Entailment with Conditional Equality Constraints[*]

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. Additionally, for comparison, we study 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, 2, 8, 11, 16, 17, 20, 22, 23, 7, 6, 27]. 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 [21]). 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 [3, 4].

In this paper, we consider the entailment problem for *conditional equality constraints*. Conditional equality constraints extend the usual equality constraints with an additional kind of constraint $\alpha \Rightarrow \tau$, which is satisfied if $\alpha = \bot$ or $\alpha = \tau$. Conditional equality constraints have been used in a number of program analyses, such as the tagging analysis of Henglein [14], the pointer analysis proposed by Steensgaard [25], and a form of equality-based flow systems for higher order functional languages [19]. We also consider entailment for a natural extension of conditional equality 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

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

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 [18]) with subtyping (as in object-oriented languages such as Java [10]), 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.

Considerable effort has been directed at constraint simplification. One body of work considers practical issues with regard to simplification of constraints [5,7,6,27,17], 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 [7,12,13]. Flanagan and Felleisen [7] consider the simplification problem for a particular form of set constraints and show that a form of entailment is PSPACE-hard. Henglein and Rehof [12,13] 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 [3, 4, 15, 24]. 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* [24]), which we introduce in Section 2. Restricted entailment arises naturally in problems that compare polymorphic constrained types (see Section 2). We show there are polynomial time algorithms for conditional equality constraints for both versions of entailment. We believe these algorithms will be of practical interest. In addition, we consider restricted entailment for a natural extension of conditional equality constraints. We show that 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.

Due to space constraints, we only provide sketches of some proofs or omit them entirely. Details may be found in the full paper [26].

2 Preliminaries

We work with simple types. 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 $\to$, and variables α ranging over a denumerable set $\mathcal{V}$ of type variables. The algorithms we present apply to type languages with other base types and type constructors. 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:

$$\rho(\bot) = \bot; \quad \rho(\top) = \top; \quad \rho(\tau_1 \to \tau_2) = \rho(\tau_1) \to \rho(\tau_2)$$

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. The set of valuations satisfying a constraint system C is the solutions of C, denoted by $S(C)$. We denote by $S(C)|_E$ the set of solutions of C restricted to a set of variables E.

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 [28]. 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:

a) $\alpha \Rightarrow \bot$	Solution: α must be $\bot$.
b) $\alpha \Rightarrow \top$	Solution: α is either $\bot$ or $\top$.
c) $\alpha \Rightarrow \beta \to \gamma$	Solution: α is either $\bot$ or a function type $\beta \to \gamma$, 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.

[1] Notice that using a linear unification algorithm such as [21] does not give a more efficient algorithm, because equality constraints are added dynamically.

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-$\perp$ 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, [25] for more information. $\square$

In later discussions, we refer to this algorithm as CondResolve. The result of running the algorithm on C is a term dag denoted by CondResolve(C) (see Definition 5). As is standard, for any term τ, we denote the equivalence class to which τ belongs by ecr(τ).

In this paper, we consider two forms of entailment: *simple entailment*: $C \models c$, and *restricted entailment*: $C_1 \models_E C_2$, 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, $C_1 \models_E C_2$ is sometimes written $C_1 \models \exists E'.C_2$, 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 [9,7]. 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 $C \models c$ if for every valuation ρ with $\rho \models C$, we have $\rho \models c$ 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 $C_1 \models_E C_2$ if for every valuation ρ_1 with $\rho_1 \models C_1$ there exists ρ_2 with $\rho_2 \models C_2$ and $\rho_1(\alpha) = \rho_2(\alpha)$ for all $\alpha \in E$.

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

Notation

– 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 $C \models c$, 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, $C \models \tau_1 = \tau_2$ if and only if $C \cup \{\alpha = \tau_1, \beta = \tau_2\} \models \alpha = \beta$, where α

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 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.

Fig. 1. Modified conditional unification algorithm.

and β do not appear in C and $\tau_1 = \tau_2$. The same also holds for when c is of the form $\alpha \Rightarrow \tau$.

Simple entailment also enjoys a distributive property, that is $C_1 \models C_2$ if and only if $C_1 \models c$ for each $c \in C_2$. Thus it suffices to only study $C \models c$. This distributive property does not hold for restricted entailment. Consider $\emptyset \models_{\{\alpha,\beta\}} \{\alpha \Rightarrow \sigma, \beta \Rightarrow \sigma\}$, 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 both the entailments $\emptyset \models_{\{\alpha,\beta\}} \{\alpha \Rightarrow \sigma\}$ and $\emptyset \models_{\{\alpha,\beta\}} \{\beta \Rightarrow \sigma\}$ 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*.

The following known result is applied extensively in the rest of the paper [3,4].

Theorem 1 (Entailment over Equality Constraints). Both simple entailment and restricted entailment over equality constraints can be decided in polynomial time.

3 Simple Entailment over Conditional Equality Constraints

In this section, we consider simple entailment over 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 1 (Transitivity of $\Rightarrow$). Any valuation ρ satisfying $\alpha \Rightarrow \beta$ and $\beta \Rightarrow \gamma$, also satisfies $\alpha \Rightarrow \gamma$.

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

1. a) Run the conditional unification algorithm in Figure 1 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 1 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.

Fig. 2. Simple entailment $C \models \alpha \Rightarrow \beta$ over conditional equality constraints.

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 1.

We now present an algorithm for deciding $C \models \alpha = \beta$ and $C \models \alpha \Rightarrow \beta$ where C is a system of conditional equality constraints. Note $C \models \alpha = \beta$ holds if and only if both $C \models \alpha \Rightarrow \beta$ and $C \models \beta \Rightarrow \alpha$ hold. We give the algorithm in Figure 2. 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. Congruence closure is required to make explicit the implied equalities between terms involving $\to$. Computing strongly connected 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.

Theorem 2. The simple entailment algorithm in Figure 2 is correct.

4 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 2.

$$\begin{tikzcd}
 & \alpha_1 \arrow[dl] \arrow[dr] & & \alpha_2 \arrow[dl] \arrow[dr] & & \alpha_3 \arrow[dl] \arrow[dr] & \\
\bot & & \sigma_1 & & \sigma_2 & & \top
\end{tikzcd}$$

Notice that the solutions of the constraints in Example 2 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 3 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 4.1
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 4.2
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 4.3
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.

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

Fig. 3. Solutions for all subsets of two constraints.

4.1 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 4a. 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 4b. 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.

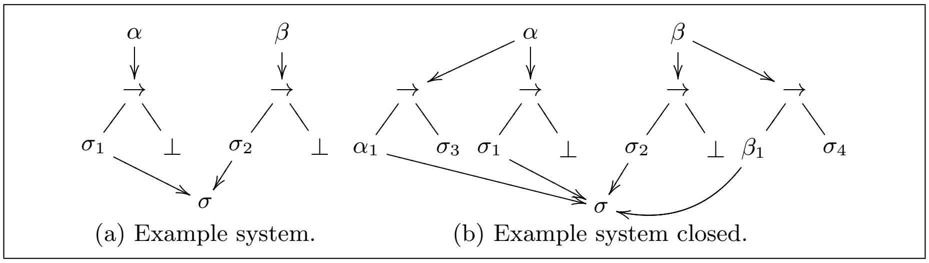

Fig. 4. An example constraint system and its closed system.

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 4a, $\text{TR}(\sigma_1, \alpha \Rightarrow \sigma_1 \to \bot)$ yields the constraint $\alpha \Rightarrow \alpha_1 \to \sigma_3$ (shown in Figure 4b).

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:

1. Let $C' = \text{CondResolve}(C)$.
2. Set W to E.
3. For each variable $\alpha \in W$, if $\alpha \Rightarrow \tau$ is in C', where $\sigma \in \text{Var}(\tau)$, and $\sigma \Rightarrow \tau' \in C'$, add $\text{TR}(\sigma, \alpha \Rightarrow \tau)$ 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 4a, 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 3. 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

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

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

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

It remains to show

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

Let ρ be a valuation in $\bigcap_{c_i, c_j \in C_\Rightarrow} S(C_= \cup \{c_i, c_j\})\ |_{E\cup A}$. 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 $\perp$. 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 $\text{ECR}(\sigma) = \bot$, then σ is forced to the same value, i.e., $\bot$, because $\sigma = \bot \in C_=$.
- If $\text{ECR}(\sigma) = \alpha$, with $\alpha \in E \cup A$, then σ is forced to $\rho(\alpha)$ in both systems, because $\sigma = \alpha \in C_=$.
- If $\text{ECR}(\sigma) = \tau_1 \to \tau_2$, one can show that ρ forces σ to the same value with an induction over the structure of $\text{ECR}(\sigma)$ (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 $\rho_1 \models C_{i,j}$, and for each extension ρ_2 of ρ with $\rho_2 \models C_{i',j'}$, 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 $\rho_3 \models C_= \cup \{\alpha \Rightarrow \tau_1, \beta \Rightarrow \tau_2\}$. 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.

□

4.2 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 2) which is important in finding a polynomial algorithm for entailment of pair constraints.

Lemma 2. 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 2, 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 4. 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
   - $C_1 \cup \{\alpha = \bot, \beta = \bot\} \models_E C_=$
   - $C_1 \cup \{\alpha = \bot, \beta = \nu_1 \to \nu_2\} \models_E C_= \cup \{\beta = \tau_2\}$
   - $C_1 \cup \{\alpha = \sigma_1 \to \sigma_2, \beta = \bot\} \models_E C_= \cup \{\alpha = \tau_1\}$
   - $C_1 \cup \{\alpha = \sigma_1 \to \sigma_2, \beta = \nu_1 \to \nu_2\} \models_E C_= \cup \{\alpha = \tau_1, \beta = \tau_2\}$

   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 2.
2. For the case $C_1 \models_E C_= \cup \{\alpha \Rightarrow \tau_1, \mu \Rightarrow \tau_2\}$, we consider two cases:

- $C_1 \cup \{\alpha = \bot\} \models_E C_= \cup \{\mu \Rightarrow \tau_2\}$;
- $C_1 \cup \{\alpha = \sigma_1 \to \sigma_2\} \models_E C_= \cup \{\alpha = \tau_1, \mu \Rightarrow \tau_2\}$

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.

- $\textsc{ecr}(\mu) = \bot$
- $\textsc{ecr}(\mu) = \tau_1 \to \tau_2$
- $\textsc{ecr}(\mu) \in E$
- $\textsc{ecr}(\mu) \notin E$

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 $C_1 \models_E C_= \cup \{\mu_1 \Rightarrow \tau_1, \mu_2 \Rightarrow \tau_2\}$, 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.

□

4.3 Reduction of Entailment to Closed Systems

We now reduce an entailment $C_1 \models_E C_2$ 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 5. Figures 5a and 5c show two constraint systems C_1 and C_2. Suppose we want to decide $C_1 \models_{\{\alpha,\beta\}} C_2$. One can verify that the entailment does hold. Figures 5b and 5d show the closed systems for C_1 and C_2, which we name C_1' and C_2'. Note that we include the $\textsc{tr}$ constraints of C_2 in C_1'. One can verify that the entailment $C_1' \models_{\{\alpha,\beta,\alpha_1,\beta_1\}} C_2'$ 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 $\alpha \Rightarrow \alpha_1 \to \sigma_3$

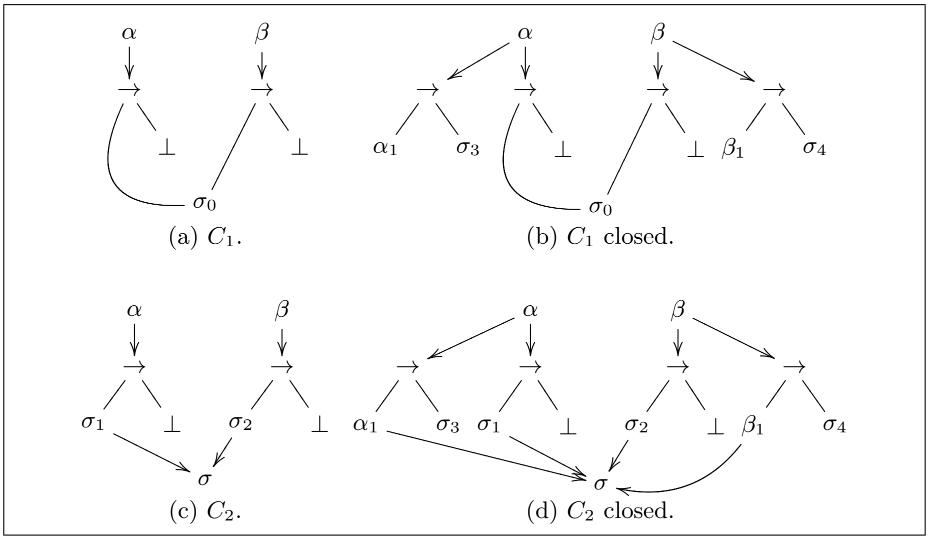

Fig. 5. Example entailment.

and $\beta \Rightarrow \beta_1 \rightarrow \sigma_4$ with $\alpha = \alpha_1 \rightarrow \sigma_3$ and $\beta = \beta_1 \rightarrow \sigma_4$ (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 4). 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 5. 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 $C_1 \models_E C_2$. Let $\rho \models C_1'$. Then $\rho \models C_1$. Thus there exists $\rho' \models C_2$ with $\rho \mid_E = \rho' \mid_E$. We extend $\rho' \mid_E$ 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 $\rho'' \mid_{E \cup A \cup A'} = \rho \mid_{E \cup A \cup A'}$.

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 $\textsc{root}(\alpha_i)$ and $\textsc{root}(\beta_j)$ (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 $\textsc{root}(\alpha_i)$ and $\textsc{root}(\beta_j)$ 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 $C_1 \models_E C_2$. 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 $C_1' \models_{E \cup A \cup A'} C_2'$.

□

4.4 Putting Everything Together

Theorem 6. Restricted entailment for conditional constraints can be decided in polynomial time.

Proof. Consider the problem $C_1 \models_E C_2$. By Theorem 5, it is equivalent to testing $C_1' \models_{E \cup A \cup A'} C_2'$ (see Theorem 5 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 3, 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 4, 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.

□

5 Extended Conditional Constraints

In this section, we show that restricted entailment for a natural extension of the standard conditional constraint language is coNP-complete. [2] This section

[2] Simple entailment for this extension is in P. See [26] for details.

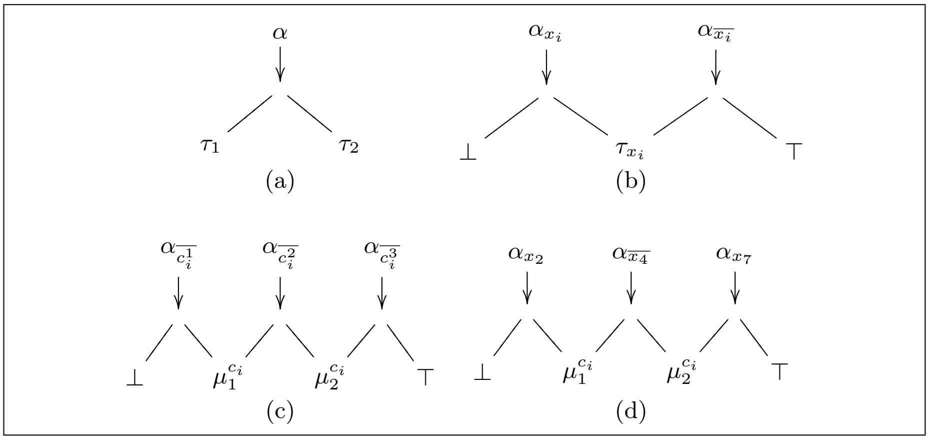

Fig. 6. Graph representations of constraints.

is helpful for a comparison between this constraint language with the standard conditional constraint language, which we consider in Section 3 and Section 4. The results in this section provide one natural boundary between tractable and intractable entailment problems.

We extend the constraint language with a new construct $\alpha \Rightarrow (\tau_1 = \tau_2)$, 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 [19].

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 restricted entailment for this extended language.

5.1 Restricted Entailment

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

We define the decision problem NENT as the problem of deciding whether $C_1 \not\models_E C_2$, 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 7. The decision problem NENT for extended conditional constraints 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 [3] is coNP-hard.

Theorem 8. 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 $\perp$ is treated as the boolean value false and any non-$\perp$ 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 shown in Figure 6a.

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 shown in Figure 6b, 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 6c, 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 shown in Figure 6d.

We let C_1 be the union of all the constraints C_{x_i} and C_{c_j} for $1 \leq i \leq n$ and $1 \leq j \leq m$, *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 7. 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 7, allowing $\mu_1, \ldots, \mu_{i-1}$ to be set to $\perp$ and $\mu_i, \ldots, \mu_{n-1}$ to be set to $\top$.

[3] Only variables, $\perp$, and $\top$ are in the constraint system.

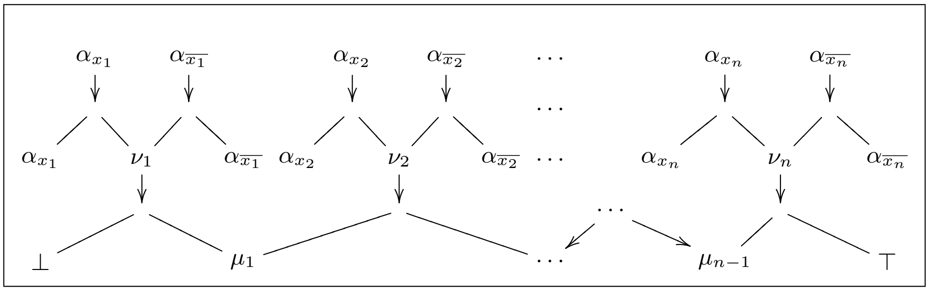

Fig. 7. Constructed constraint system C_2.

We let the set of interface variables $E = \{\alpha_{x_i}, \alpha_{\overline{x_i}} \mid 1 \leq i \leq n\}$. 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.

$\square$

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

6 Conclusions and Future Work

We have given a complete characterization of the complexities of deciding entailment for conditional equality constraints over finite types (finite trees). 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 $c(\perp) = \perp$)?
- What is the relationship with a type system equivalent to the equality-based flow systems [19]? In this type system, the only subtype relation is given by $\perp \leq t_1 \to t_2 \leq \top$, 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, Anders Møller, and the anonymous referees 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. L. O. Andersen. *Program Analysis and Specialization for the C Programming Language*. PhD thesis, DIKU, University of Copenhagen, May 1994. DIKU report 94/19.
3. A. Colmerauer. Prolog and Infinite Trees. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 231–251. Academic Press, London, 1982.
4. A. Colmerauer. Equations and Inequations on Finite and Infinite Trees. In *2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison Wesley, 1996.
11. 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.
12. F. Henglein and J. Rehof. The Complexity of Subtype Entailment for Simple Types. In *Symposium on Logic in Computer Science*, pages 352–361, 1997.
13. F. Henglein and J. Rehof. Constraint Automata and the Complexity of Recursive Subtype Entailment. In *ICALP98*, pages 616–627, 1998.
14. Fritz Henglein. Global Tagging Optimization by Type Inference. In *1992 ACM Conference on Lisp and Functional Programming*, pages 205–215, June 1992.
15. Joxan Jaffar and Michael J. Maher. Constraint Logic Programming: A Survey. *The Journal of Logic Programming*, 19 & 20:503–582, May 1994.
16. 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.
17. 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.

18. R Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17(3):348–375, December 1978.
19. J. Palsberg. Equality-based Flow Analysis versus Recursive Types. *ACM Transactions on Programming Languages and Systems*, 20(6):1251–1264, 1998.
20. 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.
21. M.S. Paterson and M.N. Wegman. Linear Unification. *Journal of Computer and Systems Sciences*, 16(2):158–167, 1978.
22. J. C. Reynolds. *Automatic Computation of Data Set Definitions*, pages 456–461. Information Processing 68. North-Holland, 1969.
23. 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.
24. G. Smolka and R. Treinen. Records for Logic Programming. *Journal of Logic Programming*, 18(3):229–258, 1994.
25. 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.
26. Z. Su and A. Aiken. Entailment with Conditional Equality Constraints. Technical Report UCB//CSD-00-1113, University of California, Berkeley, October 2000.
27. 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.
28. R.E. Tarjan. Efficiency of a Good but Not Linear Set Union Algorithm. *JACM*, pages 215–225, 1975.