \Rightarrow \text{bnl}\ldots \text{end} \\
& \text{end}
\end{align*}
\]
So \( \exists t_2' =_u t_2 \quad \Gamma \vdash t_2' \to \)

**Lemma 4.2.8.7**

\[
\forall t_1, \forall t_2, \forall t_3, \forall U, \forall \Gamma \quad \Gamma \vdash t_1 \downarrow_U t_2 \quad \Gamma \vdash t_2 \downarrow_U t_3 \quad U \not\rightarrow
\]

\[
\frac{\Gamma \vdash t_1 \downarrow_U t_3}{\Gamma \vdash t_1 \downarrow_U t_3}
\]

Choose arbitrary \( t_1, t_2, t_3, U, \) and \( \Gamma \).
Assume: \( \Gamma \vdash t_1 \downarrow_U t_2 \quad \Gamma \vdash t_2 \downarrow_U t_3 \)

Derive: \( \Gamma \vdash t_1 \downarrow_U t_3 \)

Because \( \Gamma \vdash t_1 \downarrow_U t_2 \) and \( U \not\rightarrow \), \( \forall t_1', \forall t_2' \quad \Gamma \vdash t_1 \rightarrow_U t_1' \quad \Gamma \vdash t_2 \rightarrow_U t_2' \quad t_1' =_U t_2' \)

Because \( \Gamma \vdash t_2 \downarrow_U t_3 \) and \( U \not\rightarrow \), \( \forall t_2'', \forall t_3'' \quad \Gamma \vdash t_2 \rightarrow_U t_2'' \quad \Gamma \vdash t_3 \rightarrow_U t_3'' \quad t_2'' =_U t_3'' \)

Choose arbitrary \( t_1', t_2', t_2'', \) and \( t_3'' \) (termination is assumed)
From lemma 4.2.3, \( t_2' =_U t_2'' \)
From transitivity, \( t_1' =_U t_3'' \)
So \( \Gamma \vdash t_1 \downarrow_U t_3 \)
5 Conclusion and Future Work

This paper presented an algorithm that can be used to deduce the equality of terms in many common cases in OpTT. This algorithm will significantly reduce the amount of effort required to develop most formal proofs in OpTT. However, there are a few areas in which additional research would improve the algorithm even further. These areas are discussed in this section.

5.1 Termination

The algorithm in this paper is conjectured to terminate if all terms encountered are terminating with respect to the operational semantics of OpTT. A proof of this conjecture and an investigation into the conditions under which the hypjoin algorithm terminates would be incredibly useful. This information would allow the programmer to fully understand and predict when the algorithm will terminate.

5.2 Clash/Contra

It would be fairly trivial to extend hypjoin so that it is also complete with respect to the clash and contra proof rules which are used to derive conclusions in the presence of contradictions. The algorithm presented in this paper will terminate and fail if inconsistency is detected. It would be just as easy to successfully equate any two terms if the provided equations result in some contradiction.

5.3 Injectivity

Another simple extension involves completeness with respect to the inj proof rule which is used for reasoning about injectivity. The algorithm could simply detect equations that lead to some conclusion that can be derived via injectivity, then that conclusion could be added to the list of equations. For example, if the user-provided equations contain the equation
\{(C\ a) = (C\ b)\} where C is a term constructor, then the algorithm would add \{a = b\} to the list of user equations.
References


**Vita**

**Adam Petcher**

**Date of Birth**  
June 2, 1980

**Place of Birth**  
Dallas, Texas

**Degrees**  
B.S. *Cum Laude*, Computer Science, December 2002  
Texas Christian University, Fort Worth, Texas

M.S. Computer Science, May 2008  
Washington University in Saint Louis, Saint Louis, Missouri

May 2008