Inference of User-Defined Type Qualifiers and Qualifier Rules

Brian Chin Shane Markstrum Todd Millstein Jens Palsberg

UCLA Computer Science Department
Technical Report CSD-TR-050041
October 2005

This technical report is a companion to our ESOP 2006 paper of the same name [1]. The first section provides details on our approach to qualifier inference, and the second section provides details on our approach to rule inference. This technical report focuses on the formal details that are missing from the ESOP paper for space reasons; refer to that paper for motivation, examples, etc.

1 Qualifier Inference

The original CLARITY system supports qualifier *checking*: all variables must be explicitly annotated with their qualifiers. In this section, we show how to support qualifier *inference* in the presence of user-defined qualifier rules. We formalize qualifier inference for a simply-typed lambda calculus with references and user-defined qualifiers, as defined by the following grammar:

$$\begin{array}{lcl} e & ::= & c \mid e_1 + e_2 \mid x \mid \lambda x : \tau . e \mid e_1\ e_2 \mid \texttt{ref}\ e \mid e_1 := e_2 \mid !e \mid \texttt{assert}(e, q) \\ \tau & ::= & \texttt{int} \mid \tau_1 \rightarrow \tau_2 \mid \texttt{ref}\ \tau \end{array}$$

Let Q be the set $\{q_1, \ldots, q_n\}$ of user-defined qualifiers in a program. Sets of qualifiers from Q form a natural lattice, with partial order $\supseteq$, least-upper-bound function $\cap$, and greatest-lower-bound function $\cup$. We denote elements of this lattice by metavariable l; qualified types are ranged over by metavariable ρ and are defined as follows:

$$\rho \ ::= \ l\ \phi \qquad \phi ::= \texttt{int} \mid \rho_1 \rightarrow \rho_2 \mid \texttt{ref}\ \rho$$

We present both a type system and a constraint system for qualifier inference and prove their equivalence, and we describe an algorithm for solving the generated constraints. We assume the bound variables in expressions are annotated with unqualified types τ. It is possible to combine qualifier inference with type inference, but separating them simplifies the presentation.

1.1 Formal Qualifier Rules

Figure 1 shows a sample user-defined type qualifier for nonzero integers in CLARITY. We formalize the `case` rules as defining two kinds of relations. First, some `case` clauses have the effect of declaring a specificity relation between qualifiers. We formalize these rules as defining axioms for a relation of the form $q_1 \triangleright q_2$. For example, the second `case` clause in Figure 1 would be represented by the axiom $\texttt{pos} \triangleright \texttt{nonzero}$. We use $\triangleright^*$ to denote the reflexive, transitive closure of the user-defined $\triangleright$ relation, and we require $\triangleright^*$ to be a partial order.

```
qualifier nonzero(int Expr E)
  case E of
      decl int Const C:
        C, where C != 0
    | decl int Expr E1:
        E1, where pos(E1)
    | decl int Expr E1, E2:
        E1 * E2,
        where nonzero(E1) && nonzero(E2)
  restrict
      decl int Expr E1, E2:
        E1 / E2, where nonzero(E2)
  invariant value(E) != 0
```

Figure 1: A user-defined type qualifiers for nonzero integers.

The other kind of `case` clause uses a pattern to match on a constructor (e.g., +), and the clause determines the qualifier of the entire expression based on the qualifiers of the immediate subexpressions. We formalize these rules as defining relations of the form R_p^q, where q is a qualifier and p represents one of the constructors in our formal language, ranging over integer constants and the symbols `+`, λ, and `ref`. The arity of each relation R_p^q is the number of immediate subexpressions of the constructor represented by p, and the domain of each argument to the relation is Q. Each `case` clause is formalized through axioms for these relations. For example, the third `case` clause in Figure 1 would be represented by the axiom $R_*^{\texttt{nonzero}}(\texttt{nonzero}, \texttt{nonzero})$ (if our formal language contained multiplication). The first `case` clause in that figure would be formalized through the (conceptually infinite) set of axioms $R_1^{\texttt{nonzero}}()$, $R_2^{\texttt{nonzero}}()$, etc. For simplicity of presentation, we assume that each subexpression is required to satisfy only a single qualifier. In fact, our implementation allows each subexpression to be constrained to satisfy a set of qualifiers, and it would be straightforward to update our formalism to support this ability.

Finally, we formalize the `restrict` rules with an expression of the form $\texttt{assert}(e, q)$, which requires the type system to ensure that the top-level qualifier on expression e's type includes qualifier q. For example, the `restrict` rule in Figure 1 is modeled by replacing each denominator expression e in a program with $\texttt{assert}(e, \texttt{nonzero})$. The `assert` expression can also be used to model explicit qualifier annotations in programs.

1.2 The Type System

1.3 A Type System for Qualifier Inference

We assume we are given an expression e along with a set A of axioms representing the user-defined qualifier rules, as described above. The qualifier type system is presented in Figure 3, and the axioms in A are implicitly considered to augment this formal system. As usual, metavariable Γ ranges over type environments, which map variables to qualified types. The rule for $\texttt{assert}(e, q)$ infers a qualified type for e and then checks that q is in the top-level qualifier of this type. The *strip* function used in the rule for lambdas removes all qualifiers from a qualified type ρ, producing

$\boxed{\rho \leq \rho'}$

$$\frac{l_1 \supseteq l_2}{l_1 \texttt{int} \leq l_2 \texttt{int}}\ \text{S-Int} \qquad \frac{l_1 \supseteq l_2 \qquad \rho \leq \rho' \qquad \rho' \leq \rho}{l_1\ \texttt{ref}\ \rho \leq l_2\ \texttt{ref}\ \rho'}\ \text{S-Ref} \qquad \frac{l_1 \supseteq l_2 \qquad \rho_2 \leq \rho_1 \qquad \rho_1' \leq \rho_2'}{l_1(\rho_1 \to \rho_1') \leq l_2(\rho_2 \to \rho_2')}\ \text{S-Fun}$$

Figure 2: Subtyping rules.

$\boxed{\Gamma \vdash e : \rho}$

$$\frac{l = \{q \mid R_c^{q'}() \wedge q' \rhd^* q\}}{\Gamma \vdash c : l\ \texttt{int}}\ \text{T-Int} \qquad \frac{\Gamma \vdash e_1 : l_1\ \texttt{int} \qquad \Gamma \vdash e_2 : l_2\ \texttt{int} \qquad l = \{q \mid R_+^{q'}(q_1, q_2) \wedge q_1 \in l_1 \wedge q_2 \in l_2 \wedge q' \rhd^* q\}}{\Gamma \vdash e_1 + e_2 : l\ \texttt{int}}\ \text{T-Plus} \qquad \frac{\Gamma(x) = \rho}{\Gamma \vdash x : \rho}\ \text{T-Var}$$

$$\frac{\mathit{strip}(\rho_1) = \tau_1 \qquad \Gamma, x : \rho_1 \vdash e : \rho_2 \qquad \rho_2 = l_2\ \phi_2 \qquad l = \{q \mid R_\lambda^{q'}(q_2) \wedge q_2 \in l_2 \wedge q' \rhd^* q\}}{\Gamma \vdash \lambda x : \tau_1.e : l(\rho_1 \to \rho_2)}\ \text{T-Fun} \qquad \frac{\Gamma \vdash e_1 : l(\rho_2 \to \rho) \qquad \Gamma \vdash e_2 : \rho_2}{\Gamma \vdash e_1\ e_2 : \rho}\ \text{T-App}$$

$$\frac{\Gamma \vdash e : \rho \qquad \rho = l_0\ \phi_0 \qquad l = \{q \mid R_{\texttt{ref}}^{q'}(q_0) \wedge q_0 \in l_0 \wedge q' \rhd^* q\}}{\Gamma \vdash \texttt{ref}\ e : l\ \texttt{ref}\ \rho}\ \text{T-Ref} \qquad \frac{\Gamma \vdash e_1 : l\ \texttt{ref}\ \rho \qquad \Gamma \vdash e_2 : \rho}{\Gamma \vdash e_1 := e_2 : \rho}\ \text{T-Assgn}$$

$$\frac{\Gamma \vdash e : l\ \texttt{ref}\ \rho}{\Gamma \vdash !e : \rho}\ \text{T-Deref} \qquad \frac{\Gamma \vdash e : \rho \qquad \rho = l\ \phi \qquad q \in l}{\Gamma \vdash \texttt{assert}(e, q) : \rho}\ \text{T-Assert} \qquad \frac{\Gamma \vdash e : \rho' \qquad \rho' \leq \rho}{\Gamma \vdash e : \rho}\ \text{T-Sub}$$

Figure 3: Qualifier inference rules.

an unqualified type τ, and is defined as follows:

$$\begin{array}{rcl} \mathit{strip}(l\ \texttt{int}) & = & \texttt{int} \\ \mathit{strip}(l\ \texttt{ref}\ \rho) & = & \texttt{ref}\ \mathit{strip}(\rho) \\ \mathit{strip}(l\ (\rho_1 \to \rho_2)) & = & \mathit{strip}(\rho_1) \to \mathit{strip}(\rho_2) \end{array}$$

The main novelty in the type system is the consultation of the axioms in A to produce the top-level qualifiers for constructor expressions. For example, consider the first rule in Figure 3, which infers the qualifiers for an integer constant c using a set comprehension notation. The resulting set l includes all qualifiers q' such that the $R_c^{q'}()$ relation holds (according to the axioms in A), as well as all qualifiers q that are "less specific" than such a q' as defined by the $\triangleright^*$ relation. In this way, the rule finds all possible qualifiers that can be proven to hold given the user-defined `case` clauses. The subsumption rule at the end of the figure can then be used to forget some of these qualifiers, via the subtyping rules in Figure 2. The inference of top-level qualifiers is similar for the other constructors, except that consultation of the R relation makes use of the top-level qualifiers inferred for the immediate subexpressions.

1.4 The Constraint System

In this subsection we describe a constraint-based algorithm for qualifier inference.

1.4.1 Constraints and Solutions

The key novelty in our constraint system is the use of a specialized form of *conditional constraints* to represent the effects of user-defined qualifier rules. The metavariable α represents *qualifier variables*, and we generate constraints according to the following syntax:

$$\mathcal{C} ::= \alpha \supseteq \alpha \mid q \in \alpha \mid q \in \alpha \Rightarrow \bigvee(\bigwedge q \in \alpha)$$

Given a set C of constraints, let S be a mapping from the qualifier variables in C to sets of qualifiers. We say that S is a *solution* to C if S satisfies all constraints in C. Given two solutions S and S' to C, we say that $S \preceq S'$ if for all qualifier variables α in the domain of S and S', we have $S(\alpha) \supseteq S'(\alpha)$. Then a solution S to C is a *least solution* to C if for all other solutions S' to C, it is the case that $S \preceq S'$. The following theorem shows that unique least solutions always exist.

Theorem 1.1. If a set C of constraints of the form $\mathcal{C}$ has a solution, then it has a least solution.
Proof By Lemma 1.1, C has a least solution, and by Lemma 1.2 this least solution is unique. □

Lemma 1.1. If a set C of constraints of the form $\mathcal{C}$ has a solution, then it has a least solution.
Proof Let $S_1, \ldots, S_n$ be the set of solutions to C. We build a mapping S_0 as follows: for each qualifier variable α mentioned in C, we define $S_0(\alpha) = \bigcup_{1 \le i \le n} S_i(\alpha)$. Then by definition of $\preceq$ we have $S_0 \preceq S_i$ for each $1 \le i \le n$. Then S_0 is a least solution if we can show that it is indeed a solution. Therefore, we must show that S_0 satisfies each constraint in C. We do a case analysis on the form of this constraint:

- Case $\alpha_1 \supseteq \alpha_2$. Since each S_i is a solution, we have that $S_i(\alpha_1) \supseteq S_i(\alpha_2)$, for each $1 \le i \le n$. Then $\bigcup_{1 \le i \le n} S_i(\alpha_1) \supseteq \bigcup_{1 \le i \le n} S_i(\alpha_2)$, so S_0 satisfies the constraint.
- Case $q \in \alpha$. Since each S_i is a solution, we have that $q \in S_i(\alpha)$, for each $1 \le i \le n$. Then $q \in \bigcup_{1 \le i \le n} S_i(\alpha)$, so S_0 satisfies the constraint.

- Case $q \in \alpha \Rightarrow \bigvee_{1 \le j \le m}(\bigwedge_{1 \le k \le p_j} q_{jk} \in \alpha_{jk})$. Since each S_i is a solution, we have that each S_i satisfies this constraint. We consider two subcases:
  - For all $1 \le i \le n$, we have $q \notin S_i(\alpha)$. Then also $q \notin \bigcup_{1 \le i \le n} S_i(\alpha)$, so $q \notin S_0(\alpha)$. Therefore, S_0 satisfies the constraint vacuously.
  - There exists some $1 \le i \le n$ such that $q \in S_i(\alpha)$. Since S_i is a solution, it satisfies the constraint. Therefore, since S_i satisfies the left-hand side of the constraint, it must also satisfy the right-hand side. Therefore there exists some $1 \le j \le m$ such that for all $1 \le k \le p_j$ we have $q_{jk} \in S_i(\alpha_{jk})$. Then also $q_{jk} \in \bigcup_{1 \le i \le n} S_i(\alpha_{jk})$. Therefore we have that there exists some $1 \le j \le m$ such that for all $1 \le k \le p_j$ we have $q_{jk} \in S_0(\alpha_{jk})$. Therefore S_0 satisfies the right-hand side of the constraint, and hence it satisfies the whole constraint.

□

Lemma 1.2. A set C of constraints of the form $\mathcal{C}$ has at most one unique solution.
Proof Suppose not, so C has two distinct least solutions S and S'. Then we have both $S \preceq S'$ and vice versa, so by the definition of $\preceq$ we have that for all qualifier variables α in the domain of S and S', we have $S(\alpha) \supseteq S'(\alpha)$ and $S'(\alpha) \supseteq S(\alpha)$. Then $S = S'$, so we have a contradiction. □

1.4.2 Constraint Generation

We formalize constraint generation by a judgment of the form $\kappa \vdash e : \delta \mid C$. Here C is a set of constraints in the form $\mathcal{C}$, and the metavariable δ represents qualified types whose qualifiers are all qualifier variables:

$$\delta \ ::= \ \alpha\ \varphi \qquad \varphi ::= \texttt{int} \mid \delta_1 \to \delta_2 \mid \texttt{ref}\ \delta$$

The metavariable κ denotes type environments that map program variables to qualified types of the form δ.

The inference rules defining this judgment are shown in Figure 5. The *embed* function adds fresh qualifier variables to an unqualified type τ in order to turn it into a qualified type δ; it is defined as follows:

$$\begin{array}{rcll} \mathit{embed}(\texttt{int}) & = & \alpha\ \texttt{int} & \alpha \text{ fresh} \\ \mathit{embed}(\texttt{ref}\ \tau) & = & \alpha\ \texttt{ref}\ \mathit{embed}(\tau) & \alpha \text{ fresh} \\ \mathit{embed}(\tau_1 \to \tau_2) & = & \alpha(\mathit{embed}(\tau_1) \to \mathit{embed}(\tau_2)) & \alpha \text{ fresh} \end{array}$$

We also adapt the *strip* function described earlier to work over qualified types of the form δ:

$$\begin{array}{rcl} \mathit{strip}(\alpha\ \texttt{int}) & = & \texttt{int} \\ \mathit{strip}(\alpha\ \texttt{ref}\ \rho) & = & \texttt{ref}\ \mathit{strip}(\rho) \\ \mathit{strip}(\alpha\ (\rho_1 \to \rho_2)) & = & \mathit{strip}(\rho_1) \to \mathit{strip}(\rho_2) \end{array}$$

Finally, we define a function `refresh` for generating fresh qualifier variables for a qualified type:

$$\begin{array}{rcl} \mathit{refresh}(\rho) & = & \mathit{embed}(\mathit{strip}(\rho)) \\ \mathit{refresh}(\delta) & = & \mathit{embed}(\mathit{strip}(\delta)) \end{array}$$

To keep the constraint generation purely syntax-directed, subsumption is "built in" to each rule: the *refresh* function is used to create a fresh qualified type δ, which is constrained by a subtype constraint of the form $\delta' \sqsubseteq \delta$. Subtype constraints are also generated for applications and

$$
\begin{array}{rcl}
\alpha_1 \texttt{int} \sqsubseteq \alpha_2 \texttt{int} & \equiv & \{\alpha_1 \supseteq \alpha_2\} \\
\alpha_1 \texttt{ref}\ \delta_1 \sqsubseteq \alpha_2 \texttt{ref}\ \delta_2 & \equiv & \{\alpha_1 \supseteq \alpha_2\} \cup \delta_1 \sqsubseteq \delta_2 \cup \delta_2 \sqsubseteq \delta_1 \\
\alpha_1(\delta_1 \to \delta_1') \sqsubseteq \alpha_2(\delta_2 \to \delta_2') & \equiv & \{\alpha_1 \supseteq \alpha_2\} \cup \delta_2 \sqsubseteq \delta_1 \cup \delta_1' \sqsubseteq \delta_2'
\end{array}
$$

Figure 4: Converting type constraints into set constraints.

$$
\frac{\alpha' \text{ fresh} \qquad \delta' = \alpha'\ \texttt{int} \qquad \delta = \mathit{refresh}(\delta')}{\kappa \vdash c : \delta \mid \delta' \sqsubseteq \delta \cup \{C_c^q(\alpha') \mid q \in Q\}}\ \text{C-Int}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e_1 : \alpha_1\ \texttt{int} \mid C_1 \qquad \kappa \vdash e_2 : \alpha_2\ \texttt{int} \mid C_2 \\ \alpha' \text{ fresh} \qquad \delta' = \alpha'\ \texttt{int} \qquad \delta = \mathit{refresh}(\delta')\end{array}}{\begin{array}{l}\kappa \vdash e_1 + e_2 : \delta \mid \\ C_1 \cup C_2 \cup \delta' \sqsubseteq \delta \cup \{C_+^q(\alpha_1, \alpha_2, \alpha') \mid q \in Q\}\end{array}}\ \text{C-Plus}
$$

$$
\frac{\kappa(x) = \delta' \qquad \delta = \mathit{refresh}(\delta')}{\kappa \vdash x : \delta \mid \delta' \sqsubseteq \delta}\ \text{C-Var}
$$

$$
\frac{\begin{array}{c}\kappa, x : \delta_1 \vdash e : \delta_2 \mid C \qquad \delta_1 = \mathit{embed}(\tau_1) \qquad \delta_2 = \alpha_2\ \varphi_2 \\ \alpha' \text{ fresh} \qquad \delta' = \alpha'(\delta_1 \to \delta_2) \qquad \delta = \mathit{refresh}(\delta')\end{array}}{\kappa \vdash \lambda x : \tau_1.e : \delta \mid C \cup \delta' \sqsubseteq \delta \cup \{C_\lambda^q(\alpha_2, \alpha') \mid q \in Q\}}\ \text{C-Fun}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e_1 : \alpha(\delta_2 \to \delta') \mid C_1 \qquad \kappa \vdash e_2 : \delta_2' \mid C_2 \\ \delta = \mathit{refresh}(\delta')\end{array}}{\begin{array}{l}\kappa \vdash e_1\ e_2 : \delta \mid \\ C_1 \cup C_2 \cup \delta_2' \sqsubseteq \delta_2 \cup \delta' \sqsubseteq \delta\end{array}}\ \text{C-App}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e : \delta_0 \mid C \qquad \delta_0 = \alpha_0\ \varphi_0 \\ \alpha' \text{ fresh} \qquad \delta' = \alpha'\ \texttt{ref}\ \delta_0 \qquad \delta = \mathit{refresh}(\delta')\end{array}}{\kappa \vdash \texttt{ref}\ e : \delta \mid C \cup \delta' \sqsubseteq \delta \cup \{C_{\texttt{ref}}^q(\alpha_0, \alpha') \mid q \in Q\}}\ \text{C-Ref}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e_1 : \alpha\ \texttt{ref}\ \delta' \mid C_1 \qquad \kappa \vdash e_2 : \delta'' \mid C_2 \\ \delta = \mathit{refresh}(\delta')\end{array}}{\begin{array}{l}\kappa \vdash e_1 := e_2 : \delta \mid \\ C_1 \cup C_2 \cup \delta'' \sqsubseteq \delta' \cup \delta' \sqsubseteq \delta\end{array}}\ \text{C-Assgn}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e : \alpha\ \texttt{ref}\ \delta' \mid C \\ \delta = \mathit{refresh}(\delta')\end{array}}{\kappa \vdash !e : \delta \mid C \cup \delta' \sqsubseteq \delta}\ \text{C-Deref}
$$

$$
\frac{\begin{array}{c}\kappa \vdash e : \delta' \mid C \qquad \delta' = \alpha\ \varphi \\ \delta = \mathit{refresh}(\delta')\end{array}}{\begin{array}{l}\kappa \vdash \texttt{assert}(e, q) : \delta \mid \\ C \cup \{q \in \alpha\} \cup \delta' \sqsubseteq \delta\end{array}}\ \text{C-Assert}
$$

Figure 5: Constraint generation rules for qualifier inference.

assignments, as usual. We treat a subtype constraint as a shorthand for a set of qualifier-variable constraints, as shown in Figure 4.

Each rule for an expression with top-level constructor p produces one conditional constraint per qualifier q in Q, denoted C_p^q. Informally, the constraint C_p^q *inverts* the user-defined qualifier rules, indicating all the possible ways to prove that an expression with constructor p can be given qualifier q according to the axioms in A. For example, both the second and third `case` clauses in Figure 1 can be used to prove that a product `a*b` has the qualifier `nonzero`, so our implementation of constraint generation in Clarity produces the following conditional constraint:

$$
\texttt{nonzero} \in \alpha_{\texttt{a*b}} \Rightarrow ((\texttt{nonzero} \in \alpha_{\texttt{a}} \wedge \texttt{nonzero} \in \alpha_{\texttt{b}}) \vee (\texttt{pos} \in \alpha_{\texttt{a*b}}))
$$

More formally, let $\mathit{zip}(R_p^q(q_1, \ldots, q_m), \alpha_1, \ldots, \alpha_m)$ denote the constraint $q_1 \in \alpha_1 \wedge \ldots \wedge q_m \in \alpha_m$. Let $\{a_1, \ldots, a_u\}$ be all the axioms in A for the relation R_p^q, and let $\{q_1, \ldots, q_v\} = \{q' \in Q \mid q' \triangleright q\}$.

Then $C_p^q(\alpha_1, \ldots, \alpha_m, \alpha')$ is the following conditional constraint[1]:

$$q \in \alpha' \Rightarrow (\bigvee_{1 \le i \le u} zip(a_i, \alpha_1, \ldots, \alpha_m) \vee \bigvee_{1 \le i \le v} q_i \in \alpha')$$

1.5 Equivalence of Qualifier Checking and Qualifier Inference

We lift a solution S to a set of constraints to map qualified types δ to qualified types ρ:

$$\begin{array}{rcl} S(\alpha\ \texttt{int}) & = & S(\alpha)\ \texttt{int} \\ S(\alpha\ \texttt{ref}\ \delta) & = & S(\alpha)\ \texttt{ref}\ S(\delta) \\ S(\alpha(\delta_1 \to \delta_2)) & = & S(\alpha)(S(\delta_1) \to S(\delta_2)) \end{array}$$

Our equivalence result is then expressed as follows:

Corollary 1.1. $\emptyset \vdash e : \rho$ if and only if $\emptyset \vdash e : \delta \mid C$ and there exists a solution S to C such that $S(\delta) = \rho$.
Proof Follows from the Theorems 1.2 and 1.3 below. □

We lift *refresh* to type environments Γ, producing a type environment κ, as follows:
$\mathit{refresh}(x_1 : \rho_1, \ldots, x_n : \rho_n) = x_1 : \mathit{refresh}(\rho_1), \ldots, x_n : \mathit{refresh}(\rho_n)$

Theorem 1.2. If $\Gamma \vdash e : \rho$ and $\kappa = \mathit{refresh}(\Gamma)$, then there exist δ and C such that $\kappa \vdash e : \delta \mid C$, and there exists a solution S to C such that $S(\delta) = \rho$.
Proof We prove the theorem by induction on the derivation of $\Gamma \vdash e : \rho$.

- Case T-Int: Then $e = c$ and $\rho = l\ \texttt{int}$ and $l = \{q \mid R_c^{q'}() \wedge q' \rhd^* q\}$. By C-Int we have $\kappa \vdash e : \delta \mid C$, where α' fresh and $\delta' = \alpha'\ \texttt{int}$ and $\delta = \mathit{refresh}(\delta')$ and $C = \delta' \sqsubseteq \delta \cup \{C_c^q(\alpha') \mid q \in Q\}$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let $S_0 = \{\alpha' \mapsto l\}$, and let $S = S_0 \cup (S_0 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. By definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$. Therefore this case is proven if we can show that S satisfies $C_c^q(\alpha')$ for each $q \in Q$. There are several cases:
  - $R_c^q() \in A$: Then one disjunct of the right-hand side of $C_c^q(\alpha')$ is `true`, so S satisfies $C_c^q(\alpha')$.
  - $q \in l$ but $R_c^q() \notin A$: By definition of l there is some q' such that $R_c^{q'}()$ and $q' \rhd^* q$. Since $R_c^q() \notin A$, we know that $q' \neq q$. Then by definition of $\rhd^*$ there must exist some $q'' \neq q$ such that $q' \rhd^* q''$ and $q'' \rhd q$. Therefore by definition of l we have $q'' \in l$. By definition of $C_c^q(\alpha')$, one disjunct of the right-hand side is $q'' \in \alpha'$. Since $q'' \in l$, S satisfies this disjunct and hence S satisfies the entire constraint.
  - $q \notin l$. Then the left-hand side of $C_c^q(\alpha')$ is falsified by S, so the entire constraint is vacuously satisfied by S.
- Case T-Plus: Then $e = e_1 + e_2$ and $\rho = l\ \texttt{int}$ and $\Gamma \vdash e_1 : l_1\ \texttt{int}$ and $\Gamma \vdash e_2 : l_2\ \texttt{int}$ and $l = \{q \mid R_+^{q'}(q_1, q_2) \wedge q_1 \in l_1 \wedge q_2 \in l_2 \wedge q' \rhd^* q\}$. By induction, we have $\kappa \vdash e_1 : \delta_1 \mid C_1$ and a solution S_1 to C_1 such that $S_1(\delta_1) = l_1\ \texttt{int}$. Similarly by induction we have $\kappa \vdash e_2 : \delta_2 \mid C_2$ and a solution S_2 to C_2 such that $S_2(\delta_2) = l_2\ \texttt{int}$. Therefore δ_1 has the form $\alpha_1\ \texttt{int}$ and δ_2 has the form $\alpha_2\ \texttt{int}$. Then by C-Plus we have $\kappa \vdash e_1 + e_2 : \delta \mid C$, where α' fresh and

[1]The empty conjunction is equivalent to `true`; the empty disjunction is equivalent to `false`.

$\delta' = \alpha'$ `int` and $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta' \sqsubseteq \delta \cup \{C^q_+(\alpha_1, \alpha_2, \alpha') \mid q \in Q\}$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let $S_0 = \{\alpha' \mapsto l\}$, and let $S = S_1 \cup S_2 \cup S_0 \cup (S_0 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_1 and S_2 are respectively solutions to C_1 and C_2, S is also a solution to C_1 and C_2. By definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$. Therefore this case is proven if we can show that S satisfies $C^q_+(\alpha_1, \alpha_2, \alpha')$ for each $q \in Q$. There are several cases:

- $R^q_+(q_1, q_2) \in A$, where $q_1 \in l_1$ and $q_2 \in l_2$: Then one disjunct of the right-hand side of $C^q_+(\alpha_1, \alpha_2, \alpha')$ is $q_1 \in \alpha_1 \wedge q_2 \in \alpha_2$. Since $q_1 \in l_1$ and $q_2 \in l_2$, this disjunct is satisfied by S, so S satisfies $C^q_+(\alpha_1, \alpha_2, \alpha')$.
- $q \in l$ but there is no $q_1 \in l_1$ and $q_2 \in l_2$ such that $R^q_+(q_1, q_2) \in A$: By definition of l there is some q' such that $R^{q'}_+(q'_1, q'_2)$ and $q' \triangleright^* q$, where $q'_1 \in l_1$ and $q'_2 \in l_2$. Therefore $q' \neq q$. Then by definition of $\triangleright^*$ there must exist some $q'' \neq q$ such that $q' \triangleright^* q''$ and $q'' \triangleright q$. Therefore by definition of l we have $q'' \in l$. By definition of $C^q_+(\alpha_1, \alpha_2, \alpha')$, one disjunct of the right-hand side is $q'' \in \alpha'$. Since $q'' \in l$, S satisfies this disjunct and hence S satisfies the entire constraint.
- $q \notin l$. Then the left-hand side of $C^q_+(\alpha_1, \alpha_2, \alpha')$ is falsified by S, so the entire constraint is vacuously satisfied by S.

- Case T-Var: Then $e = x$ and $\Gamma(x) = \rho$. Since $\kappa = \mathit{refresh}(\Gamma)$, there is some δ' such that of $\kappa(x) = \delta'$. Therefore by C-Var we have $\kappa \vdash x : \delta \mid \delta' \sqsubseteq \delta$ and $\delta = \mathit{refresh}(\delta')$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$, and there is a mapping S_0 such that $S_0(\delta') = \rho$. Let $S = S_0 \cup (S_0 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to $\delta' \sqsubseteq \delta$. By definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$.

- Case T-Fun: Then $e = \lambda x : \tau_1.e_2$ and $\rho = l(\rho_1 \to \rho_2)$ and $\Gamma, x : \rho_1 \vdash e_2 : \rho_2$ and $\mathit{strip}(\rho_1) = \tau_1$ and $\rho_2 = l_2\ \phi_2$ and $l = \{q \mid R^{q'}_\lambda(q_2) \wedge q_2 \in l_2 \wedge q' \triangleright^* q\}$. Let $\delta_1 = \mathit{embed}(\tau_1)$. Then $\delta_1 = \mathit{refresh}(\rho_1)$. Therefore by induction, we have $\kappa, x : \delta_1 \vdash e_2 : \delta_2 \mid C_2$ and there exists a solution S_2 to C_2 such that $S_2(\delta_2) = \rho_2$. Let $\delta_2 = \alpha_2\ \varphi_2$. Then by C-Fun we have $\kappa \vdash \lambda x : \tau_1.e_2 : \delta \mid C$, where α' fresh and $\delta' = \alpha'(\delta_1 \to \delta_2)$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_2 \cup \delta' \sqsubseteq \delta \cup \{C^q_\lambda(\alpha_2, \alpha') \mid q \in Q\}$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$ and a mapping S_1 such that $S_1(\delta_1) = \rho_1$. Let $S'' = \{\alpha' \mapsto l\}$, and let $S = S_2 \cup S_1 \cup S'' \cup ((S_2 \cup S_1 \cup S'') \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_2 is a solution to C_2, so is S. By definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$. Therefore this case is proven if we can show that S satisfies $C^q_\lambda(\alpha_2, \alpha')$ for each $q \in Q$. There are several cases:

  - $R^q_\lambda(q_2) \in A$, where $q_2 \in l_2$: Then one disjunct of the right-hand side of $C^q_\lambda(\alpha_2, \alpha')$ is $q_2 \in \alpha_2$. Since $q_2 \in l_2$, this disjunct is satisfied by S, so S satisfies $C^q_\lambda(\alpha_2, \alpha')$.
  - $q \in l$ but there is no $q_2 \in l_2$ such that $R^q_\lambda(q_2) \in A$: By definition of l there is some q' such that $R^{q'}_\lambda(q'_2)$ and $q' \triangleright^* q$, where $q'_2 \in l_2$. Therefore $q' \neq q$. Then by definition of $\triangleright^*$ there must exist some $q'' \neq q$ such that $q' \triangleright^* q''$ and $q'' \triangleright q$. Therefore by definition of l we have $q'' \in l$. By definition of $C^q_\lambda(\alpha_2, \alpha')$, one disjunct of the right-hand side is $q'' \in \alpha'$. Since $q'' \in l$, S satisfies this disjunct and hence S satisfies the entire constraint.
  - $q \notin l$. Then the left-hand side of $C^q_\lambda(\alpha_2, \alpha')$ is falsified by S, so the entire constraint is vacuously satisfied by S.

- Case T-App: Then $e = e_1\ e_2$ and $\Gamma \vdash e_1 : l(\rho_2 \to \rho)$ and $\Gamma \vdash e_2 : \rho_2$. By induction $\kappa \vdash e_1 : \delta_1 \mid C_1$ and there is a solution S_1 to C_1 such that $S_1(\delta_1) = l(\rho_2 \to \rho)$. Therefore, δ_1 has the form $\alpha(\delta_2 \to \delta')$. Similarly by induction $\kappa \vdash e_2 : \delta_2' \mid C_2$ and there is a solution S_2 to C_2 such that $S_2(\delta_2') = \rho_2$. Therefore by C-App we have $\kappa \vdash e_1\ e_2 : \delta \mid C$, where $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta_2' \sqsubseteq \delta_2 \cup \delta' \sqsubseteq \delta$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let $S = S_1 \cup S_2 \cup (S_1 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_1 and S_2 are respectively solutions to C_1 and C_2, also S is a solution to C_1 and C_2. Further, by definition of S we have $S(\delta_2') = S(\delta_2)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta_2' \sqsubseteq \delta_2$. Similarly, by definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$.

- Case T-Ref: Then $e = \texttt{ref}\ e_0$ and $\rho = l\ \texttt{ref}\ \rho_0$ and $\Gamma \vdash e_0 : \rho_0$ and $\rho_0 = l_0\ \phi_0$ and $l = \{q \mid R^{q'}_{\texttt{ref}}(q_0) \wedge q_0 \in l_0 \wedge q' \triangleright^* q\}$. By induction, we have $\kappa \vdash e_0 : \delta_0 \mid C_0$ and a solution S_0 to C_0 such that $S_0(\delta_0) = \rho_0$. Let $\delta_0 = \alpha_0\ \varphi_0$. Then by C-Ref we have $\kappa \vdash \texttt{ref}\ e : \delta \mid C$, where α' fresh and $\delta' = \alpha'\ \texttt{ref}\ \delta_0$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \delta' \sqsubseteq \delta \cup \{C^q_{\texttt{ref}}(\alpha_0, \alpha') \mid q \in Q\}$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let $S_1 = \{\alpha' \mapsto l\}$, and let $S = S_0 \cup S_1 \cup ((S_0 \cup S_1) \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_0 is a solution to C_0, so is S. By definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$. Therefore this case is proven if we can show that S satisfies $C^q_{\texttt{ref}}(\alpha_0, \alpha')$ for each $q \in Q$. There are several cases:

  - $R^q_{\texttt{ref}}(q_0) \in A$, where $q_0 \in l_0$: Then one disjunct of the right-hand side of $C^q_{\texttt{ref}}(\alpha_0, \alpha')$ is $q_0 \in \alpha_0$. Since $q_0 \in l_0$, this disjunct is satisfied by S, so S satisfies $C^q_{\texttt{ref}}(\alpha_0, \alpha')$.
  - $q \in l$ but there is no $q_0 \in l_0$ such that $R^q_{\texttt{ref}}(q_0) \in A$: By definition of l there is some q' such that $R^{q'}_{\texttt{ref}}(q_0')$ and $q' \triangleright^* q$, where $q_0' \in l_0$. Therefore $q' \neq q$. Then by definition of $\triangleright^*$ there must exist some $q'' \neq q$ such that $q' \triangleright^* q''$ and $q'' \triangleright q$. Therefore by definition of l we have $q'' \in l$. By definition of $C^q_{\texttt{ref}}(\alpha_0, \alpha')$, one disjunct of the right-hand side is $q'' \in \alpha'$. Since $q'' \in l$, S satisfies this disjunct and hence S satisfies the entire constraint.
  - $q \notin l$. Then the left-hand side of $C^q_{\texttt{ref}}(\alpha_0, \alpha')$ is falsified by S, so the entire constraint is vacuously satisfied by S.

- Case T-Assgn: Then $e = e_1 := e_2$ and $\Gamma \vdash e_1 : l\ \texttt{ref}\ \rho$ and $\Gamma \vdash e_2 : \rho$. By induction $\kappa \vdash e_1 : \delta_1 \mid C_1$ and there is a solution S_1 to C_1 such that $S_1(\delta_1) = l\ \texttt{ref}\ \rho$. Therefore, δ_1 has the form $\alpha\ \texttt{ref}\ \delta'$. Similarly by induction $\kappa \vdash e_2 : \delta_2 \mid C_2$ and there is a solution S_2 to C_2 such that $S_2(\delta'') = \rho$. Therefore by C-Assgn we have $\kappa \vdash e_1 := e_2 : \delta \mid C$, where $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta'' \sqsubseteq \delta' \cup \delta' \sqsubseteq \delta$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let $S = S_1 \cup S_2 \cup (S_1 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_1 and S_2 are respectively solutions to C_1 and C_2, also S is a solution to C_1 and C_2. Further, by definition of S we have $S(\delta'') = S(\delta')$, so by Lemmas 1.4 and 1.6 S satisfies $\delta'' \sqsubseteq \delta'$. Similarly, by definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$.

- Case T-Deref: Then $e = !e_0$ and $\Gamma \vdash e_0 : l\ \texttt{ref}\ \rho$. Therefore by induction we have $\kappa \vdash e_0 : \delta_0 \mid C_0$ and a solution S_0 to C_0 such that $S_0(\delta_0) = l\ \texttt{ref}\ (\rho)$. Therefore δ_0 has the form $\alpha_0 \texttt{ref}\ (\delta')$. Then by C-Deref we have $\kappa \vdash !e_0 : \delta \mid C$, where $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \delta' \sqsubseteq \delta$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let let $S = S_0 \cup (S_0 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_0 is a solution to C_0, so is S. Further, by definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$.

- Case T-Assert: Then $e = \texttt{assert}(e_0, q)$ and $\Gamma \vdash e_0 : \rho$ and $\rho = l\ \phi$ and $q \in l$. Therefore by induction we have $\kappa \vdash e_0 : \delta' \mid C_0$ and a solution S_0 to C_0 such that $S_0(\delta') = \rho$. Let $\delta' = \alpha\varphi$. Then by C-Assert we have $\kappa \vdash \texttt{assert}(e_0, q) : \delta \mid C$, where $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \{q \in \alpha\} \cup \delta' \sqsubseteq \delta$. By Lemma 1.3 there is a mapping S' such that $S'(\delta) = \delta'$. Let let $S = S_0 \cup (S_0 \circ S')$. Then $S(\delta) = \rho$, so it remains to show that S is a solution to C. Since S_0 is a solution to C_0, so is S. Further, since $q \in l$, by definition of S we have that S satisfies $q \in \alpha$. Finally, by definition of S we have $S(\delta') = S(\delta)$, so by Lemmas 1.4 and 1.6 S satisfies $\delta' \sqsubseteq \delta$.

- Case T-Sub: Then $\Gamma \vdash e : \rho'$ and $\rho' \leq \rho$. By induction we have $\kappa \vdash e : \delta \mid C$ and there is a solution S' to C such that $S'(\delta) = \rho'$. Consider the last inference rule used in the derivation of $\kappa \vdash e : \delta \mid C$. By inspection of the rules, we see that there is some δ' such that $\delta = \mathit{refresh}(\delta')$, and the only constraint in C that mentions qualifier variables in δ is the constraint $\delta' \sqsubseteq \delta$. Since S' is a solution to C, S' satisfies $\delta' \sqsubseteq \delta$, so by Lemma 1.6 we have $S'(\delta') \leq S'(\delta)$. Since $S'(\delta) = \rho'$, this means $S'(\delta') \leq \rho'$, and since $\rho' \leq \rho$, by Lemma 1.5 also $S'(\delta') \leq \rho$.

  Let $S'' = S' - \{\alpha \mapsto l \mid \alpha \in \delta\}$. By Lemma 1.3 there is a mapping S_0 such that $S_0(\delta) = \delta'$. Then $(S'' \circ S_0)(\delta) = \rho'$, so by Lemma 1.7 there is a mapping S_1 such that $S_1(\delta) = \rho$. Let $S = S'' \cup S_1$. Since $\delta' \sqsubseteq \delta$ is the only constraint in C mentioning qualifier variables in δ, by definition of S'' we have that S'' satisfies all constraints in $C - \delta' \sqsubseteq \delta$, so S does as well. Also by definition of S we have $S(\delta') = S'(\delta')$ and $S(\delta) = \rho$. Therefore, since $S'(\delta') \leq \rho$, by Lemma 1.6 we have that S satisfies $\delta' \sqsubseteq \delta$.

□

Theorem 1.3. If $\kappa \vdash e : \delta \mid C$ and S is a solution to C, then $S(\kappa) \vdash e : S(\delta)$.
Proof By induction on the derivation of $\kappa \vdash e : \delta \mid C$.

- Case C-Int: Then $e = c$ and α' fresh and $\delta' = \alpha'\ \texttt{int}$ and $\delta = \mathit{refresh}(\delta')$ and $C = \delta' \sqsubseteq \delta \cup \{C_c^q(\alpha') \mid q \in Q\}$. Since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash c : S(\delta')$. By T-Int we have $S(\kappa) \vdash c : l\ \texttt{int}$, where $l = \{q \mid R_c^{q'}() \wedge q' \rhd^* q\}$. By T-Sub, we have shown $S(\kappa) \vdash c : S(\delta')$ if we can prove that $l\ \texttt{int} \leq S(\delta')$. This fact in turn follows by S-Int if we can show that $l \supseteq S(\alpha')$.

  Consider some $q \in S(\alpha')$. We will show that $q \in l$. We prove this by complete induction on the number of qualifiers q' such that $q' \rhd^* q$. We have two subcases. First, suppose that $R_c^q() \in A$. Then since $\rhd^*$ is reflexive we have $q \rhd^* q$, so we have that $q \in l$ by the definition of l. Second, suppose that $R_c^q() \notin A$. Since $C_c^q(\alpha')$ is in C, we have that S satisfies $C_c^q(\alpha')$. Since $q \in S(\alpha')$, S satisfies the left-hand side of $C_c^q(\alpha')$, so S must also satisfy the right-hand side. Therefore by the definition of $C_c^q(\alpha')$, there is some q_0 such that $q_0 \rhd q$ and $q_0 \in S(\alpha')$. Then by induction $q_0 \in l$, so there is some q'' such that $R_c^{q''}() \in A$ and $q'' \rhd^* q_0$. Then also $q'' \rhd^* q$, so by the definition of l also $q \in l$.

- Case C-Plus: Then $e = e_1 + e2$ and $\kappa \vdash e_1 : \alpha_1\ \texttt{int} \mid C_1$ and $\kappa \vdash e_2 : \alpha_2\ \texttt{int} \mid C_2$ and α' fresh and $\delta' = \alpha'\ \texttt{int}$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta' \sqsubseteq \delta \cup \{C_+^q(\alpha_1, \alpha_2, \alpha') \mid q \in Q\}$. Since S is a solution to C, S is also a solution to C_1 and C_2, so by induction we have $S(\kappa) \vdash e_1 : S(\alpha_1)\ \texttt{int}$ and $S(\kappa) \vdash e_2 : S(\alpha_2)\ \texttt{int}$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash e_1 + e_2 : S(\delta')$. By T-Plus we have $S(\kappa) \vdash e_1 + e_2 : l\ \texttt{int}$, where $l = \{q \mid R_+^{q'}(q_1, q_2) \wedge$

$q_1 \in S(\alpha_1) \wedge q_2 \in S(\alpha_2) \wedge q' \rhd^* q\}$. By T-Sub, we have shown $S(\kappa) \vdash e_1 + e_2 : S(\delta')$ if we can prove that l $\texttt{int} \leq S(\delta')$. This fact in turn follows by S-Int if we can show that $l \supseteq S(\alpha')$.

Consider some $q \in S(\alpha')$. We will show that $q \in l$. We prove this by complete induction on the number of qualifiers q' such that $q' \rhd^* q$. We have two subcases. First, suppose that $R^q_+(q_1, q_2) \in A$, for some $q_1 \in S(\alpha_1)$ and $q_2 \in S(\alpha_2)$. Then since $\rhd^*$ is reflexive we have $q \rhd^* q$, so we have that $q \in l$ by the definition of l. Second, suppose that there is no $q_1 \in S(\alpha_1)$ and $q_2 \in S(\alpha_2)$ such that $R^q_+(q_1, q_2) \in A$. Since $C^q_+(\alpha_1, \alpha_2, \alpha')$ is in C, we have that S satisfies $C^q_+(\alpha_1, \alpha_2, \alpha')$. Since $q \in S(\alpha')$, S satisfies the left-hand side of $C^q_+(\alpha_1, \alpha_2, \alpha')$, so S must also satisfy the right-hand side. Therefore by the definition of $C^q_+(\alpha_1, \alpha_2, \alpha')$, there is some q_0 such that $q_0 \rhd q$ and $q_0 \in S(\alpha')$. Then by induction $q_0 \in l$, so there is some q'' and $q'_1 \in S(\alpha_1)$ and $q'_2 \in S(\alpha_2)$ such that $R^{q''}_+(q'_1, q'_2) \in A$ and $q'' \rhd^* q_0$. Then also $q'' \rhd^* q$, so by the definition of l also $q \in l$.

- Case C-Var: Then $e = x$ and $\kappa(x) = \delta'$ and $\delta = \mathit{refresh}(\delta')$ and $C = \delta' \sqsubseteq \delta$. Since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash x : S(\delta')$. Since $\kappa(x) = \delta'$, also $S(\kappa)(x) = S(\delta')$, so the result follows by T-Var.

- Case C-Fun: Then $e = \lambda x : \tau_1.e_2$ and $\kappa, x : \delta_1 \vdash e_2 : \delta_2 \mid C_2$ and $\delta_1 = \mathit{embed}(\tau_1)$ and $\delta_2 = \alpha_2\ \varphi_2$ and α' fresh and $\delta' = \alpha'(\delta_1 \to \delta_2)$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_2 \cup \delta' \sqsubseteq \delta \cup \{C^q_\lambda(\alpha_2, \alpha') \mid q \in Q\}$. Since S is a solution to C, S is also a solution to C_2, so by induction we have $S(\kappa, x : \delta_1) \vdash e_2 : S(\delta_2)$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash \lambda x : \tau_1.e_2 : S(\delta')$. Since $\delta_1 = \mathit{embed}(\tau_1)$, by Lemma 1.8 we have $\mathit{strip}(\delta_1) = \tau_1$, so also $\mathit{strip}(S(\delta_1)) = \tau_1$. Therefore by T-Fun we have $S(\kappa) \vdash \lambda x : \tau_1.e_2 : l(S(\delta_1) \to S(\delta_2))$, where $l = \{q \mid R^{q'}_\lambda(q_2) \wedge q_2 \in S(\alpha_2) \wedge q' \rhd^* q\}$. By T-Sub, we have shown $S(\kappa) \vdash \lambda x : \tau_1.e_2 : S(\delta')$ if we can prove that $l(S(\delta_1) \to S(\delta_2)) \leq S(\delta')$. This fact in turn follows by S-Fun if we can show that $l \supseteq S(\alpha')$ and $S(\delta_1) \leq S(\delta_1)$ and $S(\delta_2) \leq S(\delta_2)$. The last two obligations follow from Lemma 1.4, so it remains to show that $l \supseteq S(\alpha')$.

  Consider some $q \in S(\alpha')$. We will show that $q \in l$. We prove this by complete induction on the number of qualifiers q' such that $q' \rhd^* q$. We have two subcases. First, suppose that $R^q_\lambda(q_2) \in A$, for some $q_2 \in S(\alpha_2)$. Then since $\rhd^*$ is reflexive we have $q \rhd^* q$, so we have that $q \in l$ by the definition of l. Second, suppose that there is no $q_2 \in S(\alpha_2)$ such that $R^q_\lambda(q_2) \in A$. Since $C^q_\lambda(\alpha_2, \alpha')$ is in C, we have that S satisfies $C^q_\lambda(\alpha_2, \alpha')$. Since $q \in S(\alpha')$, S satisfies the left-hand side of $C^q_\lambda(\alpha_2, \alpha')$, so S must also satisfy the right-hand side. Therefore by the definition of $C^q_\lambda(\alpha_2, \alpha')$, there is some q_0 such that $q_0 \rhd q$ and $q_0 \in S(\alpha')$. Then by induction $q_0 \in l$, so there is some q'' and $q'_2 \in S(\alpha_2)$ such that $R^{q''}_+(q'_2) \in A$ and $q'' \rhd^* q_0$. Then also $q'' \rhd^* q$, so by the definition of l also $q \in l$.

- Case C-App: Then $e = e_1\ e_2$ and $\kappa \vdash e_1 : \alpha(\delta_2 \to \delta') \mid C_1$ and $\kappa \vdash e_2 : \delta'_2 \mid C_2$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta'_2 \sqsubseteq \delta_2 \cup \delta' \sqsubseteq \delta$. Since S is a solution to C, S is also a solution to C_1 and C_2, so by induction we have $S(\kappa) \vdash e_1 : S(\alpha(\delta_2 \to \delta'))$ and $S(\kappa) \vdash e_2 : S(\delta'_2)$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta'_2) \leq S(\delta_2)$ and $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash e_1\ e_2 : S(\delta')$. Since $S(\kappa) \vdash e_2 : S(\delta'_2)$ and $S(\delta'_2) \leq S(\delta_2)$, by T-Sub we have $S(\kappa) \vdash e_2 : S(\delta_2)$, so the result follows by T-App.

- Case C-Ref: Then $e = \texttt{ref}\ e_0$ and $\kappa \vdash e_0 : \delta_0 \mid C_0$ and $\delta_0 = \alpha_0\ \varphi_0$ and α' fresh and $\delta' = \alpha'\ \texttt{ref}\ \delta_0$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \delta' \sqsubseteq \delta \cup \{C^q_{\texttt{ref}}(\alpha_0, \alpha') \mid q \in Q\}$. Since S is a

solution to C, S is also a solution to C_0, so by induction we have $S(\kappa) \vdash e_0 : S(\delta_0)$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash \texttt{ref}\ e_0 : S(\delta')$. By T-Ref we have $S(\kappa) \vdash \texttt{ref}\ e_0 : l\ \texttt{ref}\ S(\delta_0)$, where $l = \{q \mid R^{q'}_{\texttt{ref}}(q_0) \wedge q_0 \in S(\alpha_0) \wedge \wedge q' \triangleright^* q\}$. By T-Sub, we have shown $S(\kappa) \vdash \texttt{ref}\ e_0 : S(\delta')$ if we can prove that $l\ \texttt{ref}\ S(\delta_0) \leq S(\delta')$. This fact in turn follows by S-Ref if we can show that $l \supseteq S(\alpha')$ and $S(\delta_0) \leq S(\delta_0)$. The latter obligation follows by Lemma 1.4, so it remains to show that $l \supseteq S(\alpha')$.

Consider some $q \in S(\alpha')$. We will show that $q \in l$. We prove this by complete induction on the number of qualifiers q' such that $q' \triangleright^* q$. We have two subcases. First, suppose that $R^q_{\texttt{ref}}(q_0) \in A$, for some $q_0 \in S(\alpha_0)$. Then since $\triangleright^*$ is reflexive we have $q \triangleright^* q$, so we have that $q \in l$ by the definition of l. Second, suppose that there is no $q_0 \in S(\alpha_0)$ such that $R^q_{\texttt{ref}}(q_0) \in A$. Since $C^q_{\texttt{ref}}(\alpha_0, \alpha')$ is in C, we have that S satisfies $C^q_{\texttt{ref}}(\alpha_0, \alpha')$. Since $q \in S(\alpha')$, S satisfies the left-hand side of $C^q_{\texttt{ref}}(\alpha_0, \alpha')$, so S must also satisfy the right-hand side. Therefore by the definition of $C^q_{\texttt{ref}}(\alpha_0, \alpha')$, there is some q_0 such that $q_0 \triangleright q$ and $q_0 \in S(\alpha')$. Then by induction $q_0 \in l$, so there is some q'' and $q'_0 \in S(\alpha_0)$ such that $R^{q''}_{\texttt{ref}}(q'_0) \in A$ and $q'' \triangleright^* q_0$. Then also $q'' \triangleright^* q$, so by the definition of l also $q \in l$.

- Case C-Assgn: Then $e = e_1 := e_2$ and $\kappa \vdash e_1 : \alpha\ \texttt{ref}\ \delta' \mid C_1$ and $\kappa \vdash e_2 : \delta'' \mid C_2$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_1 \cup C_2 \cup \delta'' \sqsubseteq \delta' \cup \delta' \sqsubseteq \delta$. Since S is a solution to C, S is also a solution to C_1 and C_2, so by induction we have $S(\kappa) \vdash e_1 : S(\alpha\ \texttt{ref}\ \delta')$ and $S(\kappa) \vdash e_2 : S(\delta'')$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta'') \leq S(\delta')$ and $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash e_1\ e_2 : S(\delta')$. Since $S(\kappa) \vdash e_2 : S(\delta'')$ and $S(\delta'') \leq S(\delta')$, by T-Sub we have $S(\kappa) \vdash e_2 : S(\delta')$, so the result follows by T-Assgn.

- Case C-Deref: Then $e = !e_0$ and $\kappa \vdash e_0 : \alpha\ \texttt{ref}\ \delta' \mid C_0$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \delta' \sqsubseteq \delta$. Since S is a solution to C, S is also a solution of C_0, so by induction we have $S(\kappa) \vdash e_0 : S(\alpha\ \texttt{ref}\ \delta')$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash !e_0 : S(\delta')$. The result follows by T-Deref.

- Case C-Assert: Then $e = \texttt{assert}(e_0, q)$ and $\kappa \vdash e_0 : \delta' \mid C_0$ and $\delta' = \alpha\ \varphi$ and $\delta = \mathit{refresh}(\delta')$ and $C = C_0 \cup \{q \in \alpha\} \cup \delta' \sqsubseteq \delta$. Since S is a solution to C, S is also a solution of C_0, so by induction we have $S(\kappa) \vdash e_0 : S(\delta')$. Also since S is a solution to C, by Lemma 1.6 we have $S(\delta') \leq S(\delta)$. Therefore by T-Sub, this case is proven if we can prove $S(\kappa) \vdash \texttt{assert}(e_0, q) : S(\delta')$. Since S is a solution to C, we have $q \in S(\alpha)$, so the result follows by T-Assert.

□

Lemma 1.3. (a) If $\delta = \mathit{refresh}(\delta')$ then there is a mapping S from the qualifier variables in δ to the qualifier variables in δ' such that $S(\delta) = \delta'$. (b) If $\delta = \mathit{refresh}(\rho)$ then there is a mapping S from the qualifier variables in δ to the qualifier variables in ρ such that $S(\delta) = \rho$.

Lemma 1.4. $\rho \leq \rho$

Lemma 1.5. If $\rho_1 \leq \rho_2$ and $\rho_2 \leq \rho_3$, then $\rho_1 \leq \rho_3$.

Lemma 1.6. $S(\delta') \leq S(\delta)$ if and only if S satisfies $\delta' \sqsubseteq \delta$.

Lemma 1.7. If $\delta = \mathit{refresh}(\delta')$ and $S'(\delta) = \rho'$ and $\rho' \leq \rho$, then there exists a mapping S such that $S(\delta) = \rho$.

Lemma 1.8. If $\delta = \mathit{embed}(\tau)$, then $\mathit{strip}(\delta) = \tau$.

2 Rule Inference

A naive approach to rule inference is to generate each candidate rule and use the qualifier validator to remove all candidates that do not respect the intended invariant. However, since qualifier validation is relatively expensive, requiring usage of decision procedures, and since there are an exponential number of candidates in the number of qualifiers, it is desirable to minimize the number of candidates that need to be explicitly considered.[2] To efficiently search the space of candidate rules, we define a partial order $\preceq$ that formalizes the situation when one candidate subsumes another.

The most precise partial ordering on `case` clauses is logical implication. For example, the third `case` clause in Figure 1 corresponds to the following formula, obtained by replacing qualifiers with their invariants:

$$\texttt{value(E1)} \neq 0 \wedge \texttt{value(E2)} \neq 0 \Rightarrow \texttt{value(E1} * \texttt{E2)} \neq 0$$

The above clause subsumes a clause that requires both `E1` and `E2` to be `pos` instead of `nonzero`, since the above formula logically implies the formula associated with the new clause. Unfortunately, precisely computing this partial order requires an exponential number of calls to decision procedures to reason about logical implication, which is exactly what we are trying to avoid.

Instead, our approach is to use logical implication to define a partial ordering on individual qualifiers, but to then lift this partial ordering to `case` clauses in a purely syntactic way. Therefore, we need only make a quadratic number of calls to the decision procedures in order to compute the partial order. This approximation of the "true" partial ordering is still guaranteed to completely exhaust the space of candidates, but it is now possible to produce qualifier rules that are redundant. The rest of this section formalizes our partial order and describes the rule inference algorithm.

2.1 The Partial Order

We assume the set Q contains every qualifier in this system. Every qualifier $q \in Q$ has a semantic predicate defined as $q(\cdot) : X \to \{T, F\}$, where X is a set of opaque values. For any two qualifiers $q_1, q_2 \in Q$, we assume that there exists some value x such that $q_1(x) \not\Leftrightarrow q_2(x)$. Thus no two qualifiers have the same semantic predicate.

Definition 2.1. The *Q-relation* $\preceq_Q$ is defined as follows:

$$q_1 \preceq_Q q_2 \stackrel{\Delta}{=} \forall x.q_1(x) \Rightarrow q_2(x)$$

Theorem 2.1. The Q-relation $\preceq_Q$ is transitive, reflexive, and antisymmetric

Proof Since $\preceq_Q$ is based on implication, it shares the transitive, reflexive, and antisymmetric properties of implication since by definition no two distinct qualifiers have the same semantic meaning. □

Definition 2.2. We define the set $S = \mathcal{P}(Q)$ of qualsets. For any qualset $s \in S$, $s(\cdot) : X \to \{T, F\}$ is its semantic meaning which is defined as follows:

$$s(x) \stackrel{\Delta}{=} \forall q \in s.q(x)$$

[2] Conceptually, there are an infinite number of candidates, due to constants. We handle constants through a simple heuristic that works well in practice. For each qualifier, we only consider a single candidate rule (possibly) containing constants, which is derived from the qualifier's invariant by replacing all references to `value(E)` with a metavariable ranging over constants.

Definition 2.3. $\preceq_S$ is a binary relation on qualsets such that, for any qualsets s_1 and s_2

$$s_1 \preceq_S s_2 \stackrel{\Delta}{=} \forall q_2 \in s_2.\exists q_1 \in s_1.q_1 \preceq_Q q_2$$

Theorem 2.2. For all $s_1, s_2 \in S$

$$s_1 \preceq_S s_2 \Rightarrow \forall x.(s_1(x) \Rightarrow s_2(x))$$

Proof

We will prove this by contradiction. To be false, $s_1(x)$ must hold while $s_2(x)$ must not. This means that there is at least one qualifier $a \in s_2$ for which $a(x)$ is false. By the definition of $\preceq_S$, there exists some qualifier $b \in s_1$ such that $b \preceq_Q a$. By the definition of $\preceq_Q$, this means that $\forall x.b(x) \Rightarrow a(x)$. Since $a(x)$ is false, $b(x)$ must also be false by contrapositive. But since $s_1(x)$ holds in the above assumption, it means that every qualifier expression including $b(x)$ must hold. This is a contradiction, thus the theorem holds. □

This approach of using $\preceq_Q$ as the basis for the partial order on qualsets has some limitations. For example, consider qualifiers $q_1, q_2, q_3 \in Q$ such that $q_2 \not\Rightarrow q_1$ and $q_3 \not\Rightarrow q_1$, but $q_2 \wedge q_3 \Rightarrow q_1$. Since our partial ordering of qualifiers only deals with pairwise implications, and not grouped implications as in the example, it is not the case that $\{q_2, q_3\} \preceq_S \{q_1\}$. We could modify the $\preceq_S$ relation to directly check implication of qualsets, but then we would need to do exponential work to find all implication sets. Our technique of only doing pairwise comparisons keeps the initial work to $O(|Q|^2)$ implication queries while simultaneously getting most of the true redundancies.

Theorem 2.3. The S-relation is reflexive and transitive.

Proof

Take any qualset $s \in S$. For every qualifier $q \in s$, there exists some $q' \in s$ such that $q' \preceq_Q q$, namely $q' = q$. This must be true due to reflexivity of $\preceq_Q$. This is precisely the definition of $\preceq_S$, thus $s \preceq_S s$, and thus $\preceq_S$ is reflexive.

Take any qualsets $s_1, s_2, s_3 \in S$ such that $s_1 \preceq_S s_2$ and $s_2 \preceq_S s_3$. Take any qualifier $q \in s_3$. By the definition of $\preceq_S$, there exists some $q' \in s_2$ such that $q' \preceq_Q q$. By the same logic, there exists some $q'' \in s_1$ such that $q'' \preceq_Q q'$. By the transitivity of $\preceq_Q$, $q'' \preceq_Q q$. Since there is such a $q'' \in s_1$ for every $q \in s_3$, then by the definition of $\preceq_S$, $s_1 \preceq_S s_3$. Thus $\preceq_S$ is transitive. □

Definition 2.4. A set s is *pairwise Q-unrelated* if:

$$\forall x, y \in s.(x = y) \vee \neg(x \preceq_Q y)$$

The set of qualsets $S_{\bar{Q}} \subset S$ contains exactly those qualsets that are pairwise Q-unrelated.

In plain English, every pair of qualifiers in a qualset are unrelated.

Lemma 2.1. Given two qualifiers a and b such that $a \preceq_Q b$, then $a(x) \wedge b(x) \Leftrightarrow a(x)$.

Proof

This can be proved through simple boolean manipulation. From the definition of $a \preceq_Q b$, we know that $\forall x.a(x) \Rightarrow b(x)$. We have as a tautology that $a(x) \Rightarrow a(x)$. Since we also have $a(x) \Rightarrow b(x)$, $a(x) \Rightarrow a(x) \wedge b(x)$. In addition, since adding constraints to an implication does not change the truth of the implication, we also have that $a(x) \wedge b(x) \Rightarrow a(x)$. This gives us both the if and only if statements of $a(x) \wedge b(x) \Leftrightarrow a(x)$, and thus the statement must hold. □

Theorem 2.4. [Qualset Pairwise Q-unrelated Cannonical Form]

For every qualset s which is not pairwise Q-unrelated, there exists some qualset s' which is pairwise Q-unrelated, and such that $\forall x.s(x) \Leftrightarrow s'(x)$.

Proof

By lemma 2.1, for any two related qualifiers a and b, $a(x) \wedge b(x) \Leftrightarrow a(x)$. Take any qualset s which is not Q-unrelated. Since s is not Q-unrelated, there exists some pair of qualifiers $a, b \in s$ such that $a \preceq_Q b$. In the expansion of $s(x)$, we get the statement $a(x) \wedge b(x) \wedge \cdots$. Since $\wedge$ is commutative, the position of $a(x)$ and $b(x)$ in the expression is irrelevant. By the lemma, this is equivalent to $a(x) \wedge \cdots$, omitting $b(x)$ entirely. This statement has the same meaning as the original, and can be derived by the qualset $s' = s - b$. Since b is no longer in s, the prior related pair no longer exists. If there still remains a related pair in s', the same procedure can be applied, still preserving the meaning. This can be applied until there are no more such unrelated pairs. Since the semantic meaning is preserved, this final Q-unrelated set is equivalent to the original. Since this can be done for every non-Q-unrelated set, the set of Q-related sets is semantically equivalent to the set of all qualsets. □

Theorem 2.5. The S-relation $\preceq_S$ is antisymmetric over all pairwise Q-unrelated sets.

Proof

The property of antisymmetry holds that $S_1 \preceq_S S_2 \wedge S_2 \preceq_S S_1 \Rightarrow S_1 = S_2$. We will prove this by contradiction.

Assume that for two pairwise Q-unrelated sets S_1 and S_2 that $S_1 \preceq_S S_2$, $S_2 \preceq_S S_1$, and $S_1 \neq S_2$. Since the sets are not equal, there exists some qualifier $q \in S_1$ such that $q \notin S_2$ or $q \in S_2$ and $q \notin S_1$. Since the premise is symmetrical for S_1 and S_2 we can assume the former case without loss of generality.

By the definition of the S-relation, the following must hold:

$$\forall y \in S_1.\exists x \in S_2.x \preceq_Q y \tag{1}$$

$$\forall y \in S_2.\exists x \in S_1.x \preceq_Q y \tag{2}$$

By the first equation, there must exist some qualifier $a \in S_2$ such that $a \preceq_Q q$. We can see that $a \neq q$ because $q \notin S_2$. By the second equation, there must be some qualifier $b \in S_1$ such that $b \preceq_Q a$. The qualifier $b \neq q$ because if $b = q$ then we would then have $q \preceq_Q a$ and $a \preceq_Q q$ where $a \neq q$, which would violate the antisymmetry of $\preceq_Q$. By the transitivity of $\preceq_Q$, $b \preceq_Q q$. However, both b and q are in S_1. Since $b \neq q$, this violates the pairwise Q-unrelated property of S_1, and thus we have a contradiction. □

Definition 2.5. We define $M_k = S^k_{\bar{Q}}$ as k-nary tuples of pairwise Q-unrelated qualsets. Given $m = (m_1, m_2, \ldots, m_k) \in M_k$, $m(\cdot) : X^k \to \{T, F\}$ is its semantic meaning which is defined as follows:

$$m(x_1, \ldots, x_k) \triangleq \forall i \in \{1, \ldots, k\}.(\pi_i m)(x_i)$$

Definition 2.6. $\preceq_M$ is a binary relation over M_k for arbitrary k such that, for any k-nary tuples of qualsets m and n

$$m \preceq_M n \triangleq \forall i \in \{1, \ldots, k\}.\pi_i m \preceq_S \pi_i n$$

Theorem 2.6. For all $m, n \in M_k$,

$$m \preceq_M n \Rightarrow \forall x_1, x_2, \ldots, x_k.(m(x_1, x_2, \ldots, x_k) \Rightarrow n(x_1, x_2, \ldots, x_k))$$

Proof

In order to be false, $m(x_1, x_2, \ldots, x_k)$ must be true while $n(x_1, x_2, \ldots, x_k)$ must be false. For this to be true, there must be some i such that $\pi_i n(x_i)$ is false, and $\pi_i m(x_i)$ is true. Thus $\pi_i m(x_i) \not\Rightarrow \pi_i n(x_i)$. But the definition of $\preceq_M$ holds that $\forall x.\pi_i m(x_i) \Rightarrow \pi_i n(x_i)$, which is a contradiction. □

Theorem 2.7. The relation $\preceq_M$ is reflexive, transitive, and antisymmetric.

Proof

Consider any multiset m. $m \preceq_M m$ is the equivalent of $m_1 \preceq_S m_1 \wedge m_2 \preceq_S m_2 \cdots$, which must follow due to the reflexivity of $\preceq_S$.

Given sets m, n, and p, say that $m \preceq_M n$ and $n \preceq_M p$. By the definition of $\preceq_M$, this is the same as $m_1 \preceq_S n_1 \wedge \cdots \wedge n_1 \preceq_S p_1 \wedge \cdots$. Since for any $i \leq k$, there will be the two statements $m_i \preceq_S n_i$ and $n_i \preceq_S p_i$, it follows that $m_i \preceq_S p_i$ by transitivity of $\preceq_S$. Since this is true for every i, it follows that $m \preceq_M p$ by the definition of $\preceq_M$. Thus $\preceq_M$ is transitive.

Assume that there exist $m, n \in M_k$ such that $m \neq n$, $m \preceq_M n$ and $n \preceq_M m$. This means there must be some $i \in [1, k]$ such that $m_i \neq n_i$. By the definition of $\preceq_M$, it must be true that $m_i \preceq_S n_i$ and $n_i \preceq_S m_i$. This contradicts the antisymmetry of $\preceq_S$. Thus $\preceq_M$ must be antisymmetric. □

Next we define the notion of candidate qualifier rules:

Definition 2.7. A *candidate* c can be considered to be a triple containing the constructor p used as the pattern; a k-tuple of qualsets, one per subexpression of p, representing the clause's condition; and the qualifier q that the clause is defined for. We denote the set of candidates as C.

Definition 2.8. We define the relation $\preceq_C$ on candidates as follows:

$$(p_1, m_1, q_1) \preceq_C (p_2, m_2, q_2) \stackrel{\Delta}{=} p_1 = p_2 \wedge m_2 \preceq_M m_1 \wedge q_1 \preceq_Q q_2$$

Theorem 2.8. The relation $\preceq_C$ is reflexive, transitive, and anti-symmetric.

Proof

For the purposes of this proof, we define c_1, c_2, and c_3 to be candidates in C, where $c_1 = (m_1, p_1, t_2)$, $c_2 = (m_2, p_2, t_2)$, and $c_3 = (m_3, p_3, t_3)$.

The statement $c_1 \preceq_C c_1$ is equivalent to the statement $m_1 \preceq_M m_1 \wedge p_1 = p_1 \wedge t_1 \preceq_Q t_1$. The substatement $m_1 \preceq_M m_1$ must be true by the reflexive property of $\preceq_M$. $p_1 = p_2$ is true by the reflexivity of equality. The substatement $t_1 \preceq_Q t_1$ is also true by the reflexive property of $\preceq_Q$. Thus the conjunction of these substatements is also true, thus the statement $c_1 \preceq_C c_1$ is true, thus $\preceq_C$ is reflexive.

Assume that $c_1 \preceq_C c_2$ and $c_2 \preceq_C c_3$. From these relations, we know that $m_2 \preceq_M m_1$ and $m_3 \preceq_M m_2$. By the transitivity of $\preceq_M$, we know that $m_3 \preceq_M m_1$. Similarly we know that $p_1 = p_2$ and $p_2 = p_3$. By the transitivity of equality, we know that $p_1 = p_3$. Finally, we know that $t_1 \preceq_Q t_2$ and $t_2 \preceq_Q t_3$. By the transitivity of $\preceq_Q$, we know that $t_1 \preceq_Q t_3$. Altogether, we have $m_3 \preceq_M m_1 \wedge p_1 = p_3 \wedge t_1 \preceq_Q t_3$, which is equivalent to $c_1 \preceq_C c_3$. Thus $\preceq_C$ is transitive.

We prove the antisymmetric property of $\preceq_C$ by contradiction. Assume that $c_1 \preceq_C c_2$ and $c_2 \preceq_C c_1$ and $c_1 \neq c_2$. The last term implies that either $m_1 \neq m_2$, $p_1 \neq p_2$, or $t_1 \neq t_2$. If $m_1 \neq m_2$, then that combined with the fact that $m_1 \preceq_M m_2$ and $m_2 \preceq_M m_1$ would create a contradiction. Similarly, if $t_1 \neq t_2$, then the combination of that with $t_1 \preceq_Q t_2$ and $t_2 \preceq_Q t_1$ would also create a

contradiction. If $p_1 \neq p_2$, then that contradicts with $p_1 = p_2$. Thus if $c_1 \preceq_C c_2$ and $c_2 \preceq_C c_1$, then $c_1 = c_2$. Thus $\preceq_C$ is antisymmetric. □

Finally, we formalize the notion of candidate validation. For this formalization, we model a constructor p as a function. For example, the + constructor would be represented as a function that takes two values and returns their sum.

Definition 2.9. A candidate $(p, m, q) \in (X^k \to X) \times M_k \times Q$ is *valid* if the following formula is logically valid:

$$\forall x_1, x_2, \ldots, x_k \in X.m(x_1, x_2, \ldots, x_k) \Rightarrow q(p(x_1, x_2, \ldots, x_k))$$

The following theorem formalizes the fact that our partial order is sensible: if candidate c_1 subsumes c_2 according to our partial order and c_1 is valid, then so is c_2.

Theorem 2.9. Given any two candidates $c_1, c_2 \in C$, the following holds:

$$c_1 \preceq_C c_2 \Rightarrow (c_1 \text{ is valid} \Rightarrow c_2 \text{ is valid})$$

Proof

Assume that $c_1 \preceq_C c_2$, where $c_1 = (p_1, m_1, q_1)$, and $c_2 = (p_2, m_2, q_2)$. By definition, $m_2 \preceq_M m_1$, $p_1 = p_2$, and $q_1 \preceq_Q q_2$. Assume that c_1 is valid. Thus $\forall x_1, \ldots, x_k \in X.m_1(x_1, \ldots, x_k) \Rightarrow q_1(p_1(x_1, \ldots, x_k))$. Since $m_2 \preceq_M m_1$, by theorem 2.6 this means that $\forall x_1, \ldots, x_k \in X.m_2(x_1, \ldots, x_k) \Rightarrow m_1(x_1, \ldots, x_k)$. We also know that $p_1 = p_2$. We will simply use p to refer to this function. Similarly, $q_1 \preceq_Q q_2$ implies that $\forall x \in X.q_1(x) \Rightarrow q_2(x)$. Specifically, this is true when $x = p(x_1, \ldots, x_k)$. Thus $\forall x_1, \ldots, x_k \in X.q_1(p(x_1, \ldots, x_k)) \Rightarrow q_2(p(x_1, \ldots, x_k))$. In summary, we have the fact that

$$\forall x_1, \ldots, x_k \in X.m_2(x_1, \ldots, x_k) \Rightarrow m_1(x_1, \ldots, x_k) \Rightarrow q_1(p(x_1, \ldots, x_k)) \Rightarrow q_2(p(x_1, \ldots, x_k))$$

By the transitive property of implication, this gives us

$$\forall x_1, \ldots, x_k \in X.m_2(x_1, \ldots, x_k) \Rightarrow q_2(p(x_1, \ldots, x_k))$$

which is precisely the necessary statement to show that c_2 is valid. □

2.2 The Algorithm

To generate all the valid rules that are not redundant, we need a way to enumerate candidates in topological order according to the $\preceq_C$ relation. Therefore, by its definition, for each target qualifier $\preceq_Q$, which should be enumerated in topological order, we need to enumerate k-nary tuples of qualsets in reverse topological order. To do so, we first illustrate how to map a qualifier, qualset, and k-nary tuple of qualsets to integers, such that this mapping creates a total ordering that is consistent with a topological sort. We will then describe how to use this total ordering in a procedure for generating the valid candidate rules.

We start with the set of qualifiers Q. We make an arbitrary reverse topological sort of Q by the partial ordering $\preceq_Q$ by making $|Q|^2$ queries to decision procedures. We will number the qualifiers $q_0, q_1, \ldots, q_{|Q|-1}$.

Definition 2.10. The function $f_Q : Q \to N$ is defined as follows:

$$f_Q(q_i) \stackrel{\Delta}{=} 2^i$$

where i is the qualifier's index in the above ordering.

Definition 2.11. The function $f_S : S \to N$ is defined as follows:

$$f_S(s) \stackrel{\Delta}{=} \sum_{q \in s} f_Q(q)$$

Lemma 2.2. For any two qualsets $s, t \in S_{\bar{Q}}$, if $s \subseteq t$ then $f_S(s) \leq f_S(t)$
Proof

Let $u = t - s$. The value $f_S(u)$ is either zero or greater than zero. Since s and u are disjoint, then by the definition of f_S, $f_S(t) = f_S(s \cup u) = f_S(s) + f_S(u)$. Since $f_S(u) \geq 0$, this means that $f_S(s) \leq f_S(t)$. □

Lemma 2.3. For any $k \in Z^+$,

$$\sum_{i=0}^{k-1} 2^i < 2^k$$

Proof

We prove this by induction. For the base case of $k = 1$, we get that:

$$\sum_{i=0}^{0} 2^i < 2^1 \tag{3}$$

$$1 < 2 \tag{4}$$

For the inductive case, we prove that given that the above holds for k, then it holds for $k + 1$. By the inductive case we are given:

$$\sum_{i=0}^{k-1} 2^i < 2^k$$

$$\sum_{i=0}^{k-1} 2^i + 2^k < 2^k + 2^k$$

$$\sum_{i=0}^{k} 2^i < 2^{k+1}$$

This is the same form as $k + 1$, thus the statement must hold for all $k \geq 1$. □

Lemma 2.4. Given any qualifier q and any set of qualifiers s such that $q \notin s \wedge \forall q' \in s.q \preceq_Q q'$, then $f_Q(q) > f_S(s)$.
Proof

Say that $q = q_k$. Due to the topological sort by $\preceq_Q$, we know that any qualifiers r such that $q \preceq_Q r$ must be in the set $\{q_0, q_2, \ldots, q_{k-1}\}$. This means that the set s must be a subset of the set $\{q_0, q_1, \ldots, q_{k-1}\}$.

Let $s' = \{q_0, q_1, \ldots, q_{k-1}\}$. By the definition of f_S and f_Q we get

$$\begin{aligned} f_S(s') &= \sum_{q \in s'} f_Q(q) \\ &= \sum_{i=0}^{k-1} f_Q(q_i) \\ &= \sum_{i=0}^{k-1} 2^i \end{aligned}$$

By the definition of f_Q, we know that $f_Q(q) = 2^k$. By lemma 2.3, we have

$$\sum_{i=0}^{k-1} 2^i < 2^k$$

Thus $f_Q(q) > f_S(s')$. Since any valid s is a subset of s', then $f_S(s') \geq f_S(s)$ by lemma 2.2. By transitivity, $f_Q(q) > f_S(s)$ for all valid s. □

Theorem 2.10. For any pairwise Q-unrelated qualsets s and t,

$$s \preceq_S t \Rightarrow f_S(s) \geq f_S(t).$$

Proof

We define a relation $R : s \to \mathcal{P}(t)$ with the following properties:

$$\bigcup_{q \in s} R(q) = t$$

$$\forall q_1, q_2 \in s . q_1 = q_2 \vee R(q_1) \cap R(q_2) = \emptyset$$

$$\forall q_s \in s . \forall q_t \in R(q_s) . q_s \preceq_Q q_t$$

Such a relation R can be created for any two qualsets s and t where $s \preceq_S t$ by the following procedure:

Start with the empty relation R where for every $q \in s$, $R(s) = \emptyset$. For every element $q_t \in t$, find an element $q_s \in s$ such that $q_s \preceq_Q q_t$ (such an element must exist by the definition of $\preceq_S$). Create a new relation R' such that for all qualifiers which are not q_s, $R'(q) = R(q)$, and for the qualifier q_s, $R'(q_s) = R(q_s) \cup \{q_t\}$. Replace R with R', and repeat for the next element of t. Since each element of t is added into the range of R, the first statement must hold. Since each element of t is added only once into the range of R, the second statement must hold. Since the method which we generated the terms required that each element of t be put in the set relating to an element of s which it relates to, the third statement holds.

We will prove that for every $q_s \in s$ that $f_Q(q_s) \geq f_S(R(q_s))$. Since t is pairwise Q-unrelated, we know that if $q_s \in R(q_s)$, then $\{q_s\} = R(q_s)$. If there were any other elements besides q_s in $R(q_s)$, then it would mean that t would contain two elements which are Q-related (namely q_s and any other qualifier in that set). Thus this must be true. This leaves us with two possibilities: Either $\{q_s\} = R(q_s)$ or $q_s \notin R(q_s)$. In the former case, $f_Q(q_s) = f_S(\{q_s\}) = f_S(R(q_s))$, and thus the two are equal which is sufficient for the statement to hold. In the latter case, we know that $q_s \notin R(q_s)$,

and that $\forall q \in R(q_s).q_s \preceq_Q q$. According to lemma 2.4, this means that $f_Q(q_s) > f_S(R(q_s))$. Thus in either case, the statement holds.

From this proof we get a series of inequalities. We know by definition of f_S that $f_S(s) = \sum_{q \in s} f_Q(q)$, which means that the sum of the left hand side of these inequalities is equivalent to $f_S(s)$. We also know that $f_S(s_1) + f_S(s_2) = f_S(s_1 \cup s_2)$ if $s_1 \cap s_2 = \emptyset$. By the second condition on R, this is precisely the case for the elements of the range of R. Since the union of all of the ranges of R precisely equal t, then $\sum_{q \in s} f_S(R(q)) = f_S(t)$. We just summed up the left and right sides of a series of inequalities, such that the direction of the inequalities do not change. Thus the same inequality must apply to the sum. Thus $f_S(s) \geq f_S(t)$. □

Definition 2.12. The function $f_M : M_k \to N$ is defined as follows:

$$f_M(m) \triangleq \sum_{s \in m} f_S(s)$$

Theorem 2.11. For any two multisets m_1 and m_2,

$$m_1 \preceq_M m_2 \Rightarrow f_M(m_1) \geq f_M(m_2).$$

Proof

For every pair of qualsets $s_1 \in m_1$ and $s_2 \in m_2$, it holds that $s_1 \preceq_S s_2$ by definition of $\preceq_M$. By the previous theorem, $f_S(s_1) \geq f_S(s_2)$. Since this is true for every pair, summing up the values from the function f_S on each side should preserve the fact that $\sum_{s_1 \in m_1} f_S(s_1) \geq \sum_{s_2 \in m_2} f_S(s_2)$. Since this is the definition of f_M, the theorem holds. □

We now use these functions to enumerate candidates in topological order according to $\preceq_C$. Consider generating all valid `case` rules for a single qualifier q. Further, fix a particular constructor p to use in the rule's pattern, and assume that this constructor has exactly one subexpression. Let W be a worklist of pairs of the form (s, l) where s is a qualset and l is a list of qualifiers. Initialize the set W to $\{(\emptyset, [q_1, q_2, q_3, \ldots])\}$, where $[q_1, q_2, q_3, \ldots]$ is an ordered list of all the qualifiers in reverse topological order according to $\preceq_Q$. Using reverse topological order ensures we will generate qualsets for use in a `case` rule from most-general to most-specific, which is necessary given the contravariance in the definition of $\preceq$. We also maintain a set T of valid `case` rules, initialized to $\emptyset$.

1. If W is empty, we are done and T contains all the valid non-redundant rules. Otherwise, remove the tuple (s, l) from W such that for all other $(s', l') \in W$ we have $s \not\preceq_S s'$. To do so efficiently, we maintain the invariant that tuples in W are in sorted order according to f_S.

2. If there is some candidate $(p', s', q') \in T$ such that $(p', s', q') \preceq_C (p, s, q)$ then (p, s, q) is redundant, so we drop the tuple (s, l) and return to the previous step. Otherwise, we continue to the next step.

3. We run our framework's qualifier validator on (p, s, q). If (p, s, q) is valid, we add (p, s, q) to T. If not, then we need to check less-specific candidates. For each $q \in l$, we add the pair $(s \cup \{q\}, l')$ to W, where l' is the suffix of l after q. These pairs are placed appropriately in W to maintain its sortedness, as described earlier; it is straightforward to compute $f_S(s \cup \{q\})$ incrementally from $f_S(s)$.

In the case when the constructor p has $k > 1$ subexpressions, we need to enumerate k-ary multisets. To do so, the worklist W now contains k-tuples of pairs of the form (s, l) and is sorted according to f_M. When adding new elements to W, we apply the procedure described in Step 3 above to each component of the k-tuple individually, keeping all other components unchanged. The only subtlety is that we want to avoid generating redundant tuples. For example, if $q_1 \preceq_Q q_2$, then the tuple $(\{q_2\}, \{q_2\})$ could be a successor of both $(\{q_1\}, \{q_2\})$ and $(\{q_2\}, \{q_1\})$. To avoid this duplication, we only augment a component of a k-tuple when generating new candidates for W in Step 3 if it is either the component that was last augmented along this path of the search, or it is to the right of that component. This rule ensures that once a component is augmented, the search cannot "go back" and modify components to its left. In our example, $(\{q_2\}, \{q_2\})$ would not be generated from $(\{q_1\}, \{q_2\})$ in Step 3, because the last component to have been augmented must have been the second one (since all components begin with the empty set).

Finally we describe the full algorithm for candidate generation. We enumerate each qualifier q in topological order according to $\preceq_Q$. For each such qualifier, we enumerate each constructor p in any order and use the procedure described above to generate all the valid non-redundant rules of the form $(p, (s_1, \ldots, s_k), q)$. The set T is initialized to $\emptyset$ at the beginning of this algorithm and is augmented throughout the entire process. In this way, candidates shown to be valid for some qualifier q can be used to find a later candidate for a target q' to be redundant. For example, a rule allowing the sum of two `pos` expressions to be considered `pos` will be found to subsume a rule allowing the sum of two `pos` expressions to be considered `nonzero`. When this algorithm completes, the set T will contain all valid rules such that none is subsumed by any other valid rule according to $\preceq$. Finally, we augment T with rules that reflect the specificity relation among qualifiers, such as the second `case` rule in Figure 1. These rules are derived directly from the computed $\preceq_Q$ relation.

References

[1] Brian Chin, Shane Markstrum, Todd Millstein, and Jens Palsberg. Inference of user-defined type qualifiers and qualifier rules. In Peter Sestoft, editor, *Programming Languages and Systems, ESOP 2006*, Lecture Notes in Computer Science. Springer-Verlag, 2006.