Entailment with Conditional Equality Constraints (Extended Version)

Zhendong Su *Alexander Aiken*

Report No. UCB/CSD-00-1113

October 2000

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Entailment with Conditional Equality Constraints (Extended Version)[⋆]

Zhendong Su and Alexander Aiken

EECS Department, University of California, Berkeley
{zhendong,aiken}@cs.berkeley.edu

Abstract Equality constraints (unification constraints) have widespread use in program analysis, most notably in static polymorphic type systems. *Conditional equality constraints* extend equality constraints with a weak form of subtyping to allow for more accurate analyses. We give a complete complexity characterization of the various entailment problems for conditional equality constraints and for a natural extension of conditional equality constraints.

1 Introduction

There are two decision problems associated with constraints: *satisfiability* and *entailment*. For the commonly used constraint languages in type inference and program analysis applications, the satisfiability problem is now well understood [1, 3, 10, 13, 18, 19, 26, 29, 31, 9, 8, 34]. For example, it is well-known that satisfiability of equality constraints can be decided in almost linear time (linear time if no infinite terms are allowed [27]). For entailment problems much less is known, and the few existing results give intractable lower bounds for the constraint languages they study, except for equality constraints where polynomial time algorithms exist [4, 5].

In this paper, we consider the entailment problem for *conditional equality constraints*. Conditional equality constraints extend the usual equality constraints with an additional form $\alpha \Rightarrow \tau$, which holds if $\alpha = \bot$ or $\alpha = \tau$. Conditional equality constraints have been used in a number of analyses, such as the tagging analysis of Henglein [16], and the pointer analysis proposed by Steensgaard [33], and a form of equality-based flow systems for higher order functional languages [25]. Besides conditional equality constraints, we also consider entailments for a natural extension of conditional constraints.

Consider the equality constraints $C_1 = \{\alpha = \beta, \beta = \gamma, \alpha = \gamma\}$. Since $\alpha = \gamma$ is implied by the other two constraints, we can simplify the constraints to $C_2 = \{\alpha = \beta, \beta = \gamma\}$. We say that "$C_1$ entails C_2", written $C_1 \vDash C_2$, which means that every solution of C_1 is also a solution of C_2. In this case we also have $C_2 \vDash C_1$, since the two systems have exactly the same solutions. In the program analysis community, the primary motivation for studying entailment problems comes from type systems with *polymorphic constrained types*. Such type systems combine polymorphism (as in ML [20]) with subtyping (as in object-oriented languages such as Java [12]), giving polymorphic types with associated subtyping constraints. A difficulty with constrained types is that there are many equivalent representations of the same type, and the "natural" ones to compute tend to be very large and unwieldy. For the type system to be practical, scalable, and understandable to the user, it is important to simplify the constraints associated with a type. As the example above illustrates, entailment of constraint systems is a decision problem closely related to constraint simplification.

[⋆] This research was supported in part by the National Science Foundation grant No. CCR-0085949 and NASA Contract No. NAG2-1210.

	Complexity	
Constraints	Simple Entailment	Restricted Entailment
Conditional Equality	P-complete	P-complete
Extended Conditional Equality	P-complete	coNP-complete

Table 1. Summary of results.

Considerable effort has been directed at constraint simplification. One body of work considers practical issues with regard to simplification of constraints [7, 9, 8, 34, 19], suggesting heuristics for simplification and experimentally measuring the performance gain of simplifications. Another body of work aims at a better understanding how difficult the simplification problems are for various constraint logics [9, 14, 15]. Flanagan and Felleisen [9] consider the simplification problem for a particular form of set constraints and show that a form of entailment is PSPACE-hard. Henglein and Rehof [14, 15] consider another simpler form of entailment problem for subtyping constraints. They show that structural subtyping entailment for constraints over simple types is coNP-complete and that for recursive types is PSPACE-complete, and that the nonstructual entailment for both simple types and recursive types is PSPACE-hard. A complete complexity characterization of nonstructual subtyping entailment remains open. In fact, it is an open problem whether nonstructual subtyping entailment is decidable. Thus for these different forms of constraints, the problems are intractable or may even be undecidable. In the constraint logic programming community, the entailment problems over equality constraints have been considered by Colmerauer and shown to be polynomial time decidable [4, 5, 17, 32]. Previous work leaves open the question of whether there are other constraint languages with efficiently decidable entailment problems besides equality constraints over trees (finite or infinite).

1.1 Contributions

We consider two forms of the entailment problem: *simple entailment* and *restricted entailment* (sometimes also referred to as *existential entailment*), which we introduce in Section 2. Restricted entailment arises naturally in problems that compare polymorphic constrained types (see Section 2). Table 1 contains a summary of the main results of this paper. In particular, we show there are polynomial time algorithms for equality and conditional equality constraints for both versions of entailment. We believe these algorithms will be of practical interest. In addition, we consider simple entailment and restricted entailment for a simple and natural extension of conditional equality constraints. We show that although simple entailment for the extension admits polynomial time algorithms, restricted entailment for this extension turns out to be coNP-complete. The coNP-completeness result is interesting because it provides a natural boundary between tractable and intractable constraint languages.

2 Preliminaries

We work with simple types. The algorithms we present apply to type languages with other base types and type constructors. Our type language is

$$\tau ::= \bot \mid \top \mid \tau_1 \to \tau_2 \mid \alpha.$$

This simple language has two constants $\bot$ and $\top$, a binary constructor $\rightarrow$, and variables α ranging over a denumerable set $\mathcal{V}$ of type variables. Variable-free types are *ground types*. $\mathcal{T}$ and $\mathcal{T}_{\mathcal{G}}$ are the set of types and the set of ground types respectively. An *equality constraint* is $\tau_1 = \tau_2$ and a *conditional equality constraint* is $\alpha \Rightarrow \tau$. A *constraint system* is a finite conjunction of equality and conditional equality constraints. An *equality constraint system* has only equality constraints.

Let C be a constraint system and $\mathrm{Var}(C)$ the set of type variables appearing in C. A *valuation* of C is a function mapping $\mathrm{Var}(C)$ to ground types $\mathcal{T}_{\mathcal{G}}$. We extend a valuation ρ to work on type expressions in the usual way:

- $\rho(\bot) = \bot$;
- $\rho(\top) = \top$;
- $\rho(\tau_1 \rightarrow \tau_2) = \rho(\tau_1) \rightarrow \rho(\tau_2)$.

A valuation ρ *satisfies* constraint $\tau_1 = \tau_2$, written $\rho \vDash \tau_1 = \tau_2$, if $\rho(\tau_1) = \rho(\tau_2)$, and it satisfies a constraint $\alpha \Rightarrow \tau$, written $\rho \vDash \alpha \Rightarrow \tau$, if $\rho(\alpha) = \bot$ or $\rho(\alpha) = \rho(\tau)$. We write $\rho \vDash C$ if ρ satisfies every constraint in C.

Definition 1 (Terms). Let C be a set of constraints. $\mathrm{Term}(C)$ is the set of terms appearing in C: $\mathrm{Term}(C) = \{\tau_1, \tau_2 \mid (\tau_1 = \tau_2) \in C \vee (\tau_1 \Rightarrow \tau_2) \in C\}$.

The satisfiability of equality constraints can be decided in almost linear time in the size of the original constraints using a union-find data structure [35]. With a simple modification to this algorithm for equality constraints, we can decide the satisfiability of a system of conditional equality constraints in almost linear time (see Proposition 1 below). [1]

Example 1. Here are example conditional constraints:

- $\alpha \Rightarrow \bot$;
  Solution: α must be $\bot$
- $\alpha \Rightarrow \top$;
  Solution: α is either $\bot$ or $\top$.
- $\alpha \Rightarrow \beta \rightarrow \gamma$
  Solution: α is either $\bot$ or a function type $\beta \rightarrow \gamma$, where β and γ can be any type.

Proposition 1. Let C be any system of constraints with equality constraints and conditional equality constraints. We can decide whether there is a satisfying valuation for C in almost linear time.

Proof. [Sketch] The basic idea of the algorithm is to solve the equality constraints and to maintain along with each variable a list of constraints conditionally depending on that variable. Once a variable α is unified with a non-$\bot$ value, any constraints $\alpha \Rightarrow \tau$ on the list are no longer conditional and are added as equality constraints $\alpha = \tau$. Note that a post-processing step is required to perform the occurs check. The time complexity is still almost linear since each constraint is processed at most twice. See, for example, [33] for more information. □

In later discussions, we refer to this algorithm as CondResolve. The result of running the algorithm on C is a term dag denoted by $\textsc{CondResolve}(C)$ (see Definition 5) similar to unification based on the union-find data structure. As is standard, for any term τ, we denote the equivalence class to which τ belongs by $\textsc{ecr}(\tau)$.

[1] Notice that using a linear unification algorithm such as [27] does not give a more efficient algorithm, because equality constraints are added dynamically.

In this paper, we consider two forms of entailment: *simple entailment*: $C \vDash c$, and *restricted entailment*: $C_1 \vDash_E C_2$, where C, C_1, and C_2 are systems of constraints, and c is a single constraint, and E is a set of *interface* variables. In the literature, $C_1 \vDash_E C_2$ is sometimes written $C_1 \vDash \exists E'.C_2$, where $E' = \mathrm{Var}(C_2) \setminus E$.

For the use of restricted entailment, consider the following situation. In a polymorphic analysis, a function (or a module) is analyzed to generate a system of constraints [11, 9]. Only a few of the variables, the *interface variables*, are visible outside the function. We would like to simplify the constraints with respect to a set of interface variables. In practice, restricted entailment is more commonly encountered than simple entailment.

Definition 2 (Simple Entailment). Let C be a system of constraints and c a constraint. We say that $C \vDash c$ if for every valuation ρ with $\rho \vDash C$, we have $\rho \vDash c$ also.

Definition 3 (Restricted Entailment). Let C_1 and C_2 be two constraint systems, and let E be the set of variables $\mathrm{Var}(C_1) \cap \mathrm{Var}(C_2)$. We say that $C_1 \vDash_E C_2$ if for every valuation ρ_1 with $\rho_1 \vDash C_1$ there exists ρ_2 with $\rho_2 \vDash C_2$ and $\rho_1(\alpha) = \rho_2(\alpha)$ for all $\alpha \in E$.

Definition 4 (Interface and Internal Variables). In $C_1 \vDash_E C_2$, variables in E are *interface variables*. Variables in $(\mathrm{Var}(C_1) \cup \mathrm{Var}(C_2)) \setminus E$, are *internal variables*.

Notations

- τ and τ_i denote type expressions.
- α, β, γ, α_i, β_i, and γ_i denote interface variables.
- μ, ν, σ, μ_i, ν_i, and σ_i denote internal variables.
- α denotes a generic variable, in places where we do not distinguish interface and internal variables.

For simple entailment $C \vDash c$, it suffices to consider only the case where c is a constraint between variables, *i.e.*, c is of the form $\alpha = \beta$ or $\alpha \Rightarrow \beta$. For simple entailment, $C \vDash \tau_1 = \tau_2$ if and only if $C \cup \{\alpha = \tau_1, \beta = \tau_2\} \vDash \alpha = \beta$, where α and β do not appear in C and c. The same also holds for when c is of the form $\alpha \Rightarrow \tau$.

Simple entailment also enjoys a distributive property, that is $C_1 \vDash C_2$ if and only if $C_1 \vDash c$ for each $c \in C_2$. Thus it suffices to only study $C \vDash c$. This distributive property does not hold for restricted entailment. Consider $\emptyset \vDash_{\{\alpha,\beta\}} \{\alpha \Rightarrow \sigma, \beta \Rightarrow \sigma\}$, where σ is a variable different from α and β. This entailment does not hold (consider $\rho_1(\alpha) = \top$ and $\rho_1(\beta) = \bot \to \bot$), but the entailments $\emptyset \vDash_{\{\alpha,\beta\}} \{\alpha \Rightarrow \sigma\}$ and $\emptyset \vDash_{\{\alpha,\beta\}} \{\beta \Rightarrow \sigma\}$ both hold.

Terms can be represented as directed trees with nodes labeled with constructors and variables. Term graphs (or term DAGs) are a more compact representation to allow sharing of common subterms.

Definition 5 (Term DAG). In a term DAG, a variable is represented as a node with out-degree 0. A function type is represented as a node $\to$ with out-degree 2, one for the domain and one for the range. No two different nodes in a term DAG may represent the same term (sharing must be maximal).

We also represent conditional constraints in the term graph. We represent $\alpha \Rightarrow \tau$ as a directed edge from the node representing α to the node representing τ. We call such an edge a *conditional edge*, in contrast to the two outgoing edges from a $\to$ node, which are called *structural edges*.

It is well-known that unification is P-complete [6]. Since unification can be trivially reduced in log-space to the various entailments we consider, it suffices to give polynomial time algorithms to show P-completeness of these problems.

$$\{\tau_1 = \tau_2, \tau_2 = \tau_3\} \subseteq S \Rightarrow \{\tau_1 = \tau_3\} \subseteq S$$
$$\{\tau_1 \to \tau_2 = \tau_1' \to \tau_2'\} \subseteq S \Rightarrow \{\tau_1 = \tau_1', \tau_2 = \tau_2'\} \subseteq S$$
$$\{\bot = \top\} \subseteq S \Rightarrow \text{not satisfiable}$$
$$\{\bot = \tau_1 \to \tau_2\} \subseteq S \Rightarrow \text{not satisfiable}$$
$$\{\top = \tau_1 \to \tau_2\} \subseteq S \Rightarrow \text{not satisfiable}$$

Figure 1. Closure rules for equality constraints.

$$\{\tau_1 = \tau_2, \tau_2 = \tau_3\} \subseteq S \Rightarrow \{\tau_1 = \tau_3\} \subseteq S$$
$$\{\tau_1 = \tau_1', \tau_2 = \tau_2'\} \subseteq S \Rightarrow \{\tau_1 \to \tau_2 = \tau_1' \to \tau_2'\} \subseteq S$$

Figure 2. Congruence closure for equality constraints.

3 Entailment of Equality Constraints

In this section we consider the entailment problems for equality constraints. These results are useful in later sections. Algorithms for entailment over equality constraints are known [17, 32, 4, 5]. We include them for completeness since these results are used heavily in later sections for studying entailment over conditional equality constraints.

3.1 Simple Entailment

We first consider the simple entailment problem for equality constraints. For $C \models c$ to hold, the constraint c cannot put extra restrictions (beyond those in C) on the variables in C. We use this idea to get an efficient algorithm for deciding whether $C \models c$ for equality constraints.

Recall that the basic algorithm for checking the satisfiability of an equality constraint system is to put the constraint system into some closed form according to the rules in Figure 1 (Robinson's algorithm) [30].

An efficient implementation of Robinson's algorithm is based on the union-find data structure. The algorithm operates on a graph representation of the constraints and closes the graphs according to structural decomposition. A union-find data structure maintains equivalence classes, with a designated *representative* for each equivalence class. For any term τ, we denote the equivalence class to which τ belongs by $\text{ECR}(\tau)$. If the algorithm succeeds, the resulting DAG forest represents the *most general unifier* (m.g.u.) of the original constraint system. The algorithm fails if it discovers a constructor mismatch (the last three rules in Figure 1).

The standard unification algorithm is not sufficient for deciding entailment because structural equivalence is not explicit in the resulting DAG representation. That is, constraints of the form $\tau_1 \to \tau_2 = \tau_1' \to \tau_2'$ are decomposed into $\tau_1 = \tau_1'$ and $\tau_2 = \tau_2'$, but the equivalence of $\tau_1 \to \tau_2$ and $\tau_1' \to \tau_2'$ is not represented explicitly. Moreover, besides constraint decomposition, there are situations in which $\tau_1 = \tau_1'$ and $\tau_2 = \tau_2'$, but the equivalence $\tau_1 \to \tau_2 = \tau_1' \to \tau_2'$ is not explicitly represented. For entailment, we would like equivalence classes to mean both that all members of a class X are equal in all solutions, but also that every other equivalence class Y is *different* from X in at least one solution. Thus, equivalence classes should be as large as possible (maximal). This property is guaranteed by congruence closure.

1. Compute m.g.u. of C. If fail, output YES; else continue.
2. Compute the congruence closure on the m.g.u. of C. If $\alpha = \beta$ is in the closure, output YES; else output NO.

Figure 3. Simple entailment $C \vDash \alpha = \beta$ over equality constraints.

For $C \vDash \alpha = \beta$ to hold, it suffices to check whether $\alpha = \beta$ is implied by C with respect to the congruence closure rules in Figure 2, *i.e.*, whether $\alpha = \beta$ is a constraint in the congruence closure of C. Congruence closure can be computed in $\mathcal{O}(n \log n)$ [22]. Figure 3 gives an algorithm for simple entailment over equality constraints [2].

Lemma 1. Let C be a constraint system. C and $Cong(C)$, the congruence closure of C, have the same solutions.

Proof. The rules in Figure 2 preserve solutions. □

The algorithm clearly runs in $\mathcal{O}(n \log n)$, where $n = |C|$.

Theorem 1 (Correctness). The algorithm in Figure 3 is correct.

Proof. If the algorithm outputs YES, the constraint $\alpha = \beta$ is contained in the congruence closure of the m.g.u. of C. By Lemma 1, we know that C and the congruence closure of C's m.g.u. have the same solutions. Thus $C \vDash \alpha = \beta$.

If the algorithm outputs NO, then the constraint $\alpha = \beta$ is not in the congruence closure of the m.g.u. of C. One can show with an induction on the structure of the *equivalence class representatives* of α and β, *i.e.*, ECR(α) and ECR(β), that any two non-congruent classes admit a valuation $\rho \vDash C$ which maps the two non-congruent classes to different ground types. Thus, $C \nvDash \alpha = \beta$. □

3.2 Restricted Entailment

In this subsection, we consider restricted entailment for equality constraints. This relation is more involved, since for polymorphic analyses, there are some interface variables we are interested in, and the other internal variables may be eliminated, which can result in a smaller constraint system.

The goal is to decide $C_1 \vDash_E C_2$ for two constraint systems C_1 and C_2 and $E = \mathrm{Var}(C_1) \cup \mathrm{Var}(C_2)$. Recall that $C_1 \vDash_E C_2$ if and only if for every valuation $\rho_1 \vDash C_1$ there exists a valuation $\rho_2 \vDash C_2$ such that $\rho_1(\alpha) = \rho_2(\alpha)$ for all $\alpha \in E$.

We first consider an example to illustrate the issues. Consider the two constraint systems $C_1 = \{\alpha = \sigma \to \sigma\}$ and $C_2 = \{\alpha = \sigma_1 \to \sigma_2\}$ with the interface variables $E = \{\alpha\}$. For any valuation $\rho \vDash C_1$, we know $\rho(\alpha) = \tau \to \tau$ for some τ. Consider the valuation ρ' of C_2 with $\rho'(\alpha) = \rho(\alpha)$ and $\rho'(\sigma_1) = \rho'(\sigma_2) = \rho(\sigma)$. It is easy to see that $\rho' \vDash C_2$, and thus the relation $C_1 \vDash_E C_2$ holds. The algorithm for simple entailment given in Figure 3 does not apply since for this example, it would give the answer NO. Note that if $C_1 \vDash C_2$ (holds iff for all $c \in C_2$ we have $C_1 \vDash c$) then $C_1 \vDash_E C_2$ for any E. The converse, however, is not true, as shown by the example.

We modify the simple entailment algorithm over equality constraints to get an algorithm for restricted entailment over equality constraints. The intuition behind the algorithm is that we can

[2] The congruence closure computation is unnecessary. We could simply add the constraint $\alpha = \beta$ to the m.g.u. of C_1 and continue with unification to check if any two distinct equivalence classes are merged. This results in an almost linear time algorithm for a single query. Congruence closure gives a simpler explanation, and also gives an algorithm that answers queries in constant time.

1. Compute m.g.u. of C_1. If fail, output YES; else continue.
2. Add each term of C_2 to the m.g.u. of C_1 if the term is not already present.
3. Compute the congruence closure on the term graph obtained in Step 2.
4. Unify the constraints of C_2 in the result obtained in Step 3, and perform the following check. For any two non-congruent classes that are unified, we require at least one of the representatives to be a variable in $\mathrm{Var}(C_2) \setminus E$. If this requirement is not met, output NO; else output YES.

Figure 4. Restricted entailment $C_1 \vDash_E C_2$ over equality constraints.

$$\begin{cases} \rho(\alpha) = \rho_1(\alpha) & \text{if } \alpha \in \mathrm{Var}(C_1) \\ \rho(\alpha) = \left\{ \begin{array}{ll} \rho_1(\textsc{ecr}(\alpha)) & \text{if } \textsc{ecr}(\alpha) \in \mathrm{Var}(C_1) \\ \bot & \text{if } \textsc{ecr}(\alpha) \in \mathrm{Var}(C_2) \setminus E \\ \bot & \text{if } \textsc{ecr}(\alpha) = \bot \\ \top & \text{if } \textsc{ecr}(\alpha) = \top \\ \rho(\tau_1) \to \rho(\tau_2) & \text{if } \textsc{ecr}(\alpha) = \tau_1 \to \tau_2 \end{array} \right\} & \text{if } \alpha \in \mathrm{Var}(C_2) \setminus E \end{cases}$$

Figure 5. Constructed valuation ρ.

relax the requirement of simple entailment to allow internal variables of C_2 to be added to equivalence classes of the term DAG representation of the m.g.u. of C_1, as long as no equivalence classes of C_1 are merged. The algorithm is given in Figure 4 [3].

In Figure 4, the choice of representatives for equivalence classes is important. We pick representatives in the following order, which guarantees that if the representative is in $\mathrm{Var}(C_2) \setminus E$, then there is no variable in $\mathrm{Var}(C_1)$ or a constructor in the equivalence class:

1. $\bot$, $\top$, and $\to$ nodes;
2. variables in $\mathrm{Var}(C_1)$;
3. variables in $\mathrm{Var}(C_2) \setminus E$.

Let $n = |C_1|$ and $m = |C_2|$. It is easy to see that the algorithm takes time $\mathcal{O}((m+n)\log(m+n))$.

Theorem 2 (Correctness). The algorithm in Figure 4 is correct.

Proof. Suppose the algorithm outputs YES. If C_1 is not satisfiable then clearly $C_1 \vDash_E C_2$. Let ρ_1 be a satisfying valuation of C_1. Consider the valuation ρ given in Figure 5.

The valuation ρ is clearly well-defined. Let ρ_2 denote the valuation obtained by restricting ρ to $\mathrm{Var}(C_2)$, *i.e.*, the variables in C_2. We want to show that $\rho_2 \vDash C_2$. Since the algorithm outputs YES, when adding the constraints in C_2, the only change to the graph is adding variables in $\mathrm{Var}(C_2) \setminus E$ to some existing equivalence classes. By the construction of ρ, one can see that ρ satisfies all the induced constraints in the term graph at step 4 of the algorithm. Thus, we have $\rho \vDash C_1 \cup C_2$, and therefore $\rho_2 \vDash C_2$. Hence, we have $C_1 \vDash_E C_2$.

Conversely, suppose the algorithm outputs NO. Then there exist two equivalence classes to be unified neither of whose ECR is a variable in $\mathrm{Var}(C_2) \setminus E$. There are two cases.

- In the first case, one ECR is a variable in $\mathrm{Var}(C_1)$, say α. If the other representative is $\bot$, then any valuation $\rho_1 \vDash C_1$ with $\rho_1(\alpha) = \top$ gives a witness for $C_1 \nvDash_E C_2$. The case where the other representative is $\top$ or a $\to$ node is similar. If the other representative is a variable $\beta \in \mathrm{Var}(C_1)$, any valuation $\rho_1 \vDash C_1$ with $\rho_1(\alpha) = \top$ and $\rho_1(\beta) = \bot$ is a witness for $C_1 \nvDash_E C_2$.

[3] The step of congruence closure is again not necessary here.

Let C be a system of constraints. The following algorithm outputs a term graph representing the solutions of C.

1. Let G be the term graph CondResolve(C).
2. For each variable α in $\mathrm{Var}(C)$, check whether it must be $\bot$: If neither $G \cup \{\alpha = \top\}$ nor $G \cup \{\alpha = \sigma_1 \to \sigma_2\}$ is satisfiable, add $\alpha = \bot$ to G.

Figure 6. Modified conditional unification algorithm.

– In the second case, both ECRs are constructors. If there is a constructor mismatch, then $C_1 \cup C_2$ is not satisfiable. Since C_1 is satisfiable, then $C_1 \not\models_E C_2$ (Since if $C_1 \models_E C_2$, any satisfying valuation for C_1 can be extended to a satisfying valuation for C_2.)
Note that if there is no constructor mismatch (where both representatives are $\to$ nodes), the error is detected when trying to unify the terms represented by these two nodes. Thus it falls into one of the above cases.

Thus it follows that $C_1 \not\models_E C_2$.

□

4 Conditional Equality Constraints

In this section, we consider the two entailment problems for constraint systems with conditional equality constraints. Recall for $\alpha \Rightarrow \tau$ to be satisfied by a valuation ρ, either $\rho(\alpha) = \bot$ or $\rho(\alpha) = \rho(\tau)$.

Lemma 2 (Transitivity of $\Rightarrow$). Any valuation ρ satisfying $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \gamma$, also satisfies $\alpha \Rightarrow \gamma$.

Consider the constraints $\{\alpha \Rightarrow \top, \alpha \Rightarrow \bot \to \bot\}$. The only solution is $\alpha = \bot$. The fact that α must be $\bot$ is not explicit. For entailment, we want to make the fact that α must be $\bot$ explicit.

Assume that we have run CondResolve on the constraints to get a term graph G. For each variable α, we check whether it must be $\bot$. If both adding $\alpha = \top$ to G and $\alpha = \sigma_1 \to \sigma_2$ to G (for fresh variables σ_1 and σ_2) fail, α must be $\bot$, in which case, we add $\alpha = \bot$ to G. We repeat this process for each variable. Notice that this step can be done in polynomial time. We present this modification to the conditional unification algorithm in Figure 6.

4.1 Simple Entailment

In this subsection, we present an algorithm for deciding $C \models \alpha = \beta$ and $C \models \alpha \Rightarrow \beta$ where C_1 and C_2 are constraint systems with conditional equality constraints. Note $C \models \alpha = \beta$ holds if and only if $C \models \alpha \Rightarrow \beta$ and $C \models \beta \Rightarrow \alpha$ both hold. We give the algorithm in Figure 7. The basic idea is that to check $C \models \alpha \Rightarrow \beta$ holds we have two cases: when α is $\top$ and when α is a function type. In both cases, we require $\beta = \alpha$. The problem then basically reduces to simple entailment over equality constraints. As for entailment for equality constraints, congruence closure is required to make explicit the implied equalities between terms involving $\to$. Computing strong components is used to make explicit, for example, $\alpha = \beta$ if both $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \alpha$. It is easy to see that the algorithm runs in worst case polynomial time in the size of C.

If both of the following cases return SUCCESS, output YES; else output NO.

1. (a) Run the conditional unification algorithm in Figure 6 on $C \cup \{\alpha = \top\}$. If not satisfiable, then SUCCESS; else continue.
   (b) Compute strongly connected components (SCC) on the conditional edges and merge the nodes in every SCC. This step yields a modified term graph.
   (c) Compute congruence closure on the term graph obtained in Step 1b. We do not consider the conditional edges for computing congruence closure.
   (d) If $\beta = \top$ is in the closure, SUCCESS; else FAIL.
2. (a) Run the conditional unification algorithm in Figure 6 on $C \cup \{\alpha = \sigma_1 \to \sigma_2\}$, where σ_1 and σ_2 are two fresh variables not in $\mathrm{Var}(C) \cup \{\alpha, \beta\}$. If not satisfiable, then SUCCESS; else continue.
   (b) Compute strongly connected components (SCC) on the conditional edges and merge the nodes in every SCC. This step yields a modified term graph.
   (c) Compute congruence closure on the term graph obtained in Step 2b. Again, we do not consider the conditional edges for computing congruence closure.
   (d) If $\beta = \sigma_1 \to \sigma_2$ is in the closure, SUCCESS; else FAIL.

Figure 7. Simple entailment $C \vDash \alpha \Rightarrow \beta$ over conditional equality constraints.

Theorem 3. The simple entailment algorithm in Figure 7 is correct.

Proof. Suppose that the algorithm outputs YES. Let ρ be a satisfying valuation for C. We have three cases.

- If $\rho(\alpha) = \bot$, then $\rho \vDash \alpha \Rightarrow \beta$.
- If $\rho(\alpha) = \top$, then $\rho \vDash C \cup \{\alpha = \top\}$. Thus $C \cup \{\alpha = \top\}$ is satisfiable. We then have $\beta = \top$ is in the closure of $C \cup \{\alpha = \top\}$. Hence, $\rho \vDash \beta = \top$, which implies that $\rho \vDash \alpha \Rightarrow \beta$.
- If $\rho(\alpha) = \tau_1 \to \tau_2$ where τ_1 and τ_2 are ground types, we have $C \cup \{\alpha = \sigma_1 \to \sigma_2\}$ is satisfiable. We thus have $\beta = \sigma_1 \to \sigma_2$ is in the closure of $C \cup \{\alpha = \sigma_1 \to \sigma_2\}$. Hence $\rho \vDash \beta = \sigma_1 \to \sigma_2$, and which implies that $\rho \vDash \alpha \Rightarrow \beta$.

Combining the three cases, we conclude that $C \vDash \alpha \Rightarrow \beta$.

Suppose the algorithm outputs NO. Then at least one of the two cases returns FAIL. Assume that the case with $\alpha = \top$ returns FAIL. Then $\beta = \top$ is not in the congruence closure of $C \cup \{\alpha = \top\}$. By assigning all the remaining *conditional variables* to $\bot$ (variables appearing as the antecedent of the conditional constraints) in the graph, we can exhibit a witness valuation ρ that satisfies C but does not satisfy $\alpha \Rightarrow \beta$ (same as the case for simple entailment over equality constraints). The other case where $\alpha = \sigma_1 \to \sigma_2$ is similar. Hence $C \nvDash \alpha \Rightarrow \beta$.

□

4.2 Restricted Entailment over Atomic Constraints

Before considering the problem $C_1 \vDash_E C_2$ over conditional equality constraints, we look at a simpler case, in which all the constraints are between variables. With minor modifications, the presented algorithm can handle constraints over atoms (variables, $\bot$ and $\top$).

We first characterize the solutions of a constraint system with respect to a set of variables. Let C be a constraint system and $E \subseteq \mathrm{Var}(C)$. Recall that a conditional constraint $\alpha \Rightarrow \beta$ is represented as a directed edge from the node representing α to the node representing β. We compute the strongly connected components (SCC) of the term graph representation of the conditional constraints. We now perform the following transformations:

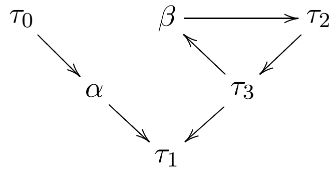
– If a variable $\alpha \notin E$ appears in a strong component with variables in E, then remove α and its incident edges from the component.
– Remove any conditional edge if the antecedent component consists only of variables in $\mathrm{Var}(C) \setminus E$.
– Remove any isolated component consisting only of variables in $\mathrm{Var}(C) \setminus E$.

Compute the transitive closure relation on the resulting dag. The resulting graph is the *normal form* $\textsc{nf}(C^E)$ of C with respect to E.

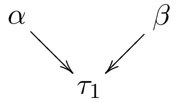
Example 2. Consider the constraints

$$\{\tau_0 \Rightarrow \alpha, \alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2, \tau_2 \Rightarrow \tau_3, \tau_3 \Rightarrow \beta, \tau_3 \Rightarrow \tau_1\}$$

The graph representation of the constraints is:



The graph $\textsc{nf}(C^{\{\alpha,\beta\}})$ is



The solutions w.r.t. $\{\alpha, \beta\}$ are the same, either α is $\bot$, or β is $\bot$, or $\alpha = \beta$.

For a constraint system C, we denote by $S(C) \mid_E$ the set of satisfying valuations restricted to E, i.e.

$$S(C) \mid_E = \{\rho' \mid \rho \vDash C \text{ and } \rho' = \rho \mid_E\},$$

where $\rho \mid_E$ denotes the valuation of ρ restricted to the variables E.

Lemma 3. $S(C) \mid_E = S(\textsc{nf}(C^E)) \mid_E$.

Proof. By transitivity of $\Rightarrow$ and the fact that we only add transitive constraints and remove other constraints from C to get $\textsc{nf}(C^E)$, it is clear that $S(C) \mid_E \subseteq S(\textsc{nf}(C^E)) \mid_E$.

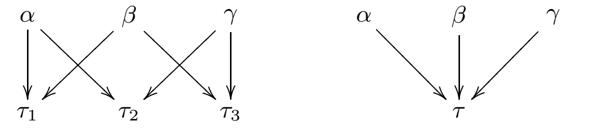
For the other direction, we need to show that each valuation in $S(\textsc{nf}(C^E)) \mid_E$ is also in $S(C) \mid_E$. Let ρ be a valuation in $S(\textsc{nf}(C^E)) \mid_E$. It is extensible to a valuation ρ' of $\textsc{nf}(C^E)$. We want to show that ρ' can be extended to a valuation ρ'' that satisfies C. We define ρ'' as follows

$$\begin{cases} \rho''(\alpha) = \rho'(\alpha) \text{ if } \alpha \in \mathrm{Var}(\textsc{nf}(C^E)) \\ \rho''(\alpha) = \rho'(\beta) \text{ if } \textsc{scc}(\alpha) = \textsc{scc}(\beta) \text{ and } \beta \in E \\ \rho''(\alpha) = \bot \qquad \text{otherwise} \end{cases}$$

where $\textsc{scc}(\alpha) = \textsc{scc}(\beta)$ means that α and β are in the same strong component.

One can verify that $\rho'' \vDash C$ by observing that the only constraints that are removed from C are of the forms $\tau \Rightarrow \alpha$, $\tau_1 = \tau_2$, or $\tau = \beta$ where τ, τ_1, and τ_2 are variables not in E and α and β are in E. These constraints are satisfied by our construction of ρ''.

□



(a) Term graph for C_1. (b) Term graph for C_2.

Figure 8. Example.

After this transformation only interface variables in E can appear as the left-side of conditional constraints.

We now consider the decision problem $C_1 \models_E C_2$. Before giving the algorithm, let us look at another example. Consider the constraints

$$C_1 = \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_1, \alpha \Rightarrow \tau_2, \gamma \Rightarrow \tau_2, \beta \Rightarrow \tau_3, \gamma \Rightarrow \tau_3\}$$

and

$$C_2 = \{\alpha \Rightarrow \tau, \beta \Rightarrow \tau, \gamma \Rightarrow \tau\}$$

We want to determine that $C_1 \equiv_E C_2$ where $E = \{\alpha, \beta, \gamma\}$. The term graph representations for C_1 and C_2 are given in Figure 8. These are also their normal forms. Notice that the constraints $\alpha \Rightarrow \tau_1$ and $\beta \Rightarrow \tau_1$ force $\alpha = \beta$ when $\alpha \neq \bot$ and $\beta \neq \bot$. Thus we can easily characterize C_1's solutions restricted to E as

- All of α, β, and γ are $\bot$;
- One of them is not $\bot$ and can be any (non-$\bot$) value;
- Two of them are not $\bot$ and have the same (non-$\bot$) value ;
- All three have the same (non-$\bot$) value.

Similarly, we get the same characterization for C_2, thus $C_1 \equiv_E C_2$.

We now give our algorithm for deciding $C_1 \models_E C_2$. The algorithm is in Figure 9. Case 2c handles the tricky case illustrated by Figure 8.

We first analyze the running time of the algorithm. The time to compute $\text{NF}(C_1^E)$ and $\text{NF}(C_2^E)$ is $\mathcal{O}(m^2 + n^2)$ where $m = |C_1|$ and $n = |C_2|$. The time to perform the checks can be done in time $\mathcal{O}(mn)$. Thus the running time of the algorithm is $\mathcal{O}(m^2 + n^2 + mn)$.

Theorem 4. The algorithm in Figure 9 is correct.

Proof. Let $pre^C(\alpha)$ be $\{\beta \mid \beta \Rightarrow \alpha\}$ in constraints C. Define $T_\rho^C(\alpha) = \bot$ if the valuation ρ maps all variables in $pre^C(\alpha)$ to $\bot$, otherwise, $T_\rho^C(\alpha) = \rho(\beta)$ for any $\beta \in pre^C(\alpha)$ with $\rho(\beta) \neq \bot$.

Assume the algorithm outputs YES. We show that for each valuation $\rho \models \text{NF}(C_1^E)$, the valuation $\rho\mid_E$ can be extended to a valuation ρ' that satisfies $\text{NF}(C_2^E)$. We define ρ' as follows

$$\begin{cases} \rho'(\alpha) = \rho(\alpha) & \text{if } \alpha \in E \\ \rho'(\alpha) = T_\rho^{\text{NF}(C_2^E)}(\alpha) & \text{if } \alpha \in \text{Var}(\text{NF}(C_2^E)) \setminus E \end{cases}$$

We first show that ρ' is well-defined. We argue that all variables in $pre^{\text{NF}(C_2^E)}(\alpha)$ must be mapped to the same value. Let α_1 and α_2 be in $pre^{\text{NF}(C_2^E)}(\alpha)$ and $\rho(\alpha_1) \neq \bot$ and $\rho(\alpha_2) \neq \bot$. Since $\alpha_1 \Rightarrow \alpha$ and $\alpha_2 \Rightarrow \alpha$ are constraints in $\text{NF}(C_2^E)$, we have either $(\alpha_1 = \alpha_2) \in \text{NF}(C_1^E)$ or $\{\alpha_1 \Rightarrow \alpha', \alpha_2 \Rightarrow$

```
1. Compute $\textsc{nf}(C_1^E)$ and $\textsc{nf}(C_2^E)$;
2. Perform the following checks (where $\alpha \in E$ and $\beta \in E$, and $\tau \notin E$ and $\tau' \notin E$)
   (a) if $(\alpha = \beta) \in \textsc{nf}(C_2^E)$, check $(\alpha = \beta) \in \textsc{nf}(C_1^E)$;
   (b) if $(\alpha \Rightarrow \beta) \in \textsc{nf}(C_2^E)$, check $(\alpha = \beta) \in \textsc{nf}(C_1^E)$ or $(\alpha \Rightarrow \beta) \in \textsc{nf}(C_1^E)$;
   (c) if $\{\alpha \Rightarrow \tau, \beta \Rightarrow \tau\} \subseteq \textsc{nf}(C_2^E)$, check $(\alpha = \beta) \in \textsc{nf}(C_1^E)$ or $\{\alpha \Rightarrow \tau', \beta \Rightarrow \tau'\} \subseteq \textsc{nf}(C_1^E)$;
3. if any of these tests fail, output NO; else output YES.
```

Figure 9. Algorithm for restricted entailment over atomic constraints.

$\alpha'\} \subseteq \textsc{nf}(C_1^E)$. Then we have $\rho(\alpha_1) = \rho(\alpha_2)$. Thus ρ' is well-defined. To see that $\rho' \vDash \textsc{nf}(C_2^E)$, notice that ρ' satisfies each constraint in $\textsc{nf}(C_2^E)$, *i.e.*, $\rho' \vDash \alpha = \beta$, $\rho' \vDash \alpha \Rightarrow \beta$, and $\rho' \vDash \alpha \Rightarrow \tau$. Thus $S(\textsc{nf}(C_1^E)) \mid_E \subseteq S(\textsc{nf}(C_2^E)) \mid_E$. By Lemma 3, we have $C_1 \vDash_E C_2$.

If the algorithm outputs NO, at least one of the checks fails in step 2 of the algorithm. We consider the three cases separately.

Case 1. Suppose $\alpha = \beta \in \textsc{nf}(C_2^E)$ but $\alpha = \beta \notin \textsc{nf}(C_1^E)$. Either $\{\alpha \mapsto \top, \beta \mapsto \bot\}$ or $\{\alpha \mapsto \bot, \beta \mapsto \top\}$ can be extended to a valuation $\rho \vDash \textsc{nf}(C_1^E)$. However, clearly $\rho \mid_E$ cannot be extended to a $\rho' \vDash \textsc{nf}(C_2^E)$. Thus $C_1 \vDash_E C_2$ does not hold.

Case 2. Suppose $\alpha \Rightarrow \beta \in \textsc{nf}(C_2^E)$, and $\alpha = \beta \notin \textsc{nf}(C_1^E)$ and $\alpha \Rightarrow \beta \notin \textsc{nf}(C_1^E)$. One can show that $\{\alpha \mapsto \top, \beta \mapsto \bot\}$ can be extended to a valuation $\rho \vDash \textsc{nf}(C_1^E)$. Thus, $C_1 \vDash_E C_2$ does not hold.

Case 3. Suppose $\{\alpha \Rightarrow \tau, \beta \Rightarrow \tau\} \subseteq \textsc{nf}(C_2^E)$, and $(\alpha = \beta) \notin \textsc{nf}(C_1^E)$ and $\{\alpha \Rightarrow \tau', \beta \Rightarrow \tau'\} \nsubseteq \textsc{nf}(C_1^E)$ for any $\tau' \notin E$. The partial valuation $\{\alpha \mapsto \top, \beta \mapsto \bot \to \bot\}$ can be extended to a valuation ρ that satisfies C_1. In particular, we construct ρ as follows

$$\begin{cases} \rho(\alpha) = \top & \\ \rho(\beta) = \bot \to \bot & \\ \rho(\gamma) = \rho(\alpha) & \text{if } \textsc{scc}(\gamma) = \textsc{scc}(\alpha) \\ \rho(\gamma) = \rho(\alpha) & \text{if } \textsc{scc}(\alpha) \Rightarrow \textsc{scc}(\gamma) \\ \rho(\gamma) = \rho(\beta) & \text{if } \textsc{scc}(\beta) = \textsc{scc}(\gamma) \\ \rho(\gamma) = \rho(\beta) & \text{if } \textsc{scc}(\beta) \Rightarrow \textsc{scc}(\gamma) \\ \rho(\gamma) = \bot & \text{otherwise} \end{cases}$$

where SCC is defined as the representative of a SCC component. In choosing a SCC for a SCC, the variables in E have precedence over the internal variables.

One can show that the constructed valuation ρ satisfies C_1. However $\{\alpha \mapsto \top, \beta \mapsto \bot \to \bot\}$ cannot be extended to a valuation that satisfies C_2 since this would require unifying $\top$ with $\bot \to \bot$.

□

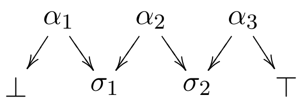
5 Restricted Entailment over Conditional Equality Constraints

In this section, we give a polynomial time algorithm for restricted entailment over conditional constraints.

Consider the following example term graph for the constraints

$$\{\alpha_1 \Rightarrow \bot, \alpha_1 \Rightarrow \sigma_1, \alpha_2 \Rightarrow \sigma_1, \alpha_2 \Rightarrow \sigma_2, \alpha_3 \Rightarrow \sigma_2, \alpha_3 \Rightarrow \top\}.$$

Example 3.



$$\begin{aligned}
S(\{\alpha_1 \Rightarrow \bot, \alpha_2 \Rightarrow \sigma_1\}) &= \{\langle v_1, v_2, v_3\rangle \mid v_1 = \bot\} \\
S(\{\alpha_2 \Rightarrow \sigma_1, \alpha_2 \Rightarrow \sigma_2\}) &= S^* \\
S(\{\alpha_3 \Rightarrow \sigma_2, \alpha_3 \Rightarrow \top\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_3 = \bot) \vee (v_3 = \top)\} \\
S(\{\alpha_1 \Rightarrow \bot, \alpha_2 \Rightarrow \sigma_1\}) &= \{\langle v_1, v_2, v_3\rangle \mid v_1 = \bot\} \\
S(\{\alpha_1 \Rightarrow \bot, \alpha_2 \Rightarrow \sigma_2\}) &= \{\langle v_1, v_2, v_3\rangle \mid v_1 = \bot\} \\
S(\{\alpha_1 \Rightarrow \bot, \alpha_3 \Rightarrow \sigma_2\}) &= \{\langle v_1, v_2, v_3\rangle \mid v_1 = \bot\} \\
S(\{\alpha_1 \Rightarrow \bot, \alpha_3 \Rightarrow \top\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_1 = \bot \wedge v_3 = \bot) \vee (v_1 = \bot \wedge v_3 = \top)\} \\
S(\{\alpha_1 \Rightarrow \sigma_1, \alpha_2 \Rightarrow \sigma_1\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_1 = \bot) \vee (v_2 = \bot) \vee (v_2 = v_3)\} \\
S(\{\alpha_1 \Rightarrow \sigma_1, \alpha_2 \Rightarrow \sigma_2\}) &= S^* \\
S(\{\alpha_1 \Rightarrow \sigma_1, \alpha_3 \Rightarrow \sigma_2\}) &= S^* \\
S(\{\alpha_1 \Rightarrow \sigma_1, \alpha_3 \Rightarrow \top\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_3 = \bot) \vee (v_3 = \top)\} \\
S(\{\alpha_2 \Rightarrow \sigma_1, \alpha_3 \Rightarrow \sigma_2\}) &= S^* \\
S(\{\alpha_2 \Rightarrow \sigma_1, \alpha_3 \Rightarrow \top\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_3 = \bot) \vee (v_3 = \top)\} \\
S(\{\alpha_2 \Rightarrow \sigma_2, \alpha_3 \Rightarrow \sigma_2\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_2 = \bot) \vee (v_3 = \bot) \vee (v_2 = v_3)\} \\
S(\{\alpha_2 \Rightarrow \sigma_2, \alpha_3 \Rightarrow \top\}) &= \{\langle v_1, v_2, v_3\rangle \mid (v_3 = \bot) \vee (v_3 = \top)\}
\end{aligned}$$

Figure 10. Solutions for all subsets of two constraints.

Notice that the solutions of the constraints in Example 3 with respect to $\{\alpha_1, \alpha_2, \alpha_3\}$ are all tuples $\langle v_1, v_2, v_3\rangle$ that satisfy

$$(v_1 = \bot \wedge v_3 = \bot) \vee (v_1 = \bot \wedge v_2 = \bot \ \wedge v_3 = \top) \vee (v_1 = \bot \wedge v_2 = \top \ \wedge v_3 = \top)$$

Now suppose we do the following: we take pairs of constraints, find their solutions with respect to $\{\alpha_1, \alpha_2, \alpha_3\}$, and take the intersection of the solutions. Let S^* denote the set of all valuations. Figure 10 shows the solutions for all the subsets of two constraints with respect to $\{\alpha_1, \alpha_2, \alpha_3\}$. One can show that the intersection of these solutions is the same as the solution for all the constraints. Intuitively, the solutions of a system of conditional constraints can be characterized by considering all pairs of constraints independently. We can make this intuition formal by putting some additional requirements on the constraints.

For simplicity, in later discussions, we consider the language without $\top$. With some extra checks, the presented algorithm can be adapted to include $\top$ in the language.

Here is the route we take to develop a polynomial time algorithm for restricted entailment over conditional constraints.

Section 5.1
We introduce a notion of a *closed system* and show that closed systems have the property that it is sufficient to consider pairs of conditional constraints in determining the solutions of the complete system with respect to the interface variables.

Section 5.2
We show that restricted entailment with a pair of conditional constraints can be decided in polynomial time, *i.e.*, $C \models_E C_= \cup \{c_1, c_2\}$ can be decided in polynomial time, where $C_=$ consists of equality constraints, and c_1 and c_2 are conditional constraints.

Section 5.3
We show how to reduce restricted entailment to restricted entailment in terms of closed systems. In particular, we show how to reduce $C_1 \models_E C_2$ to $C_1' \models_{E'} C_2'$ where C_2' is closed.

Combining the results, we arrive at a polynomial time algorithm for restricted entailment over conditional constraints.

5.1 Closed Systems

We define the notion of a closed system and show the essential properties of closed systems for entailment. Before presenting the definitions, we first demonstrate the idea with the example in Figure 11a. Let C denote the constraints in this example, with α and β the interface variables, and σ, σ_1, and σ_2 the internal variables. The intersection of the solutions of all the pairs of constraints is: α is either $\bot$ or $\tau \to \bot$, and β is either $\bot$ or $\tau' \to \bot$ for some τ and τ'. However, the solutions of C require that if $\alpha = \tau \to \bot$ and $\beta = \tau' \to \bot$, and both τ and τ' are non-$\bot$, then $\tau = \tau'$, *i.e.*, $\alpha = \beta$. Thus the intersection of solutions of pairs of constraints contains more valuations than the solution set of the entire system. The reason is that when we consider the set $\{\sigma_1 \Rightarrow \sigma, \sigma_2 \Rightarrow \sigma\}$, the solutions w.r.t. $\{\alpha, \beta\}$ are all valuations. We lose the information that α and β need to be the same in their domain.

We would like to consider σ_1 and σ_2 as interface variables if $\sigma_1 \neq \bot \neq \sigma_2$. We introduce some constraints and new interface variables into the system to *close* it. The modified constraint system is shown in Figure 11b. To make explicit the relationship between α and β, two variables α_1 and β_1 (interface variables corresponding to σ_1 and σ_2, respectively) are created with the constraints $\alpha_1 \Rightarrow \sigma$ and $\beta_1 \Rightarrow \sigma$. With this modification, the intersection of solutions of pairs of constraints w.r.t. $\{\alpha, \beta, \alpha_1, \beta_1\}$ is the same as the solution of the modified system. Restricting this intersection w.r.t. $\{\alpha, \beta\}$ we get the solution of the original constraint system. We next show how to systematically close a constraint system.

Definition 6 (TR). Consider a constraint $\alpha \Rightarrow \tau$ with the variable σ a *proper* subexpression of τ. We define a transformation TR on $\alpha \Rightarrow \tau$ over the structure of τ

- $\text{TR}(\sigma, \alpha \Rightarrow \sigma \to \tau') = \{\alpha \Rightarrow \alpha_1 \to \sigma_1\}$;
- $\text{TR}(\sigma, \alpha \Rightarrow \tau' \to \sigma) = \{\alpha \Rightarrow \sigma_2 \to \alpha_2\}$;
- $\text{TR}(\sigma, \alpha \Rightarrow \tau_1 \to \tau_2) =$
  $\begin{cases} \{\alpha \Rightarrow \alpha_1 \to \sigma_1\} \cup \text{TR}(\sigma, \alpha_1 \Rightarrow \tau_1) \text{ if } \sigma \in \text{Var}(\tau_1) \\ \{\alpha \Rightarrow \sigma_2 \to \alpha_2\} \cup \text{TR}(\sigma, \alpha_2 \Rightarrow \tau_2) \text{ otherwise} \end{cases}$
  Note if σ appears in both τ_1 and τ_2, TR is applied only to the occurrence of σ in τ_1.
- $\text{TR}(\sigma, \alpha = \tau) = \text{TR}(\sigma, \alpha \Rightarrow \tau)$.

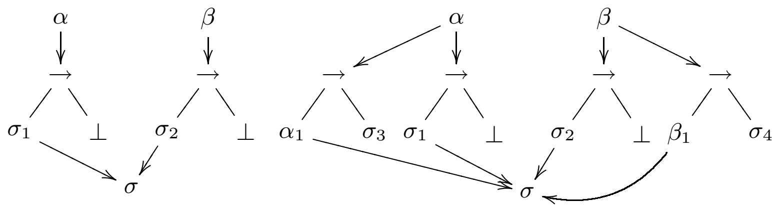
The variables α_i's and σ_i's are fresh. The newly created α_i's are called *auxiliary variables*. The variables α_i in the first two cases are called the *matching variable for* σ. The variable α is called the *root* of α_i, and is denoted by $\text{ROOT}(\alpha_i)$.

For each auxiliary variable α_i, we denote by $C_{\text{TR}}(\alpha_i)$ the TR constraints accumulated till α_i is created.

Putting this definition to use on the constraint system in Figure 11a, $\text{TR}(\sigma_1, \alpha \Rightarrow \sigma_1 \to \bot)$ yields the constraint $\alpha \Rightarrow \alpha_1 \to \sigma_3$ (shown in Figure 11b).

To understand the definition of $C_{\text{TR}}(\alpha_i)$, consider $\text{TR}(\sigma, \alpha \Rightarrow ((\sigma \to \bot) \to \bot)) = \{\alpha \Rightarrow \alpha_1 \to \sigma_1, \alpha_1 \Rightarrow \alpha_2 \to \sigma_2\}$, where α_1 and α_2 are the auxiliary variables. We have $C_{\text{TR}}(\alpha_1) = \{\alpha \Rightarrow \alpha_1 \to \sigma_1\}$ and $C_{\text{TR}}(\alpha_2) = \{\alpha \Rightarrow \alpha_1 \to \sigma_1, \alpha_1 \Rightarrow \alpha_2 \to \sigma_2\}$.

Definition 7 (Closed Systems). A system of conditional constraints C' is *closed* w.r.t. a set of variables E in C after the following steps:



(a) Example system. (b) Example system closed.

Figure 11. An example constraint system and its closed system.

1. Let $C' = \textsc{CondResolve}(C)$.
2. Set W to E.
3. For each variable $\alpha \in W$, if $\alpha \Rightarrow \tau$ is in C', where $\sigma \in \mathrm{Var}(\tau)$, and $\sigma \Rightarrow \tau' \in C'$, add $\mathrm{TR}(\sigma, \alpha \Rightarrow \tau)$ to C'. Let α' be the matching variable for σ and add $\alpha' \Rightarrow \tau'$ to C'.
4. Set W to the set of auxiliary variables created in Step 3 and repeat Step 3 until W is empty.

Step 3 of this definition warrants explanation. In the example $\mathrm{TR}(\sigma_1, \alpha \Rightarrow \sigma_1)$ we add the constraint $\alpha \Rightarrow \alpha_1 \to \sigma_3$ with α_1 as the matching variable for σ_1. We want to ensure that α_1 and σ_1 are actually the same, so we add the constraint $\alpha_1 \Rightarrow \sigma$. This process must be repeated to expose all such internal variables (such as σ_1 and σ_2).

Next we give the definition of a *forced variable*. Given a valuation ρ for the interface variables, if an internal variable σ is determined already by ρ, then σ is *forced by* ρ. For example, in Figure 11a, if α is non-$\perp$, then the value of σ_1 is forced by α.

Definition 8 (Forced Variables). We say that an internal variable σ is *forced* by a valuation ρ if any one of the following holds (A is the set of auxiliary variables)

- $\mathrm{ECR}(\sigma) = \perp$;
- $\mathrm{ECR}(\sigma) = \alpha$, where $\alpha \in E \cup A$;
- $\mathrm{ECR}(\sigma) = \tau_1 \to \tau_2$;
- $\rho(\alpha) \neq \perp$ and $\alpha \Rightarrow \tau$ is a constraint where $\sigma \in \mathrm{Var}(\tau)$ and $\alpha \in E \cup A$;
- σ' is forced by ρ to a non-$\perp$ value and $\sigma' \Rightarrow \tau$ is a constraint where $\sigma \in \mathrm{Var}(\tau)$.

Theorem 5. Let C be a closed system of constraints w.r.t. a set of interface variables E, and let A be the set of auxiliary variables of C. Let $C_=$ and $C_\Rightarrow$ be the systems of equality constraints and conditional constraints respectively. Then

$$S(C)\mid_{E\cup A} = \bigcap_{c_i, c_j \in C_\Rightarrow} S(C_= \cup \{c_i, c_j\})\mid_{E\cup A} .$$

In other words, it suffices to consider pairs of conditional constraints in determining the solutions of a closed constraint system.

Proof. Since C contains all the constraints in $C_= \cup \{c_i, c_j\}$ for all i and j, thus it follows that

$$S(C)\mid_{E\cup A} \subseteq \bigcap_{c_i, c_j \in C_\Rightarrow} S(C_= \cup \{c_i, c_j\})\mid_{E\cup A} .$$

It remains to show

$$S(C)\mid_{E\cup A} \supseteq \bigcap_{c_i, c_j \in C_\Rightarrow} S(C_= \cup \{c_i, c_j\})\mid_{E\cup A} .$$

Let ρ be a valuation in $\bigcap_{c_i,c_j \in C_\Rightarrow} S(C_= \cup \{c_i, c_j\}) \mid_{E \cup A}$. It suffices to show that ρ can be extended to a satisfying valuation ρ' for C. To show this, it suffices to find an extension ρ' of ρ for C such that $\rho' \vDash C_= \cup \{c_i, c_j\}$ for all i and j.

Consider the valuation ρ' obtained from ρ by mapping all the internal variables not forced by ρ (in C) to $\bot$. The valuation ρ' can be uniquely extended to satisfy C if for any c_i and c_j, c'_i and c'_j, if σ is forced by ρ in both $C_= \cup \{c_i, c_j\}$ and $C_= \cup \{c'_i, c'_j\}$, then it is forced to the same value in both systems. The value that σ is forced to by ρ is denoted by $\rho^!(\sigma)$.

We prove by cases (cf. Definition 8) that if σ is forced by ρ, it is forced to the same value in pairs of constraints. Let $C_{i,j}$ denote $C_= \cup \{c_i, c_j\}$ and $C_{i',j'}$ denote $C_= \cup \{c'_i, c'_j\}$.

- If $\textsc{ecr}(\sigma) = \bot$, then σ is forced to the same value, i.e., $\bot$, because $\sigma = \bot \in C_=$.
- If $\textsc{ecr}(\sigma) = \alpha$, with $\alpha \in E \cup A$, then σ is forced to $\rho(\alpha)$ in both systems, because $\sigma = \alpha \in C_=$.
- If $\textsc{ecr}(\sigma) = \tau_1 \to \tau_2$, one can show that ρ forces σ to the same value with an induction over the structure of $\textsc{ecr}(\sigma)$ (with the two cases above as base cases).
- Assume σ is forced in $C_{i,j}$ because $\alpha \Rightarrow \tau_1 \in C_{i,j}$ with $\rho(\alpha) \neq \bot$ and forced in $C_{i',j'}$ because $\beta \Rightarrow \tau_2 \in C_{i',j'}$ with $\rho(\beta) \neq \bot$. For each extension ρ_1 of ρ with $\rho_1 \vDash C_{i,j}$, and for each extension ρ_2 of ρ with $\rho_2 \vDash C_{i',j'}$, we have

  $$\rho(\alpha) = \rho_1(\alpha) = \rho_1(\tau_1)$$
  $$\rho(\beta) = \rho_2(\beta) = \rho_2(\tau_2)$$

  Consider the constraint system $C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$. The valuation ρ can be extended to ρ_3 with $\rho_3 \vDash C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$. Thus we have

  $$\rho(\alpha) = \rho_3(\alpha) = \rho_3(\tau_1)$$
  $$\rho(\beta) = \rho_3(\beta) = \rho_3(\tau_2)$$

  Therefore, $\rho_1(\tau_1) = \rho_3(\tau_1)$ and $\rho_2(\tau_2) = \rho_3(\tau_2)$. Hence, $\rho_1(\sigma) = \rho_3(\sigma)$ and $\rho_2(\sigma) = \rho_3(\sigma)$, which imply $\rho_1(\sigma) = \rho_2(\sigma)$. Thus σ is forced to the same value.
- Assume σ is forced in $C_{i,j}$ because σ_1 is forced to a non-$\bot$ value and $\sigma_1 \Rightarrow \tau_1 \in C_{i,j}$ and is forced in $C_{i',j'}$ because σ_2 is forced to a non-$\bot$ value and $\sigma_2 \Rightarrow \tau_2 \in C_{i',j'}$. Because C is a closed system, we must have two interface variables or auxiliary variables α and β with both $\alpha \Rightarrow \tau_1$ and $\beta \Rightarrow \tau_2$ appearing in C. Since σ_1 and σ_2 are forced, then we must have $\rho(\alpha) = \rho^!(\sigma_1)$ and $\rho(\beta) = \rho^!(\sigma_2)$, thus σ must be forced to the same value by the previous case.
- Assume σ is forced in $C_{i,j}$ because $\rho(\alpha) \neq \bot$ and $\alpha \Rightarrow \tau_1 \in C_{i,j}$ and forced in $C_{i',j'}$ because σ_2 is forced to a non-$\bot$ value and $\sigma_2 \Rightarrow \tau_2 \in C_{i',j'}$. This case is similar to the previous case.
- The remaining case, where σ is forced in $C_{i,j}$ because σ_1 is forced to a non-$\bot$ value and $\sigma_1 \Rightarrow \tau_1 \in C_{i,j}$ and is forced in $C_{i',j'}$ because $\rho(\alpha) \neq \bot$ and $\alpha \Rightarrow \tau_2 \in C_{i',j'}$, is symmetric to the above case.

□

5.2 Entailment of Pair Constraints

In the previous subsection, we saw that a closed system can be decomposed into pairs of conditional constraints. In this section, we show how to efficiently determine entailment if the right-hand side consists of a pair of conditional constraints.

We first state a lemma (Lemma 4) which is important in finding a polynomial algorithm for entailment of pair constraints.

Lemma 4. Let C_1 be a system of conditional constraints and C_2 be a system of *equality* constraints with $E = \mathrm{Var}(C_1) \cap \mathrm{Var}(C_2)$. The decision problem $C_1 \models_E C_2$ is solvable in polynomial time.

Proof. Consider the following algorithm. We first solve C_1 using CondResolve, and add the terms appearing in C_2 to the resulting term graph for C_1. Then for any two terms appearing in the term graph, we decide, using the simple entailment algorithm in Figure 7, whether the two terms are the same. For terms which are equivalent we merge their equivalence classes. Next, for each of the constraints in C_2, we merge the left and right sides. For any two non-congruent classes that are unified, we require at least one of the representatives be a variable in $\mathrm{Var}(C_2) \setminus E$. If this requirement is not met, the entailment does not hold. Otherwise, the entailment holds.

If the requirement is met, then it is routine to verify that the entailment holds. Suppose the requirement is not met, *i.e.*, there exist two non-congruent classes which are unified and none of whose ECRs is a variables in $\mathrm{Var}(C_2) \setminus E$. Since the two classes are non-congruent, we can choose a satisfying valuation for C_1 which maps the two classes to different values (This is possible because, otherwise, we would have proven that they are the same with the simple entailment algorithm for conditional constraints.) The valuation $\rho \mid_E$ cannot be extended to a satisfying valuation for C_2 because, otherwise, this contradicts the fact that $C_1 \cup C_2$ entails the equivalence of the two non-congruent terms.

□

Theorem 6. Let C_1 be a system of conditional constraints. Let $C_=$ be a system of equality constraints. The following three decision problems can be solved in polynomial time:

1. $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$, where $\alpha, \beta \in E$.
2. $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \mu \Rightarrow \tau_2\}$, where $\alpha \in E$ and $\mu \notin E$.
3. $C_1 \models_E C_= \cup \{\mu_1 \Rightarrow \tau_1, \mu_2 \Rightarrow \tau_2\}$, where $\mu_1, \mu_2 \notin E$.

Proof.

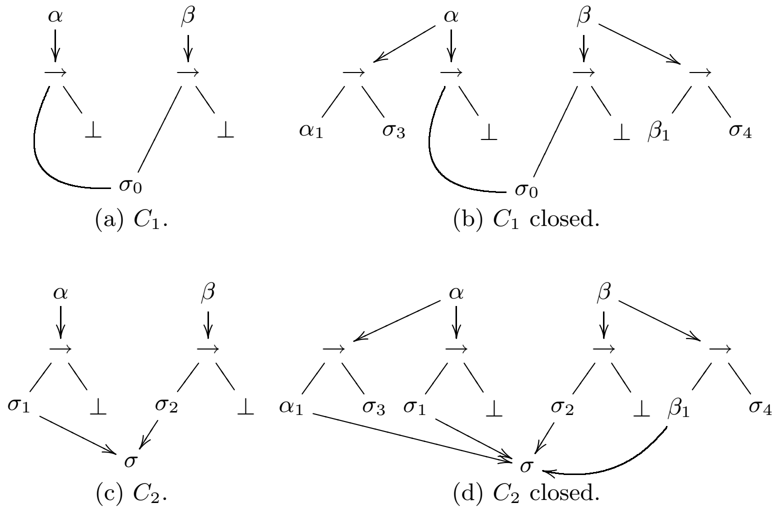
1. For the case $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$, notice that $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$ iff the following entailments hold
   - $C_1 \cup \{\alpha = \bot, \beta = \bot\} \models_E C_=$
   - $C_1 \cup \{\alpha = \bot, \beta = \nu_1 \rightarrow \nu_2\} \models_E C_= \cup \{\beta = \tau_2\}$
   - $C_1 \cup \{\alpha = \sigma_1 \rightarrow \sigma_2, \beta = \bot\} \models_E C_= \cup \{\alpha = \tau_1\}$
   - $C_1 \cup \{\alpha = \sigma_1 \rightarrow \sigma_2, \beta = \nu_1 \rightarrow \nu_2\} \models_E C_= \cup \{\alpha = \tau_1, \beta = \tau_2\}$

   where σ_1, σ_2, ν_1, and ν_2 are fresh variables not in $\mathrm{Var}(C_1) \cup \mathrm{Var}(C_2)$.

   Notice that each of the above entailments reduces to entailment of equality constraints, which can be decided in polynomial time by Lemma 4.
2. For the case $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \mu \Rightarrow \tau_2\}$, we consider two cases:
   - $C_1 \cup \{\alpha = \bot\} \models_E C_= \cup \{\mu \Rightarrow \tau_2\}$;
   - $C_1 \cup \{\alpha = \sigma_1 \rightarrow \sigma_2\} \models_E C_= \cup \{\alpha = \tau_1, \mu \Rightarrow \tau_2\}$

   where σ_1 and σ_2 are fresh variables not in $\mathrm{Var}(C_1) \cup \mathrm{Var}(C_2)$.

   We have a few cases.
   - $\text{ECR}(\mu) = \bot$
   - $\text{ECR}(\mu) = \tau_1 \rightarrow \tau_2$
   - $\text{ECR}(\mu) \in E$
   - $\text{ECR}(\mu) \notin E$


(a) C_1.

(b) C_1 closed.

(c) C_2.

(d) C_2 closed.

Figure 12. Example entailment.

Notice that the only interesting case is the last case ($\textsc{ecr}(\mu) \notin E$) when there is a constraint $\beta = \tau$ in $C_=$ and μ appears in τ. For this case, we consider all the $\mathcal{O}(n)$ resulted entailments by setting β to some appropriate value according to the structure of τ, *i.e.*, we consider all the possible values for β. For example, if $\tau = (\mu \to \bot) \to \mu$, we consider the following cases:

- $\beta = \bot$;
- $\beta = \bot \to \nu_1$;
- $\beta = (\bot \to \nu_2) \to \nu_1$;
- $\beta = ((\nu_3 \to \nu_4) \to \nu_2) \to \nu_1$

where ν_1,ν_2,ν_3, and ν_4 are fresh variables.

Each of the entailments will have only equality constraints on the right-hand side. Thus, these can all be decided in polynomial time. Together, the entailment can be decided in polynomial time.

3. For the case $C_1 \vDash_E C_= \cup \{\mu_1 \Rightarrow \tau_1, \mu_2 \Rightarrow \tau_2\}$, the same idea as in the second case applies as well. The sub-case which is slightly different is when, for example, μ_2 appears in τ_1 only. In this case, for some β and τ, $\beta = \tau$ is in $C_=$ where μ_1 occurs in τ. Let $\tau' = \tau[\tau_1/\mu_1]$, where $\tau[\tau_1/\mu_1]$ denotes the type obtained from τ by replacing each occurrence of μ_1 by τ_1. Again, we consider $\mathcal{O}(n)$ entailments with right-side an equality constraint system by assigning β appropriate values according to the structure of τ'. Thus this form of entailment can also be decided in polynomial time.

□

5.3 Reduction of Entailment to Closed Systems

We now reduce an entailment $C_1 \vDash_E C_2$ to entailment of closed systems, thus completing the construction of a polynomial time algorithm for restricted entailment over conditional constraints.

Unfortunately we cannot directly use the closed systems for C_1 and C_2 as demonstrated by the example in Figure 12. Figures 12a and 12c show two constraint systems C_1 and C_2. Suppose we want to decide $C_1 \vDash_{\{\alpha,\beta\}} C_2$. One can verify that the entailment does hold. Figures 12b and 12d show the closed systems for C_1 and C_2, which we name C_1' and C_2'. Note that we include the TR constraints of C_2 in C_1'. One can verify that the entailment $C_1' \vDash_{\{\alpha,\beta,\alpha_1,\beta_1\}} C_2'$ does not hold (take $\alpha = \beta = \bot$, $\alpha_1 = \bot \to \bot$, and $\beta_1 = \bot \to \top$, for example). The reason is that there is some information about α_1 and β_1 missing from C_1'. In particular, when both α_1 and β_1 are forced, we should have $\alpha_1 \Rightarrow \sigma'$ and $\beta_1 \Rightarrow \sigma'$ (actually in this case they satisfy the stronger relation that $\alpha_1 = \beta_1$). By replacing $\alpha \Rightarrow \alpha_1 \to \sigma_3$ and $\beta \Rightarrow \beta_1 \to \sigma_4$ with $\alpha = \alpha_1 \to \sigma_3$ and $\beta = \beta_1 \to \sigma_4$ (because that is when both are forced), we can decide that $\alpha_1 = \beta_1$. The following definition of a *completion* does exactly what we have described.

Definition 9 (Completion). Let C be a closed constraint system of C_0 w.r.t. E. Let A be the set of auxiliary variables. For each pair of variables α_i and β_j in A, let $C(\alpha_i, \beta_j) = C_{\mathrm{TR}}(\alpha_i) \cup C_{\mathrm{TR}}(\beta_j)$ (see Definition 6) and $C^{=}(\alpha_i, \beta_j)$ be the equality constraints obtained by replacing $\Rightarrow$ with $=$ in $C(\alpha_i, \beta_j)$. Decide whether $C \cup C^{=}(\alpha_i, \beta_j) \vDash_{\{\alpha_i,\beta_j\}} \{\alpha_i \Rightarrow \sigma, \beta_j \Rightarrow \sigma\}$ (cf. Theorem 6). If the entailment holds, add the constraints $\alpha_i \Rightarrow \sigma_{(\alpha_i,\beta_j)}$ and $\beta_j \Rightarrow \sigma_{(\alpha_i,\beta_j)}$ to C, where $\sigma_{(\alpha_i,\beta_j)}$ is a fresh variable unique for α_i and β_j. The resulting constraint system is called the *completion* of C.

Theorem 7. Let C_1 and C_2 be two conditional constraint systems. Let C_2' be the closed system of C_2 w.r.t. to $E = \mathrm{Var}(C_1) \cap \mathrm{Var}(C_2)$ with A the set of auxiliary variables. Construct the closed system for C_1 w.r.t. E with A' the auxiliary variables, and add the TR constraints of closing C_2 to C_1 after closing C_1. Let C_1' be the completion of modified C_1. We have $C_1 \vDash_E C_2$ iff $C_1' \vDash_{E\cup A\cup A'} C_2'$.

Proof.
($\Leftarrow$): Assume $C_1' \vDash_{E\cup A\cup A'} C_2'$. Let $\rho \vDash C_1$. We can extend ρ to ρ' which satisfies C_1'. Since $C_1' \vDash_{E\cup A\cup A'}$ C_2', then there exists ρ'' such that $\rho'' \vDash C_2'$ with $\rho' |_{E\cup A\cup A'} = \rho'' |_{E\cup A\cup A'}$. Since $\rho'' \vDash C_2'$, we have $\rho'' \vDash C_2$. Also $\rho |_E = \rho' |_E = \rho'' |_E$. Therefore, $C_1 \vDash_E C_2$.
($\Rightarrow$): Assume $C_1 \vDash_E C_2$. Let $\rho \vDash C_1'$. Then $\rho \vDash C_1$. Thus there exists $\rho' \vDash C_2$ with $\rho |_E = \rho' |_E$. We extend $\rho' |_E$ to ρ'' with $\rho''(\alpha) = \rho'(\alpha)$ if $\alpha \in E$ and $\rho''(\alpha) = \rho(\alpha)$ if $\alpha \in (A \cup A')$. It suffices to show that ρ'' can be extended with mappings for variables in $\mathrm{Var}(C_2') \setminus (E \cup A \cup A') = \mathrm{Var}(C_2') \setminus (E \cup A)$, because $\rho'' |_{E\cup A\cup A'} = \rho |_{E\cup A\cup A'}$.

Notice that all the TR constraints in C_2' are satisfied by some extension of ρ'', because they also appear in C_1'. Also the constraints C_2 are satisfied by some extension of ρ''. It remains to show that the internal variables of C_2' are forced by ρ'' to the same value if they are forced by ρ'' in either the TR constraints or C_2. Suppose there is an internal variable σ forced to different values by ρ''. W.L.O.G., assume that σ is forced by ρ'' because $\rho''(\alpha_i) \neq \bot$ and $\alpha_i \Rightarrow \sigma$ and forced because $\rho''(\beta_j) \neq \bot$ and $\beta_j \Rightarrow \sigma$ for some interface or auxiliary variables α_i and β_j. Consider the interface variables $\mathrm{ROOT}(\alpha_i)$ and $\mathrm{ROOT}(\beta_j)$ (see Definition 6). Since the completion of C_1 does not include constraints $\{\alpha_i \Rightarrow \sigma', \beta_j \Rightarrow \sigma'\}$, thus we can assign $\mathrm{ROOT}(\alpha_i)$ and $\mathrm{ROOT}(\beta_j)$ appropriate values to force α_i and β_j to different non-$\bot$ values. However, C_2 requires α_i and β_j to have the same non-$\bot$ value. Thus, if there is an internal variable σ forced to different values by ρ'', we can construct a valuation which satisfies C_1, but the valuation restricted to E cannot be extended to a satisfying valuation for C_2. This contradicts the assumption that $C_1 \vDash_E C_2$. To finish the construction of a desired extension of ρ'' that satisfies C_2', we set the variables which are not forced to $\bot$.

One can easily verify that this valuation must satisfy C_2'. Hence $C_1' \vDash_{E\cup A\cup A'} C_2'$.

□

5.4 Putting Everything Together

Theorem 8. The restricted entailment for conditional constraints can be decided in polynomial time.

Proof. Consider the problem $C_1 \models_E C_2$. By Theorem 7, it is equivalent to testing $C'_1 \models_{E \cup A \cup A'} C'_2$ (see Theorem 7 for the appropriate definitions of C'_1, C'_2, A, and A'). Notice that C'_1 and C'_2 are constructed in polynomial time in sizes of C_1 and C_2. Now by Theorem 5, this is equivalent to checking $\mathcal{O}(n^2)$ entailment problems of the form $C'_1 \models_{E \cup A \cup A'} C_{2'_{=}} \cup \{c_i, c_j\}$, where $C_{2'_{=}}$ denote the equality constraints of C'_2 and c_i and c_j are two conditional constraints of C'_2. And by Theorem 6, we can decide each of these entailments in polynomial time. Putting everything together, we have a polynomial time algorithm for restricted entailment over conditional constraints.

□

6 Extended Conditional Constraints

In this section, we consider a natural extension of the standard conditional constraint language. This section is helpful for a comparison between this constraint language with the standard conditional constraint language, which we consider in Section 4. The results in this section provides a clear boundary between tractable and intractable constraint languages in terms of entailments.

We extend the language with the following construct extending the conditional constraints used in previous sections. The new construct is

$$\alpha \Rightarrow (\tau_1 = \tau_2),$$

which holds iff either $\alpha = \bot$ or $\tau_1 = \tau_2$. We call this form of constraints *extended conditional equality constraints.*

To see that this construct indeed extends $\alpha \Rightarrow \tau$, notice that $\alpha \Rightarrow \tau$ can be encoded in the new constraint language as

$$\alpha \Rightarrow (\alpha = \tau).$$

This extension is interesting because many equality based program analyses can be naturally expressed with this form of constraints. An example analysis that uses this form of constraints is the equality based flow analysis for higher order functional languages [25]. Additionally it can be used as a boundary for separating tractable and intractable constraint languages.

Note that satisfiability for this extension can still be decided in almost linear time with basically the same algorithm outlined for conditional equality constraints. We consider both simple entailment and restricted entailment for this extended language.

6.1 Simple Entailment

The algorithm is given in Figure 13. We give the basic idea of the algorithm. We consider the following cases:

1. $\alpha = \bot$, $\beta = \top$;
2. $\alpha = \bot$, $\beta = \beta_1 \rightarrow \beta_2$;
3. $\alpha = \top$, $\beta = \bot$;
4. $\alpha = \top$, $\beta = \beta_1 \rightarrow \beta_2$;
5. $\alpha = \alpha_1 \rightarrow \alpha_2$, $\beta = \bot$;
6. $\alpha = \alpha_1 \rightarrow \alpha_2$, $\beta = \top$;
7. $\alpha = \alpha_1 \rightarrow \alpha_2$, $\beta = \beta_1 \rightarrow \beta_2$.

```
$\textsc{se}(C, \alpha = \beta) = \textsc{yes}$ iff $C \vDash \alpha = \beta$.

1. Solve $C$. If $\textsc{ecr}(\alpha) = \textsc{ecr}(\beta)$, return YES, else continue.
2. If $C \cup \{\alpha = \bot, \beta = \top\}$ is satisfiable, return NO, else continue.
3. If $C \cup \{\alpha = \bot, \beta = \beta_1 \to \beta_2\}$ is satisfiable, return NO, else continue.
4. If $C \cup \{\alpha = \top, \beta = \bot\}$ is satisfiable, return NO, else continue.
5. If $C \cup \{\alpha = \top, \beta = \beta_1 \to \beta_2\}$ is satisfiable, return NO, else continue.
6. If $C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2\}$ is unsatisfiable, return YES, else continue.
7. If $\textsc{se}(C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2\}, \alpha_1 = \beta_1)$ returns NO, then return NO, else continue.
8. If $\textsc{se}(C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2, \underline{\alpha_1 = \beta_1}\}, \alpha_2 = \beta_2)$ returns NO, then return NO, else return
   YES.
```

Figure 13. Algorithm for simple entailment over extended conditional equality constraints.

For the first six cases, if adding any of the corresponding constraints to C makes the constraint system satisfiable, then $C \nvDash \alpha = \beta$. For the last case where $\alpha = \alpha_1 \to \alpha_2$ and $\beta_1 \to \beta_2$, if the constraints make C unsatisfiable, then $C \vDash \alpha = \beta$. If the constraints make C satisfiable, then we recurse with $C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2\} \vDash \alpha_1 = \beta_1$ and $C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2\} \vDash \alpha_2 = \beta_2$. This naive algorithm may run in exponential time, we show, in the algorithm, how to use precomputed information to reduce the algorithm to polynomial time. The detailed algorithm is given in Figure 7. Step 8 of the algorithm is the key to reduce the running time from exponential to polynomial. The idea is if in Step 7, we know that $\alpha_1 = \beta_1$, then this information can be used in showing that $\alpha_2 = \beta_2$.

Proof. [Sketch]

The algorithm is obviously correct. We need to verify that the algorithm runs in polynomial time in the size of C. We briefly describe the justification for the polynomial running time. First notice that the depth that the algorithm recurses is at most $O(|C|)$ times, since each time two conditional constraints are changed to equality constraints and there are at most $O(|C|)$ number of conditional constraints. However the total number of recursive calls may be exponential. However, the second argument of each recursive call are two terms from the original term graph for C (except maybe the last call in which case the algorithm outputs NO). There are $O(|C|^2)$ such term pairs. Consider the call $\textsc{se}(C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2\}, \alpha_1 = \beta_1)$, where $\textsc{ecr}(\alpha_1)$ and $\textsc{ecr}(\beta_1)$ are two terms from C. If it returns YES, then $\textsc{se}(C \cup \{\alpha = \alpha_1 \to \alpha_2, \beta = \beta_1 \to \beta_2, \alpha_1 = \beta_1\}, \alpha_2 = \beta_2)$ can return YES immediately without repeating the computation if $\textsc{ecr}(\alpha_2) = \textsc{ecr}(\alpha_1)$ and $\textsc{ecr}(\beta_2) = \textsc{ecr}(\beta_1)$. With this observation, we conclude that the running time of the algorithm is polynomial in $|C|$.

□

6.2 Restricted Entailment

In this subsection, we consider the restricted entailment for extended conditional constraints. We show that the decision problem $C_1 \vDash_E C_2$ for extended conditional constraints is coNP-complete.

We define the decision problem NENT as the problem of deciding whether $C_1 \nvDash_E C_2$, where C_1 and C_2 are systems of extended conditional equality constraints and $E = \mathrm{Var}(C_1) \cap \mathrm{Var}(C_2)$.

Theorem 9. The decision problem NENT for extended conditional constraints is in NP.

Proof.

Let C_1 and C_2 be two extended constraint systems. Let $E = \mathrm{Var}(C_1) \cap \mathrm{Var}(C_2)$.

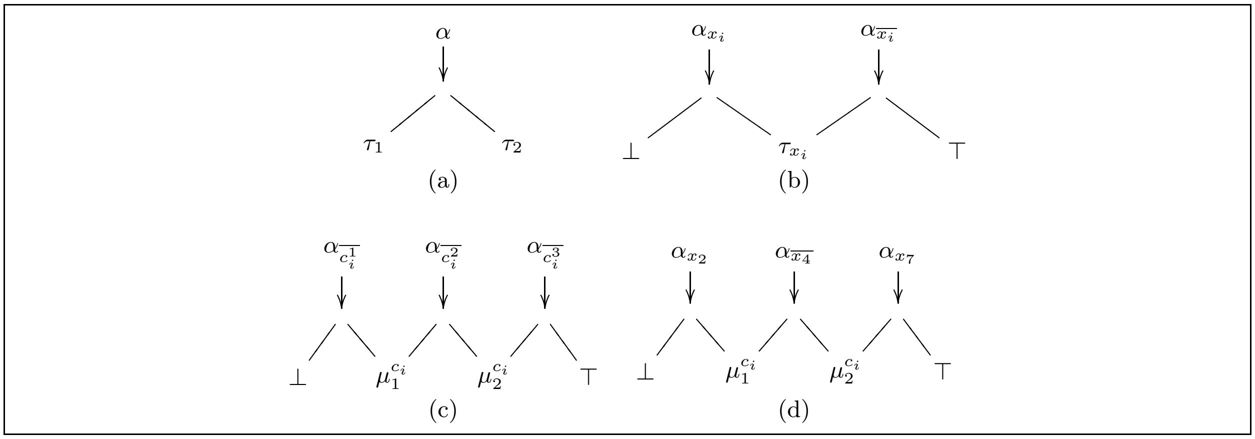


Figure 14. Graph representations of constraints.

For each variable α in $\mathrm{Var}(C_1)$, we guess whether α is $\bot$, $\top$, or $\alpha_1 \to \alpha_2$ for some fresh variables α_1 and α_2. We add these constraints to C_1 to obtain C_1'. For each α in E, we add to C_2 the constraints $\alpha = \bot$, $\alpha = \top$, or $\alpha = \alpha_1 \to \alpha_2$ depending on what we guessed for C_1. Notice now that C_1' is a system of equality constraints. C_2', however, may still have some conditional constraints. This means that our guess for the variables E needs to be refined to get rid of these conditional constraints in C_2'. In C_2', for each conditional constraint with a fresh variable α_i (the generated variables) as antecedent, we guess the value of α_i and add the corresponding constraints to both C_1' and C_2'. This process is repeated until there are no more conditional constraints in C_2' with any fresh variables as antecedents. Since there are at most $\mathcal{O}(|C_2|)$ number of conditional constraints in C_2, thus we make at most $\mathcal{O}(|C_2|)$ number of guesses. Finally, conditional constraints with variables in $\mathrm{Var}(C_2) \setminus E$ as antecedents are discarded since these constraints do not affect the solutions of the constraints w.r.t. E.

Let C_1'' and C_2'' be the resulting constraint systems. Notice they are equality constraints. Thus at the end, we turn the problem into entailment over equality constraints, which we can decide in polynomial time. The guessing step takes time polynomial in $|C_1|$ and $|C_2|$. Thus NENT is in NP.

□

Next we show that the problem NENT is hard for NP, and thus an efficient algorithm is unlikely to exist for the problem. The reduction actually shows that with extended conditional constraints, even atomic restricted entailment [4] is coNP-hard.

Theorem 10. The decision problem NENT is NP-hard.

Proof. [Sketch] We reduce 3-CNFSAT to NENT. As mentioned, the reduction shows that even atomic restricted entailment over extended conditional constraints is coNP-complete.

Let ψ be a boolean formula in 3-CNF form and let $\{x_1, x_2, \ldots, x_n\}$ and $\{c_1, c_2, \ldots, c_m\}$ be the boolean variables and clauses in ψ respectively. For each boolean variable x_i in ψ, we create two term variables α_{x_i} and $\alpha_{\overline{x_i}}$, which we use to decide the truth value of x_i. The value $\bot$ is treated as the boolean value false and any non-$\bot$ value is treated as the boolean value true.

Note, in a graph, a constraint of the form $\alpha \Rightarrow (\tau_1 = \tau_2)$ is represented as Figure 14a.

[4] Only variables, $\bot$, and $\top$ are in the constraint system.

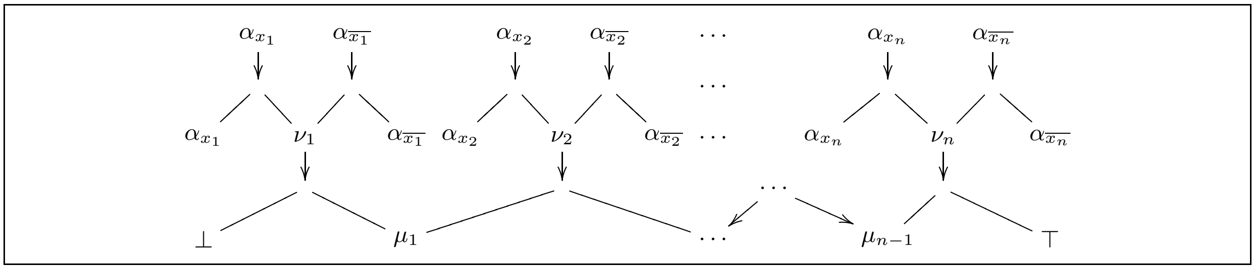


Figure 15. Constructed constraint system C_2.

First we need to ensure that a boolean variable takes on at most one truth value. We associate with each x_i constraints C_{x_i}, graphically represented as Figure 14b, where τ_{x_i} is some internal variable. These constraints guarantee that at least one of α_{x_i} and $\alpha_{\overline{x_i}}$ is $\perp$. These constraints still allow both α_{x_i} and $\alpha_{\overline{x_i}}$ to be $\perp$, which we deal with below.

In the following, let $\alpha_{\overline{\overline{x}}} = \alpha_x$. For each clause $c_i = c_i^1 \vee c_i^2 \vee c_i^3$ of ψ, we create constraints C_{c_i} that ensure every clause is satisfied by a truth assignment. A clause is satisfied if at least one of the literals is true, which is the same as saying that the negations of the literals cannot all be true simultaneously. The constraints are in Figure 14c, where $\mu_1^{c_i}$ and $\mu_2^{c_i}$ are internal variables associated with c_i. As an example consider $c_i = \overline{x_2} \vee x_4 \vee \overline{x_7}$. The constraints C_{c_i} are in Figure 14d.

We let C_1 be the union of all the constraints C_{x_i} and C_{c_j} for $1 \le i \le n$ and $1 \le j \le m$, *i.e.*,

$$C_1 = (\bigcup_{i=1}^{n} C_{x_i}) \cup (\bigcup_{j=1}^{m} C_{c_j})$$

There is one additional requirement that we want to enforce: not both α_{x_i} and $\alpha_{\overline{x_i}}$ are $\perp$. This cannot be enforced directly in C_1. We construct constraints for C_2 to enforce this requirement. The idea is that if for any x_i, the term variables α_{x_i} and $\alpha_{\overline{x_i}}$ are both $\perp$, then the entailment holds.

We now proceed to construct C_2. The constraints C_2 represented graphically are shown in Figure 15. In the constraints, all the variables except α_{x_i} and $\alpha_{\overline{x_i}}$ are internal variables. These constraints can be used to enforce the requirement that for all x_i at least one of α_{x_i} and $\alpha_{\overline{x_i}}$ is non-$\perp$. The intuition is that if α_{x_i} and $\alpha_{\overline{x_i}}$ are both $\perp$, the internal variable ν_i can be $\perp$, which breaks the chain of conditional dependencies along the bottom of Figure 15, allowing $\mu_1, \ldots, \mu_{i-1}$ to be set to $\perp$ and $\mu_i, \ldots, \mu_{n-1}$ to be set to $\top$.

We let the set of interface variables $E = \{\alpha_{x_i}, \alpha_{\overline{x_i}} \mid 1 \le i \le n\}$. One can show that ψ is satisfiable iff $C_1 \not\models_E C_2$. To prove the NP-hardness result, observe that the described reduction is a polynomial-time reduction. Thus, the decision problem NENT is NP-hard. We let the set of interface variables $E = \{\alpha_{x_i}, \alpha_{\overline{x_i}} \mid 1 \le i \le n\}$. One can show that ψ is satisfiable iff $C_1 \not\models_E C_2$. To prove the NP-hardness result, observe that the described reduction is a polynomial-time reduction. Thus, the decision problem NENT is NP-hard.

We let the set of interface variables $E = \{\alpha_{x_i}, \alpha_{\overline{x_i}} \mid 1 \le i \le n\}$. We claim that ψ is satisfiable iff $C_1 \not\models_E C_2$.

Claim. ψ is satisfiable iff $C_1 \not\models_E C_2$.

Proof.

Assume that ψ is satisfiable. Let $f : \{x_i \mid 1 \le i \le n\} \to \{0, 1\}$ be a satisfying assignment for ψ. We construct from f a valuation ρ for the variables E such that ρ can be extended to a satisfying valuation for C_1 while it cannot be extended to a satisfying valuation for C_2. The existence of such a ρ is sufficient to conclude that $C_1 \not\models_E C_2$. We construct ρ as follows

$$\begin{cases} \rho(\alpha_{x_i}) = \bot \ \wedge \ \rho(\alpha_{\overline{x_i}}) = \top \text{ if } f(x_i) = 0 \\ \rho(\alpha_{x_i}) = \top \ \wedge \ \rho(\alpha_{\overline{x_i}}) = \bot \text{ if } f(x_i) = 1 \end{cases}$$

The valuation ρ can be extended to a satisfying valuation ρ_1 for C_1. For each boolean variable x_i, there is a unique way to extend ρ to satisfy the constraints in C_{x_i}, namely $\rho_1(\tau_{x_i}) = \bot$ if $\rho(\alpha_{x_i}) = \top$ and $\rho_1(\tau_{x_i}) = \top$ otherwise. For each clause $c_i = c_i^1 \vee c_i^2 \vee c_i^3$, at least one of $f(c_i^1)$, $f(c_i^2)$, and $f(c_i^3)$ is 1. Thus, at least one of $\rho(\alpha_{\overline{c_i^1}})$, $\rho(\alpha_{\overline{c_i^2}})$, and $\rho(\alpha_{\overline{c_i^3}})$ is $\bot$. Assume $\rho(\alpha_{\overline{c_i^j}})$ is $\bot$ for some j with $1 \leq j \leq 3$. We map $\rho'(\mu_k^{c_i}) = \bot$ for all $1 \leq k < j$ and $\rho'(\mu_k^{c_i}) = \top$ for all $j \leq k < 3$. The extension clearly satisfies the constraints C_{c_i} for all c_i. Thus ρ can be extended to a satisfying valuation for C_1.

As for C_2, ρ cannot be extended to a satisfying assignment. Notice that it requires ν_i to be mapped to $\top$ for all i. This would result in requiring mapping μ_1 to $\bot$ and μ_{n-1} to $\top$ and mapping $\mu_i = \mu_j$ for all $1 \leq i, j \leq n-1$, which is impossible. Thus all extensions of ρ do not satisfy C_2. And therefore, we have $C_1 \not\models_E C_2$.

For the other direction, assume that $C_1 \not\models_E C_2$. Then there exists a $\rho_1 \models C_1$ and there does not exist a $\rho_2 \models C_2$ with $\rho_1(\alpha) = \rho_2(\alpha)$ for all $\alpha \in E$. Since C_1 is satisfiable, this is equivalent to saying that there exists a ρ on the variables E such that ρ can be extended to a satisfying valuation for C_1 and no extensions of ρ satisfies C_2.

We construct from ρ a satisfying assignment for the boolean formula ψ. First notice that for each boolean variable x_i, ρ must map exactly one of the two type variables α_{x_i} and $\alpha_{\overline{x_i}}$ to $\bot$ and exact one to a non-$\bot$ value. To see this notice $\rho(\alpha_{x_i})$ and $\rho(\alpha_{\overline{x_i}})$ cannot both be non-$\bot$, or the constraints C_{x_i} would then require $\bot$ and $\top$ to be unified. If $\rho(\alpha_{x_i})$ and $\rho(\alpha_{\overline{x_i}})$ are both $\bot$, then ρ can be extended to a satisfying valuation for C_2. In particular, $\rho'(\nu_i) = \bot$ if both $\rho(\alpha_{x_i})$ and $\rho(\alpha_{\overline{x_i}})$ are $\bot$. For the μ_j's, we let $\rho'(\mu_j) = \bot$ if $j < i$ and $\rho(\mu_j) = \top$ if $j \geq i$. It is easy to see that $\rho' \models C_2$. Thus we have shown for each variable x_i exactly one of $\rho(\alpha_{x_i})$ and $\rho(\alpha_{\overline{x_i}})$ is $\bot$ and exactly one is a non-$\bot$ value.

Now we can show how to construct from ρ a satisfying assignment f of ψ. We let

$$\begin{cases} f(x_i) = 0 \ \wedge \ f(\overline{x_i}) = 1 \ \text{ if } \rho(\alpha_{x_i}) = \bot \\ f(x_i) = 1 \ \wedge \ f(\overline{x_i}) = 0 \ \text{ otherwise} \end{cases}$$

We show that f satisfies each clause of ψ. Let $c_i = c_i^1 \vee c_i^2 \vee c_i^3$ be a clause of ψ. Consider the constraints C_{c_i}. At least one of $\rho(\alpha_{\overline{c_i^1}})$, $\rho(\alpha_{\overline{c_i^2}})$, and $\rho(\alpha_{\overline{c_i^3}})$ must be $\bot$. W.L.O.G., assume that $\rho(\alpha_{\overline{c_i^1}})$ is $\bot$. Then $\rho(\alpha_{c_i^1})$ is non-$\bot$. Thus, $f(c_i^1)$ is 1. Therefore, f satisfies the clause c_i. Hence, f satisfies every clause of ψ, and f satisfies ψ itself.

□

To prove the NP-hardness result, observe that the described reduction is a polynomial-time reduction. Thus, the decision problem NENT is NP-hard.

□

We thus have shown that the entailment problem over extended conditional constraints is coNP-complete. The result holds even if all the constraints are restricted to be atomic.

Theorem 11. The decision problem $C_1 \models_E C_2$ over extended conditional constraints is coNP-complete.

7 Related Work

To our knowledge, we present the first results on entailment for conditional equality constraints. In this section, we survey in more detail the related work on constraint simplification and entailment mentioned in Section 1.

7.1 Complexity Issues

There are two major results on constraint simplification. Henglein and Rehof consider the problem of subtyping constraint entailment of the form $C \models \alpha \leq \beta$, where C is a constraint set with subtyping constraints and α and β are type variables [14,15]. Intuitively, the relation holds iff every solution of C also satisfies $\alpha \leq \beta$. The types are constructed from a finite lattice of base elements with the function ($\rightarrow$) and product ($\times$) constructors. They consider four cases for this problem.

- *structural subtyping over finite (simple) types*
  The entailment problem $C \models \alpha \leq \beta$ is shown to be coNP-complete [14].
- *structural subtyping over recursive types*
  The problem is shown to be PSPACE-complete [15].
- *nonstructual subtyping over finite (simple) types*
  The problem is shown to be PSPACE-hard [15].
- *nonstructual subtyping over recursive types*
  The problem is shown to be PSPACE-hard [15].

The other result is by Flanagan and Felleisen [9]. They consider the problem of simplifying a restricted class of set constraints. They considered the problem in the context of the program analysis tool for Scheme MrSpidy, a tool for inferring runtime values of variables for static debugging. They study restricted entailment $C_1 \models_E C_2$, where C_1 and C_2 are two constraint sets and E is a set of variables appearing in C_1 and C_2. They show that the restricted entailment problem is decidable (in exponential time and space) by reducing the problem to an extended version of regular tree grammar containment. They show the problem is PSPACE-hard by a polynomial time reduction from nondeterministic finite state automata containment (which is PSPACE-complete) to the set constraint entailment problem. An exact characterization of the complexity of the problem remains open.

A recent result has shown that the *atomic set constraint* entailment problem of the form $C \models \alpha \subseteq \beta$ is PSPACE-complete [23]. Atomic set constraints are a restricted class of set constraints without union and intersections and interpreted over the Herbrand universe.

A few researchers consider the semantic notions for subtyping constraint simplification. The most powerful one is the notion of *observational equivalence* defined in [36]. Intuitively it says that from the analysis point of view replacing one constraint set with an equivalent one does not change the observable behavior of the constraint system. A similar notion is used in [28] for simplifying subtyping constraints.

Aiken, Wimmers, and Palsberg [2] consider the problem of representing polymorphic types. They identify a simple algorithm and show it sound and complete for a few simple type languages. The *optimal type* of a polymorphic type is defined as a type which is equivalent and contains least number of type variables. The goal of this work, as put by the authors, is to understand why it seems difficult to get a practical system combining polymorphism and subtyping. They leave open the problem of optimal representation for polymorphically constrained types.

In constraint logic programming community, extensive research have been directed at constraint solving and entailment. Colmerauer consider similar problems for equality constraints over finite and infinite trees [4, 5]. Smolka and Treinen consider decision problems for CFT, a generalization of rational trees considered by Colmerauer. Intractability results [21] have been shown for ordering constraints over feature trees [24].

7.2 Practical Simplifications

Some researchers have considered practical simplifications of constraints.

Fahndrich and Aiken identify a few simple techniques for simplifying polymorphically constrained types and demonstrate better scalability [7]. Pottier provides a sound but incomplete algorithm for simplifying polymorphically constrained types and shows some improvement [28]. Marlow and Wadler designed and implemented a subtyping system for Erlang [19]. They give a sound approximate algorithm for deciding entailment and claim the algorithm to be complete. They have implemented a prototype system which shows some promise of being practical.

Flanagan and Felleisen identify a few practical techniques for simplifying a form of set constraints [9]. They show promising reduction in both constraint size and analysis time.

Recent work demonstrates very promising performance improvement using the idea of eliminating constraint cycles (chains of inclusion $X \subseteq Y \subseteq Z \cdots \subseteq X$) [8]. The simplification technique is demonstrated using a cubic time pointer analysis for C [3]. Adding another technique *projection merging* (to merge constructed upper bounds of a variable), the same analysis can be used to analyze `gimp` ($> 440,000$ uncommented source lines) [34]. Experiments have also shown that cycle elimination and projection merging helps dramatically with the scaling of a version of polymorphic pointer analysis based on inclusion constraints [11].

8 Conclusions and Future Work

We have given a complete characterization of the complexities of deciding entailment for conditional equality constraints and extended conditional constraints. We believe the polynomial time algorithms in the paper are of practical use. There are a few related problems to be considered:

- What happens if we allow recursive types (*i.e.*, regular trees)?
- What is the relationship with strict constructors (*i.e.*, if $c(\bot) = \bot$)?
- What is the relationship with a type system equivalent to the equality-based flow systems [25]? In this type system, the only subtype relation is given by $\bot \leq t_1 \rightarrow t_2 \leq \top$, and there is no non-trivial subtyping between function types.

We believe the same or similar techniques can be used to address the above mentioned problems, and many of the results should carry over to these problem domains.

Acknowledgments

We thank Jeff Foster and Anders Møller for their comments on an earlier version of this paper.

References

1. A. Aiken, E. Wimmers, and T.K. Lakshman. Soft Typing with Conditional Types. In *Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 163–173, January 1994.
2. A. Aiken, E. Wimmers, and J. Palsberg. Optimal Representations of Polymorphic Types with Subtyping. In *Proceedings of Third International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, pages 47–76, 1997.

3. L. O. Andersen. *Program Analysis and Specialization for the C Programming Language.* PhD thesis, DIKU, University of Copenhagen, May 1994. DIKU report 94/19.
4. A. Colmerauer. Prolog and Infinite Trees. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 231–251. Academic Press, London, 1982.
5. A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.
6. C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the Sequential Nature of Unification. *The Journal of Logic Programming*, 1:35–50, 1984.
7. M. Fähndrich and A. Aiken. Making Set-Constraint Based Program Analyses Scale. In *First Workshop on Set Constraints at CP'96*, Cambridge, MA, August 1996. Available as Technical Report CSD-TR-96-917, University of California at Berkeley.
8. M. Fähndrich, J. Foster, Z. Su, and A. Aiken. Partial Online Cycle Elimination in Inclusion Constraint Graphs. In *Proceedings of the 1998 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 85–96, Montreal, CA, June 1998.
9. C. Flanagan and M. Felleisen. Componential Set-Based Analysis. In *Proceedings of the 1997 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 235–248, June 1997.
10. C. Flanagan, M. Flatt, S. Krishnamurthi, S. Weirich, and M. Felleisen. Catching Bugs in the Web of Program Invariants. In *Proceedings of the 1996 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 23–32, May 1996.
11. J. Foster, M. Fähndrich, and A. Aiken. Monomorphic versus Polymorphic Flow-insensitive Points-to Analysis for C. In *Proceedings of the 7th International Static Analysis Symposium*, pages 175–198, 2000.
12. James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification.* Addison Wesley, 1996.
13. N. Heintze. Set Based Analysis of ML Programs. In *Proceedings of the 1994 ACM Conference on LISP and Functional Programming*, pages 306–317, June 1994.
14. F. Henglein and J. Rehof. The Complexity of Subtype Entailment for Simple Types. In *Symposium on Logic in Computer Science*, pages 352–361, 1997.
15. F. Henglein and J. Rehof. Constraint Automata and the Complexity of Recursive Subtype Entailment. In *ICALP98*, pages 616–627, 1998.
16. Fritz Henglein. Global Tagging Optimization by Type Inference. In *1992 ACM Conference on Lisp and Functional Programming*, pages 205–215, June 1992.
17. Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *The Journal of Logic Programming*, 19 & 20:503–582, May 1994.
18. N. D. Jones and S. S. Muchnick. Flow Analysis and Optimization of LISP-like Structures. In *Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 244–256, January 1979.
19. S. Marlow and P. Wadler. A Practical Subtyping System For Erlang. In *Proceedings of the International Conference on Functional Programming (ICFP '97)*, pages 136–149, June 1997.
20. R Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.
21. Martin Müller, Joachim Niehren, and Ralf Treinen. The First-Order Theory of Ordering Constraints over Feature Trees. In *Thirteenth annual IEEE Symposium on Logic in Computer Sience (LICS98)*, pages 432–443, Indianapolis, Indiana, 21–24 June 1998. IEEE Press.
22. C.G. Nelson and D.C. Oppen. Fast Decision Algorithm Based on Congruence Closure. *JACM*, 1(2):356–364, 1980.
23. J. Niehren, M. Mueller, and J. Talbot. Entailment of Atomic Set Constraints is PSPACE-Complete. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 285–294, 1999.
24. Joachim Niehren and Andreas Podelski. Feature Automata and Recognizable Sets of Feature Trees. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT 93: Theory and Practice of Software Development*, Lecture Notes in Computer Science, vol. 668, pages 356–375, Orsay, France, 13–16 April 1993. Springer-Verlag.
25. J. Palsberg. Equality-based Flow Analysis versus Recursive Types. *ACM Transactions on Programming Languages and Systems*, 20(6):1251–1264, 1998.
26. J. Palsberg and M. I. Schwartzbach. Object-Oriented Type Inference. In *Proceedings of the ACM Conference on Object-Oriented programming: Systems, Languages, and Applications*, pages 146–161, October 1991.
27. M.S. Paterson and M.N. Wegman. Linear Unification. *Journal of Computer and Systems Sciences*, 16(2):158–167, 1978.
28. F. Pottier. Simplifying Subtyping Constraints. In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming (ICFP '96)*, pages 122–133, January 1996.
29. J. C. Reynolds. *Automatic Computation of Data Set Definitions*, pages 456–461. Information Processing 68. North-Holland, 1969.
30. J.A. Robinson. Computational Logic: The Unification Computation. *Machine Intelligence*, 6:63–72, 1971.

31. O. Shivers. Control Flow Analysis in Scheme. In *Proceedings of the ACM SIGPLAN '88 Conference on Programming Language Design and Implementation*, pages 164–174, June 1988.
32. G. Smolka and R. Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, 1994.
33. B. Steensgaard. Points-to Analysis in Almost Linear Time. In *Proceedings of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 32–41, January 1996.
34. Z. Su, M. Fähndrich, and A. Aiken. Projection Merging: Reducing Redundancies in Inclusion Constraint Graphs. In *Proceedings of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 81–95, 2000.
35. R.E. Tarjan. Efficiency of a Good but Not Linear Set Union Algorithm. *JACM*, pages 215–225, 1975.
36. V. Trifonov and S. Smith. Subtyping Constrained Types. In *Proceedings of the 3rd International Static Analysis Symposium*, pages 349–365, September 1996.