

Properties of a Rewrite System for Unification with Expansion Variables^{*}

Adam Bakewell and Assaf J. Kfoury

Boston University, <http://types.bu.edu/>

Abstract. A study of properties of a rewrite system for solving constraint sets that are instances of the unification with expansion variables problem. The terms in the constraints are built from the intersection type constructors plus type variables and applied expansion variables. We show that:

- Constraint set rewriting is *confluent* (modulo *isomorphism*).
- There is a set of *well-named* constraint sets which is closed under reduction and normalisation of well-named sets always produces a unifier.
- The rewrite system partitions naturally such that all reductions of well-named constraint sets are finite (terminating) with each part.
- The subset of *acyclic* well-named constraint sets is preserved by reduction the *occur check* may be omitted for reduction of such sets.
- The acyclic well-named constraint sets include those necessary for intersection typing inference, therefore the above properties hold for its application in the intersection type framework of System E.

1 Summary

We study a rewrite system for solving instances of the *unification with expansion variables* (unification with E-variables) problem that we define in [BK04]. We apply the rewrite system to the System E exact typing inference problem in [BCKW04].

An *expansion* is a special term that generalises from the idea of a *substitution*: it is a tree of constructors whose leaves may be constants or substitutions. Expansions may be applied to create a tree of differently substituted instances of the argument. An instance of the unification problem is a set of constraints. A constraint is a pair of *types*.

The key novelty of unification with E-variables, compared to first-order unification, is that in addition to constructors and term variables, types (owing to our main application, the constructors we use are the intersection type constructors, and terms are known as types) may contain applications of *expansion variables*. Substitutions map type variables to types and expansion variables to expansions.

Section 2 recalls the definition of the unification problem and the rewrite system.

^{*} Work partly funded by NSF grant CCR-0113193 *Implementing Modular Program Analysis via Intersection and Union Types*.

Section 3 adapts the method described by Barendregt in [Bar84] to show that rewriting of constraint sets is *confluent* modulo *isomorphism*. This involves extending reduction to allow steps that apply an isomorphism and defining reduction on *marked* constraints that is locally confluent and terminating.

Section 4 defines the set of *well-named* constraint sets and proves that it is closed under reduction and proves that all well-named unsolved constraints are reducible.

The reduction system naturally divides the three parts that solve constraints by: substituting for type variables (T-rewriting); substituting for expansion variables (E-rewriting); substituting the constant ω to eliminate insoluble constraints (Z-rewriting) Section 5 shows that T-rewriting and/or Z-rewriting reduction is strongly normalising, and E-rewriting of well-named constraint sets is strongly normalising (of course, the normal forms produced by these divisions are not always solved).

Section 6 introduces a further restriction on constraint sets inspired by the invariant maintained by the System I unification procedure [KW04]. The reduction rule occur checks, and Z-rewriting, can be omitted for these constraint sets. The main element is the guarantee that the variables (in our setting, the *outer* variables) in the two sides of each constraint are disjoint.

Section 7 shows that the constraints involved in the System E typing inference algorithm, which uses the rewrite system, are acyclic well-named, and therefore inherit all the properties discussed here.

2 Problem Formulation

Section 2.1 defines the main syntactic elements: types, expansions, and substitutions. Section 2.2 defines the semantics of expansions. Section 2.3 defines constraints and unifiers. Section 2.4 defines constraint simplification. Section 2.5 defines partial unifier generation for constraints. Section 2.6 defines the rewrite system for constraint solving.

2.1 Syntax

Definition 1 (Variables, E-Variable, T-Variable). *Expansion variables (E-variables) and type variables (T-variables) are disjoint sets whose elements are indexed by elements of two arbitrary countable sets of indices, \mathbb{I} and \mathbb{J} .*

$$\text{E-Variable} = \{ e_i \mid i \in \mathbb{I} \} \quad \text{T-Variable} = \{ a_i \mid i \in \mathbb{J} \}$$

Metavariable e ranges over E-variables, and metavariable α ranges over T-variables. Let metavariable v range over $\text{Variable} = \text{E-Variable} \cup \text{T-Variable}$. \square

Definition 2 (Types, Type, Type^\rightarrow). *Restricted types and types are defined simultaneously as the least sets satisfying:*

$$\begin{aligned} \text{Type}^\rightarrow &\supseteq \text{T-Variable} \cup \{ \tau_1 \rightarrow \tau_2 \mid \tau_i \in \text{Type} \} \\ \text{Type} &\supseteq \{ \omega \} \cup \text{Type}^\rightarrow \cup \{ \tau_1 \cap \tau_2 \mid \tau_i \in \text{Type} \} \cup \{ e\tau \mid e \in \text{E-Variable} \text{ and } \tau \in \text{Type} \} \end{aligned}$$

Metavariable $\bar{\tau}$ ranges over $\text{Type}^{\rightarrow}$ and metavariable τ ranges over Type . The $\text{Type}^{\rightarrow}$ types have constructors of the simply-typed λ -calculus outermost. The Type types add the binary intersection type constructor \cap , E-variables and the type constant ω .

\rightarrow and \cap associate to the right and that \cap binds more tightly than \rightarrow , i.e., $\alpha \cap \alpha \rightarrow \alpha \cap \alpha$ means $(\alpha \cap \alpha) \rightarrow (\alpha \cap \alpha)$. \square

Note 1 In this paper, the constructor \cap is not commutative, associative or idempotent and it does not absorb ω . These laws, and \cap and ω -distribution laws for E-variables, are allowed in System E [CPWK04] where the enlarged equality classes they induce are a benefit. Adding these laws would not change our results.

Definition 3 (Substitutions and expansions, Substitution, Expansion). Substitutions and expansions are defined simultaneously as the least sets satisfying¹

$$\begin{aligned} \text{Substitution} &\supseteq \{\square\} \cup \{S: \text{Variable} \rightarrow (\text{Type} \cup \text{Expansion}) \mid S(\alpha) \in \text{Type} \text{ and } S(e) \in \text{Expansion}\} \\ \text{Expansion} &\supseteq \{\omega\} \cup \text{Substitution} \cup \{e E \mid E \in \text{Expansion}\} \cup \{E_1 \cap E_2 \mid E_i \in \text{Expansion}\} \end{aligned}$$

The \square constant is a total function from Variable to $\text{Type} \cup \text{Expansion}$, recursively defined as follows:

$$\square = \{\alpha \mapsto \alpha \mid \alpha \in \text{T-Variable}\} \cup \{e \mapsto e \square \mid e \in \text{E-Variable}\}$$

The expansion semantics in Definition 5 lift \square to the identity function on Type . Thus substitutions are sort-preserving total functions from Variable to $\text{Type} \cup \text{Expansion}$; expansions are formal expressions built from binary constructor \cap , unary E-variables, and substitutions or ω at the leaves. \square

Definition 4 (Substitution support, support : Substitution \rightarrow Variable). The support of substitution S is:

$$\text{support}(S) = \{\alpha \in \text{T-Variable} \mid S(\alpha) \neq \alpha\} \cup \{e \in \text{E-Variable} \mid S(e) \neq e \square\} \quad \square$$

Remark 1. Substitutions may be defined by enumerating their support, e.g., $\{\mathbf{a}_1 \mapsto \mathbf{a}_2\}$ is equal to \square on all variables apart from \mathbf{a}_1 . \square

2.2 Expansion

Note 2 Substitutions S and expansions E can be applied to any type τ or expansion E_1 , using the following square bracket operator. Application of expansion E to type τ (resp., expansion E_1), written $[E]\tau$ (resp., $[E]E_1$), returns a type (resp., an expansion).

¹ If A and B are arbitrary sets, $f : A \rightarrow B$ denotes a total function f from A to B .

Definition 5 (Expansion application, $[\cdot]$). By induction on the applied expansion and its argument:

<p><i>Applying substitutions to types:</i></p> $[S] \alpha = S(\alpha)$ $[S] (\tau_1 \rightarrow \tau_2) = [S] \tau_1 \rightarrow [S] \tau_2$ $[S] \omega = \omega$ $[S] (e \tau) = [S(e)] \tau$ $[S] (\tau_1 \cap \tau_2) = [S] \tau_1 \cap [S] \tau_2$ <p><i>Applying expansions to types:</i></p> $[\omega] \tau = \omega$ $[e E] \tau = e ([E] \tau)$ $[E_1 \cap E_2] \tau = [E_1] \tau \cap [E_2] \tau$	<p><i>Applying substitutions to expansions:</i></p> $[S] S_1 = \{v \mapsto [S] (S_1(v)) \mid v \in \text{Variable}\}$ $[S] \omega = \omega$ $[S] (e E) = [S(e)] E$ $[S] (E_1 \cap E_2) = [S] E_1 \cap [S] E_2$ <p><i>Applying expansions to expansions:</i></p> $[\omega] E = \omega$ $[e E_1] E = e ([E_1] E)$ $[E_1 \cap E_2] E = [E_1] E \cap [E_2] E$
--	---

□

Note 3 Applied substitutions distribute down to the outermost variables of their argument then get applied; expansions distribute their argument down into their leaves and then substitute or annihilate them.

Proposition 1 (\square lifts to the identity function). For all types τ and expansions E :

1. $[\square] \tau = \tau$,
2. $[\square] E = E$, and
3. $[E] \square = E$.

□

Notation 1 For expansions E_1 and E_2 , let $(E_1; E_2)$ be shorthand for the composition $[E_2] E_1$ — i.e. E_1 then E_2 . □

Note 4 Expansion application, and the “;” operator, are associative.

2.3 Constraints and Unifiers

Note 5 An instance of the unification with E -variables problem is a set of constraints, and a unifier is a solution.

Definition 6 (Constraints and constraint sets, Constraint, CstrSet). A constraint $\tau_1 \doteq \tau_2$ is an unordered pair of types (i.e. \doteq is commutative.) The set Constraint comprises all constraints and CstrSet all sets of constraints.²

Metavariables $\bar{\Delta}$ and Δ , possibly decorated, range over constraints and constraint sets, respectively. □

Definition 7 (Constraint expansion).

² $\mathbb{P}(S)$ denotes the powerset of set S .

- If $\bar{\Delta}$ is the constraint $\tau_1 \doteq \tau_2$ and E an expansion, then $[E] \bar{\Delta}$ denotes the constraint $[E] \tau_1 \doteq [E] \tau_2$.
- If Δ is a constraint set, then $[E] \Delta$ denotes the constraint set $\{[E] \bar{\Delta} \mid \bar{\Delta} \in \Delta\}$. \square

Note 6 The predicate *solved* is defined on constraints and sets as follows.

Definition 8 (Solved constraint set, solved : CstrSet \rightarrow Boolean).

- $\text{solved}(\tau_1 \doteq \tau_2)$ iff $\tau_1 = \tau_2$;
- $\text{solved}(\Delta)$ iff $\text{solved}(\bar{\Delta})$ for every $\bar{\Delta} \in \Delta$. \square

Note 7 Any expansion E that solves a constraint Δ is a unifier of Δ .

Definition 9 (Unifiers). Expansion E is a unifier of constraint set Δ iff $\text{solved}([E] \Delta)$. \square

Note 8 Expansion constants (ω in the present setting) are trivial unifiers of all constraints.

2.4 Simplification Rules

Note 9 The remainder of this section defines the rewrite system for unification studied in this report, beginning with some helpful notation.

Notation 2

1. Let \vec{e} be a metavariable ranging over the set of all finite sequences of E -variables, including the empty sequence ε .
2. A constraint of the form $\vec{e} \tau_1 \doteq \vec{e} \tau_2$, where τ_1 and τ_2 do not have applications of the same E -variable outermost, may be written $\vec{e}(\tau_1 \doteq \tau_2)$. Call \vec{e} the prefix of such a constraint.
3. Let the notation e/S be shorthand for the substitution $\{e \mapsto eS\}$. Think of such a substitution as acting under e , or in namespace e .
4. We extend the “/” notation to arbitrary sequences of E -variables: $\vec{e} \cdot e/S$ means $\vec{e}/(e/S)$ and ε/S means S . \square

Note 10 The factor eliminates common structure from constraint sets. The soundness property (Proposition 2) is proved in [BK04].

Definition 10 (Constraint factorisation, factor : CstrSet \rightarrow CstrSet). By induction on constraints:

$$\text{factor}(\{\bar{\Delta}_1, \dots, \bar{\Delta}_n\}) = \text{factor}(\bar{\Delta}_1) \cup \dots \cup \text{factor}(\bar{\Delta}_n)$$

$$\text{factor}(\bar{\Delta}) = \begin{cases} \text{factor}(\{\vec{e}(\tau_1 \doteq \tau_2), \vec{e}(\tau'_1 \doteq \tau'_2)\}) & \text{if } \bar{\Delta} = \vec{e}(\tau_1 \cap \tau'_1 \doteq \tau_2 \cap \tau'_2) \\ \text{factor}(\{\vec{e}(\tau_1 \doteq \tau_2), \vec{e}(\tau'_1 \doteq \tau'_2)\}) & \text{if } \bar{\Delta} = \vec{e}(\tau_1 \rightarrow \tau'_1 \doteq \tau_2 \rightarrow \tau'_2) \\ \{\bar{\Delta}\} & \text{otherwise.} \end{cases}$$

\square

Proposition 2 (Soundness). $\text{solved}([E] \Delta)$ iff $\text{solved}([E] (\text{factor}(\Delta)))$. \square

2.5 Unifier Generation Rules

Note 11 *The runifier relation introduced in this section uses the following concepts. Renamings are substitutions that replace variables with variables.*

Definition 11 (Renamings, Renaming). *Let \mathcal{V} be a set of E-variables and \mathcal{S} a set of substitutions. We define $\mathcal{V} \cdot \mathcal{S}$ as follows: $\mathcal{V} \cdot \mathcal{S} = \{e.S \mid e \in \mathcal{V} \text{ and } S \in \mathcal{S}\}$. The set $\mathcal{V} \cdot \mathcal{S}$ is a subset of Expansion. We define the set of renamings as the least such that:*

$$\text{Renaming} \supseteq \{\square\} \cup \{R : \text{Variable} \rightarrow (\text{T-Variable} \cup \text{E-Variable} \cdot \text{Renaming}) \mid R(\alpha) \in \text{T-Variable} \text{ and } R(e) \in \text{E-Variable} \cdot \text{Renaming}\}$$

Thus $\text{Renaming} \subseteq \text{Substitution}$. □

Note 12 *The following special case of renamings act non-trivially only in the outer namespace, and they replace distinct variables in some set \mathcal{V} with distinct variables not in \mathcal{V} .*

Definition 12 (Injective outer renaming for variables). *Substitution $S \in \text{Renaming}$ is an injective outer renaming for variables $\mathcal{V} \subseteq \text{Variable}$ iff:*

- for all T-variables $\alpha, \alpha_1 \in V$ where $\alpha \neq \alpha_1$, $S(\alpha) \notin \mathcal{V}$, and $S(\alpha) \neq S(\alpha_1)$;
- for all E-variables $e, e_1 \in \mathcal{V}$ where $e \neq e_1$ there is an e' such that $S(e) = e' \square$, and $e' \notin \mathcal{V}$, and $S(e) \neq S(e_1)$. □

Note 13 *The vars function gives the set of all variables in its argument.*

Definition 13 (Variables in, vars). *By induction on types and expansions:*

$$\begin{aligned} \text{vars}(\square) &= \emptyset \\ \text{vars}(\alpha) &= \{\alpha\} \\ \text{vars}(\tau_1 \rightarrow \tau_2) &= \text{vars}(\tau_1) \cup \text{vars}(\tau_2) \\ \text{vars}(\omega) &= \emptyset \\ \text{vars}(e \tau) &= \{e\} \cup \text{vars}(\tau) \\ \text{vars}(\tau_1 \cap \tau_2) &= \text{vars}(\tau_1) \cup \text{vars}(\tau_2) \end{aligned}$$

For constraints: $\text{vars}(\tau_1 \doteq \tau_2) = \text{vars}(\tau_1) \cup \text{vars}(\tau_2)$.

For constraint sets: $\text{vars}(\Delta) = \bigcup_{\bar{\Delta} \in \Delta} \text{vars}(\bar{\Delta})$. □

Note 14 *The ovs function gives the set of all variables in the empty namespace of its argument type.*

Definition 14 (Outer variables, ovs : (Type \cup Expansion) $\rightarrow \mathbb{P}(\text{Variable})$). *By induction:*

$$\begin{aligned} \text{ovs}(S) &= \emptyset \\ \text{ovs}(\alpha) &= \{\alpha\} \\ \text{ovs}(\tau_1 \rightarrow \tau_2) &= \text{ovs}(\tau_1) \cup \text{ovs}(\tau_2) \\ \text{ovs}(\omega) &= \emptyset \\ \text{ovs}(e \tau) &= \{e\} \\ \text{ovs}(\tau_1 \cap \tau_2) &= \text{ovs}(\tau_1) \cup \text{ovs}(\tau_2) \end{aligned}$$

□

Note 15 The topmost expansion of τ for variables \mathcal{V} is the largest tree of expansion constructors from the root of τ down to where non-expansion constructors are encountered, with a distinct renaming of the variables in \mathcal{V} at each leaf.

Definition 15 (Topmost expansion extraction, $\text{retract} \subseteq \text{Type} \times \mathbb{P}(\text{Variable}) \times \text{Expansion} \times \mathbb{P}(\text{Variable})$). By induction on types, for an arbitrary finite set V of variables:

$$\begin{aligned} (\bar{\tau}, \mathcal{V}) & \xrightarrow{\text{retract}} (S, \bigcup_{v \in \mathcal{V}} \text{vars}(S(v))) \text{ if } S \text{ is an injective outer renaming for } \mathcal{V} \\ (\omega, \mathcal{V}) & \xrightarrow{\text{retract}} (\omega, \emptyset) \\ (\tau_1 \cap \tau_2, \mathcal{V}) & \xrightarrow{\text{retract}} (E_1 \cap E_2, \mathcal{V}_1 \cup \mathcal{V}_2) \text{ if } (\tau_i, \mathcal{V}) \xrightarrow{\text{retract}} (E_i, \mathcal{V}_i) \text{ and } \mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset \\ (e\tau, \mathcal{V}) & \xrightarrow{\text{retract}} (eE, \mathcal{V}_1) \text{ if } (\tau, \mathcal{V}) \xrightarrow{\text{retract}} (E, \mathcal{V}_1) \quad \square \end{aligned}$$

Note 16 The constraint outer-part unifier generator `runifier` creates an expansion that acts on the outermost within the common prefix of the constraint.

Definition 16 (Topmost outer part renaming unifier, $\text{runifier} \subseteq \text{Constraint} \times \mathbb{P}(\text{Variable}) \times \text{Substitution}$). By case analysis:

$$(\vec{e}(\tau_1 \doteq \tau_2), \mathcal{V}) \xrightarrow{\text{runifier}} \begin{cases} \vec{e}/\{\alpha \mapsto \bar{\tau}\} & \text{if } \{\tau_1, \tau_2\} = \{\alpha, \bar{\tau}\} \text{ and } \alpha \notin \text{ovs}(\bar{\tau}) & \text{(R T-unify)} \\ \vec{e}/\{e \mapsto E\} & \text{if } \{\tau_1, \tau_2\} = \{e\bar{\tau}, \tau\} \text{ and } & \text{(R E-unify)} \\ & (\tau, \mathcal{V}) \xrightarrow{\text{retract}} (E, \mathcal{V}') \text{ and } e \notin \text{ovs}(E) \\ \vec{e}/\omega & \text{if: } \{\tau_1, \tau_2\} = \{\bar{\tau}, \omega\}; & \text{(R Z-unify)} \\ \text{or } \{\tau_1, \tau_2\} = \{\omega, \tau_3 \cap \tau_4\}; & \\ \text{or } \{\tau_1, \tau_2\} = \{\bar{\tau}, \tau_3 \cap \tau_4\}; & \\ \text{or } \{\tau_1, \tau_2\} = \{e\bar{\tau}, \tau\} \text{ and } (\tau, \mathcal{V}) \xrightarrow{\text{retract}} (E, \mathcal{V}') \text{ and } e \in \text{ovs}(E); & \\ \text{or } \{\tau_1, \tau_2\} = \{\alpha, \bar{\tau}\} \text{ and } \alpha \in \text{ovs}(\bar{\tau}) & \end{cases} \quad \square$$

Note 17

- (R T-unify) substitutes for the T-variable, as in first-order unification. But only when the non- α type is a restricted type. It includes an occur check so that constraints like $\mathbf{a}_1 \doteq \mathbf{a}_1 \rightarrow \mathbf{a}_2$ that could otherwise proliferate under an endless series of (R T-unify) steps just get solved by (R Z-unify) instead.
- (R E-unify) uses the `retract` relation and it is only applicable when E-variable e wraps a restricted type. Again, the occur check prevents proliferation of constraints like $e_1 \mathbf{a}_1 \doteq e_1 \mathbf{a}_2 \cap \mathbf{a}_3$, which must be solved by (R Z-unify). This occur check does not prevent (R E-unify) applying to constraints like $e_1 \mathbf{a}_1 \doteq e_1 \mathbf{a}_2 \rightarrow \mathbf{a}_3$, where it will make some progress towards a solution by safely eliminating e_1 .
- (R Z-unify) is applicable where there is an occur check problem, or constructor clash, and so no more “specific” solution exists. Or when a restricted type appears opposite an \cap or ω type. I.e., no unifier is generated for constraints $\tau_1 \doteq \tau_2$ where $\{\tau_1, \tau_2\} = \{e\tau_3, \tau_4\}$ and $\tau_3 \notin \text{Type}^{\rightarrow}$ and $\tau_4 \neq e_1 \bar{\tau}$. Some of

these constraints are soluble by more specific substitutions, some are not.³ It introduces the expansion ω as a “trivial” unifier for such constraints, at the deepest namespace possible so that application of the unifier will eliminate as little of the overall constraint set as necessary.

2.6 Rewrite System for Unification

Note 18 The constraint set direct reduction relation⁴ $\xRightarrow[r]{E}$ takes a constraint set, factors it, picks out any unsolved constraint, uses the runifier relation to generate an outer-part unifier E , applies E to the constraint set and factors again. Factorisation does not eliminate solved parts because it is important to know all the variable names that occur in the whole constraint (otherwise the solution principality property may be lost). Solved parts can be eliminated if the set of variable names in solved parts is retained, but that would complicate our presentation.

Definition 17 (Direct reduction, $\xRightarrow[r]{E} \subseteq \text{CstrSet} \times \text{Expansion} \times \text{CstrSet}$).

$$\Delta \xRightarrow[r]{E} \text{factor}([E] \Delta) \quad \text{if } \bar{\Delta} \in \text{factor}(\Delta) \text{ and not solved}(\bar{\Delta}) \text{ and } (\bar{\Delta}, \text{vars}(\Delta)) \xrightarrow{\text{runifier}} (E, \mathcal{V}).$$

Define $\Delta \xRightarrow[r]{E} \Delta_1$ if there is an expansion E such that $\Delta \xRightarrow[r]{E} \Delta_1$. □

Note 19 A direct reduction relation for reductions that use each of the three runifier rules is also defined. We refer to these three partitions of $\xRightarrow[r]{E}$ as E -rewriting, T -rewriting and Z -rewriting.

Definition 18 (E,T and Z-rewriting systems).

- The E -rewriting direct reduction relation is defined by $\Delta \xRightarrow[r]{E} \text{factor}([E] \Delta)$ if $\vec{e} \bar{\Delta} \in \text{factor}(\Delta)$ and $(\vec{e} \bar{\Delta}, \text{vars}(\Delta)) \xrightarrow{\text{unifier}} E$ and there are e, E' such that $E = \vec{e}/\{e := E'\}$.
- The T -rewriting direct reduction relation is defined by $\Delta \xRightarrow[r]{E} \text{factor}([E] \Delta)$ if $\vec{e} \bar{\Delta} \in \text{factor}(\Delta)$ and $(\vec{e} \bar{\Delta}, \text{vars}(\Delta)) \xrightarrow{\text{unifier}} E$ and there are α, τ such that $E = \vec{e}/\{\alpha := \tau\}$.
- The Z -rewriting direct reduction relation is defined by $\Delta \xRightarrow[r]{E} \text{factor}([E] \Delta)$ if $\vec{e} \bar{\Delta} \in \text{factor}(\Delta)$ and $(\vec{e} \bar{\Delta}, \text{vars}(\Delta)) \xrightarrow{\text{unifier}} E$ and $E = \vec{e}/\omega$. □

³ One justification for giving this less general solution (when the restricted type $\bar{\tau}$ is a T -variable) is that in [BK04] we are able to prove that the resulting rewrite system gives principal unifiers and that it is sufficient for the simulation of β -reduction or intersection typing inference. Another justification is that it enables us to prove confluence in this report.

⁴ The r stands for “renaming” because reextract introduces renamings, or “restricted” because it has principality and completeness properties under certain restrictions [BK04].

Note 20 A constraint set reduction is a sequence of any number of direct reduction steps, including zero when it just simplifies the set. The length of a reduction is the number of direct reduction steps from which it is constructed.

Definition 19 (Reduction, $\xRightarrow{x} \subseteq \text{CstrSet} \times \text{Expansion} \times \text{CstrSet}$). The reduction relation \xRightarrow{x} is the reflexive transitive closure of the direct reduction relation

\xrightarrow{x} for any x . First define \xRightarrow{S} as follows. For all Δ, Δ_1 and Δ_2 :

$$\begin{aligned} \Delta &\xRightarrow{\square}_r \text{factor}(\Delta) \quad \text{and} \\ \Delta &\xRightarrow{E_1; E_2}_r \Delta_2 \quad \text{if there is a } \Delta_1 \text{ such that } \Delta \xRightarrow{E_1}_r \Delta_1 \text{ and } \Delta_1 \xRightarrow{E_2}_r \Delta_2. \end{aligned}$$

Then define $\Delta \xRightarrow{x} \Delta_1$ if there is an expansion E such that $\Delta \xRightarrow{E}_r \Delta_1$. \square

Note 21 In [BK04] the following properties are established.

1. Soundness. For all Δ, Δ_1 and E , $\Delta \xRightarrow{E}_r \Delta_1$ and $\text{solved}(\Delta_1)$ implies $\text{solved}([E] \Delta)$.
2. Incompleteness. There exist Δ and E such that $\text{solved}([E] \Delta)$ and for all Δ_1 , $\text{solved}(\Delta_1)$ implies $\neg(\Delta \xRightarrow{x} \Delta_1)$.
3. Infinite reductions. There exist Δ such that for every n there is a Δ_1 such that there is a reduction $\Delta \xRightarrow{x} \Delta_1$ with length greater than n .
4. Simulation of β -reduction. There is a relation init mapping every term M of the λ -calculus to some Δ such that if M is β -normalising then $\Delta \xRightarrow{x} \Delta_1$ and $\text{solved}(\Delta_1)$, and if M is not β -normalising then Δ has infinite reductions.
5. Restricted completeness. In the ω -free restriction of unification with E -variables, for all Δ in the init relation, and for all E , $\text{solved}([E] \Delta)$ implies there is a Δ_1 such that $\Delta \xRightarrow{x} \Delta_1$ and $\text{solved}(\Delta_1)$.
6. Most general restricted unifiers. If substitution is restricted such that T -variables may only be mapped to restricted types then for all Δ, Δ_1, E and E_1 , $\text{solved}([E] \Delta)$ and $\Delta \xRightarrow{E_1}_r \Delta_1$ implies there is an E_2 such that $[E] \Delta = [E_1; E_2] \Delta$.

3 Confluence

Section 3.1 explains that we can only consider confluence of constraint set reduction modulo variable renaming. Section 3.2 defines marked constraints, their reduction and some key properties. Section 3.3 shows that Z-rewriting is confluent. Section 3.4 puts the confluence result together. The outline of our method is the one developed by Barendregt in [Bar84] to show confluence of β -reduction in the λ -calculus.

3.1 Modulo Isomorphism

Note 22 Confluence defined as $\Delta_1 \xRightarrow{x} \Delta_3 \xleftarrow{x} \Delta_2$ whenever $\Delta_1 \xleftarrow{x} \Delta \xRightarrow{x} \Delta_2$ is obviously untrue as both T -rewriting of constraints like $\mathbf{a}_1 \doteq \mathbf{a}_2$, and E -rewriting of constraints like $\mathbf{e}_1 \mathbf{a}_1 \doteq \mathbf{e}_2 \mathbf{a}_1$, can result in differently named reduced constraint sets. Therefore we will prove confluence modulo isomorphism.

Definition 20 (Isomorphism). A renaming I is an isomorphism if there is a renaming I' such that: $I; I' = \square$, and $I'; I = \square$. If $\tau = [I] \tau'$ (resp., $\Delta = [I] \Delta'$) for some isomorphism I , then τ and τ' (resp., Δ and Δ') are isomorphic. \square

Definition 21 (Reduction modulo isomorphism, \Rightarrow). The definition of the reduction relation \Rightarrow_x is extended by the following rule.

$$\Delta \xRightarrow[x]{I} \text{simplify}([I] \Delta) \quad \text{if } \Delta \text{ and } [I] \Delta \text{ are isomorphic}$$

\square

3.2 Marked Reduction

Note 23 The proof uses marked variables to keep track of the different developments of a constraint as it reduces. When a variable is “used” as the subject of unifier it loses its mark.

Definition 22 (Marked constraints, $\#$).

$$\begin{aligned} \alpha^\# \in \text{E-Variable}^\# &::= \{ e_i^\# \mid i \in \mathbb{I} \} \\ e^\# \in \text{T-Variable}^\# &::= \{ a_i^\# \mid i \in \mathbb{J} \} \\ \bar{\Delta}^\# \in \text{Constraint}^\# &::= \bar{e}(\alpha^\# \dot{\doteq} \bar{\tau}) \mid \bar{e}(e^\# \bar{\tau} \dot{\doteq} \tau) \end{aligned}$$

Let $\Delta^\#$ range over sets containing any number of marked or unmarked constraints. \square

Definition 23 (Marked substitution). Extend the definition of expansion application to marked variables by:

$$[S](e^\# \tau) = \begin{cases} e^\# \tau & \text{if } S(e) = e \square \\ [S(e)] \tau & \text{otherwise} \end{cases} \quad [S] \alpha^\# = \begin{cases} S(\alpha) & \text{if } S(\alpha) = \alpha \\ S(\alpha) & \text{otherwise} \end{cases}$$

\square

Note 24 Non-trivial substitution for a marked variable eliminates the mark; unmarking eliminates all marks without making any other changes.

Definition 24 (Unmarking, $\#$).

$$\begin{aligned} \#(\Delta \cup \{\bar{\Delta}_i^\#\}_{i=1}^n) &= \Delta \cup \{\#(\bar{\Delta}_i^\#)\}_{i=1}^n \\ \#(\bar{e}(\alpha^\# \dot{\doteq} \bar{\tau})) &= \bar{e}(\alpha \dot{\doteq} \bar{\tau}) \\ \#(\bar{e}(e^\# \bar{\tau} \dot{\doteq} \tau)) &= \bar{e}(e \bar{\tau} \dot{\doteq} \tau) \end{aligned}$$

\square

Note 25 A marked direct reduction picks a marked constraint from a set of marked and unmarked constraints, unmarks and uses it to generate a partial unifier in exactly the same way that \xRightarrow{r} does. The definition of marked constraints means that all marked reductions use either E -rewriting or T -rewriting. If application of the expansion solves a constraint without removing its mark then factorisation will remove the mark.

Definition 25 (Marked reduction, $\xRightarrow{\#}, \xRightarrow{\#}$). By analogy with \xRightarrow{r} :

- Extend the definition of factor such that: $\text{factor}(\bar{\Delta}^\#) = \begin{cases} \#(\bar{\Delta}^\#) & \text{if solved}(\#(\bar{\Delta}^\#)) \\ \bar{\Delta}^\# & \text{otherwise} \end{cases}$
- $\Delta^\# \uplus \{\bar{\Delta}^\#\} \xrightarrow{\#} \text{factor}([E](\Delta^\# \cup \{\#(\#)\}))$ if $(\#(\bar{\Delta}), \text{vars}(\Delta^\#)) \xrightarrow{\text{unifier}} E$ \square

Note 26 Marked normalisations are marked reductions resulting in a constraint set empty of marked constraints. A mixed reduction may combine marked and plain reductions.

Definition 26 (Marked normalisation, $\xRightarrow{\mu\#}$). The marked normalisation relation is defined by: $\Delta_1^\# \xRightarrow{\mu\#} \Delta_2^\#$ if $\Delta_1^\# \xRightarrow{\#} \Delta_2^\#$, and $\#(\Delta_2^\#) = \Delta_2^\#$. \square

Convention 3 (Mixed reduction) Let $\Delta^\# \xrightarrow[r,\#]{E} \Delta_1^\#$ if: $\Delta^\# \xrightarrow{\#} \Delta_1^\#$; or $\Delta \subseteq \Delta^\#$ and $\Delta \xrightarrow{\#} \Delta_1$; and $\Delta_1^\# = \text{factor}([E]\Delta^\#)$.

Define $\xRightarrow{z,\#}$ similarly. \square

Note 27 After any marking, there is a mixed reduction corresponding to (i.e., simulating; producing the same result modulo marking) any plain reduction; similarly, after any unmarking, there is a plain reduction corresponding to any mixed reduction.

Lemma 1 ($\xRightarrow{r,\#}$ simulates \xRightarrow{r}). Cf. Lemma 11.1.6.i in [Bar84].

$$\begin{array}{ccc} \Delta_1 & \xrightarrow[r]{E} & \Delta_2 \\ \# \uparrow & & \# \uparrow \\ \Delta_3 & \xrightarrow[r,\#]{E} & \Delta_4 \end{array}$$

Proof. By a diagram chase where we show the following.

$$\begin{array}{ccc} \Delta_1 & \xrightarrow[r]{E} & \Delta_2 \\ \# \uparrow & & \# \uparrow \\ \Delta_3 & \xrightarrow[r,\#]{E} & \Delta_4 \end{array}$$

Unmarking and the application of any expansion E commute: $[E]\#(\tau_1 \stackrel{x}{=} \tau_2)$ and $\#([E](\tau_1 \stackrel{x}{=} \tau_2))$ equal $[E]\tau_1 \doteq [E]\tau_2$ where $\stackrel{x}{=}$ is \doteq or $\stackrel{\#}{=}$. This implies the lemma when Δ_1 and Δ_2 are isomorphic.

If the upper direct reduction is $\Delta_1 \xrightarrow[r]{E} \Delta_2$, then $\bar{\Delta} \in \Delta_1$ and $(\bar{\Delta}, \text{vars}(\Delta_1)) \xrightarrow{\text{unifier}} E$ and there is a $\bar{\Delta}'$ s.t. $\#(\bar{\Delta}') = \bar{\Delta}$ and $\text{vars}(\Delta_3) = \text{vars}(\Delta_1)$. Therefore $(\bar{\Delta}', \text{vars}(\Delta_3)) \xrightarrow{\text{unifier}} E$. Simplification and unmarking commute, implying the lemma. \square

Lemma 2 ($\xrightarrow[r]{E}$ simulates $\xrightarrow[r, \#]{E}$). Cf. Lemma 11.1.6.(ii?) in [Bar84].

$$\begin{array}{ccc} \Delta'_1 & \xrightarrow[r, \#]{E} & \Delta'_2 \\ \# \downarrow & & \# \downarrow \\ \Delta_1 & \xrightarrow[r]{E} & \Delta_2 \end{array}$$

Proof. By a diagram chase where we show the following.

$$\begin{array}{ccc} \Delta'_1 & \xrightarrow[r, \#]{E} & \Delta'_2 \\ \# \downarrow & & \# \downarrow \\ \Delta_1 & \xrightarrow[r]{E} & \Delta_2 \end{array}$$

The details are similar to Lemma 1. \square

Lemma 3 (Unmarking and r reduction simulate $\#$ normalisation). Cf. Lemma 11.1.8 in [Bar84].

$$\begin{array}{ccc} & & \Delta_1 \\ & \swarrow^{E, r} & \uparrow \# \\ \Delta_2 & \xleftarrow[\mu(\#, z)]{E} & \Delta_2^\# \end{array}$$

Proof. By induction on the bottom reduction.

- Case $\Delta_2^\# \xrightarrow[\#]{I} \Delta_2$ where $\Delta_2^\#$ and Δ_2 are isomorphic. Then $\Delta_1 = \Delta_2^\#$ and $\Delta_1 \xrightarrow[r]{I} \Delta_2$.
- Case $\Delta_2^\# \xrightarrow[\#]{E_1} \Delta_3^\# \xrightarrow[\mu\#]{E_2} \Delta_2$ where $\bar{\Delta}^\# \in \Delta_2^\#$, and $(\#(\bar{\Delta}^\#), \text{vars}(\Delta_2^\#)) \xrightarrow{\text{unifier}} E_1$; or case $\Delta_2^\# \xrightarrow[z]{E_1} \Delta_3^\# \xrightarrow[\mu\#]{E_2} \Delta_2$ where $\bar{\Delta} \in \Delta_2^\#$, and $(\bar{\Delta}, \text{vars}(\Delta_2^\#)) \xrightarrow{\text{unifier}} E_1$.
Then $\#(\bar{\Delta}^\#) \in \Delta_2^\#$ and $(\#(\bar{\Delta}^\#), \text{vars}(\Delta_1)) \xrightarrow{\text{unifier}} E_1$. As in Lemma 1, $\#(\text{simplify}([E_1] \Delta_2^\#)) = \Delta_3$ where $\Delta_3 = \text{factor}([E_1] \#(\Delta_2^\#))$, so $\#(\Delta_3^\#) = \Delta_3$ and by I.H., $\Delta_3 \xrightarrow[r]{E_2} \Delta_2$. \square

3.3 Confluence of Z-Rewriting

Note 28 Every Z-rewriting reduction is finite because Z-rewriting reduces constraint set size, so we can show that Z-rewriting alone is confluent by Newman's lemma.

Definition 27 (Size measure, size).

$$\begin{aligned}
\text{size}(\Delta) &= \Sigma\{ \text{size}(\tau_1) + \text{size}(\tau_2) \mid \tau_1 \dot{=} \tau_2 \in \Delta \text{ and not solved}(\tau_1 \dot{=} \tau_2) \} \\
\text{size}(\omega) &= 1 \\
\text{size}(\alpha) &= 1 \\
\text{size}(e\tau) &= 1 + \text{size}(\tau) \\
\text{size}(\tau_1 \rightarrow \tau_2) &= 1 + \text{size}(\tau_1) + \text{size}(\tau_2) \\
\text{size}(\tau_1 \cap \tau_2) &= 1 + \text{size}(\tau_1) + \text{size}(\tau_2)
\end{aligned}$$

□

Lemma 4 (Confluence of Z-rewriting).

$$\begin{array}{ccc}
\Delta_1^\# & \xrightarrow[\text{Z}]{E_1} & \Delta_2^\# \\
\text{Z} \downarrow E_2 & & \text{Z} \downarrow E_3 \\
\Delta_3^\# & \xrightarrow[\text{Z}]{E_4} & \Delta_4^\#
\end{array}$$

and $E_2; E_4 = E_1; E_3$.

Proof. Every Z-rewriting direct reduction reduces the size measure: $\Delta^\# \xrightarrow{\text{Z}} \Delta_5^\#$ implies $\text{size}(\Delta^\#) > \text{size}(\Delta_5^\#)$. Z-rewriting is locally confluent by critical pair analysis:

If $E_1 = \bar{e}_1/\omega$ and $E_2 = \bar{e}_1 \bar{e}_2/\omega$ then $E_4 = E_1$ and $E_3 = \square$. Similarly when E_1 is within E_2 .

If $E_1 = \bar{e}_1 e_1 \bar{e}_2/\omega$ If $E_2 = \bar{e}_1 e_2 \bar{e}_3/\omega$ and $e_1 \neq e_2$ then $E_4 = E_1$ and $E_3 = E_2$.

The result follows by Newman's lemma. □

Lemma 5 (Termination of marked reduction). *For every $\Delta^\#$ there is an n such that the length of every reduction $\Delta^\# \xrightarrow[\#]{\Rightarrow} \Delta_1^\#$ is less than n .*

Proof. By the multiset ordering: let the measure of a constraint set be a multiset containing a numeric measure for each marked constraint. The measure of a marked constraint $\bar{\Delta}^\#$ is the number of other marked constraints (in shallower prefixes) whose elimination could expand $\bar{\Delta}^\#$. □

3.4 Confluence Result

Note 29 *The confluence theorem is proved by a strip lemma which in is proved by a marked strip lemma which in turn is proved using the following commutation property.*

Lemma 6 (Reduction commutation lemma).

$$\begin{array}{ccc}
\Delta_1^\# & \xrightarrow[\text{r}]{E_1} & \Delta_2^\# \\
\mu\# \downarrow E_2 & & \mu(\text{Z}, \#) \downarrow E_3 \\
\Delta_3^\# & \xrightarrow[\text{r}]{E_4} & \Delta_4^\#
\end{array}$$

and $E_2; E_4; E_5 = E_1; E_3$.

Proof. 1. Bring the marked reductions to the front. This is always possible because marked constraints can only be introduced by expanding or factoring pre-existing marked constraints. 2. The Z-rewriting steps that are not dependent on unmarked T-rewrite or E-rewrite steps can be brought to the front where they form a normalising Z-reduction. \square

Lemma 7 (Marked strip lemma). *Cf. Lemma 11.1.7.ii in [Bar84].*

$$\begin{array}{ccc}
 \Delta_1^\# & \xrightarrow{E_1} & \Delta_2^\# \\
 \downarrow \text{z, \#} E_2 & \text{r, \#} & \downarrow \mu(\text{z, \#}) E_3 \\
 \Delta_3 & \xrightarrow[\text{r}]{E_4} & \Delta_4
 \end{array}$$

and $E_2; E_4 = E_1; E_3$.

Proof. By induction on the top reduction using Lemma 6, $\Delta_1^\# \xrightarrow[\mu\#\#]{E_5} \Delta_5 \xrightarrow[\mu\text{z}]{E_6} \Delta_7 \xrightarrow[\text{r}]{E_7} \Delta_4$ where $E_1; E_3 = E_5; E_6; E_7$. If $\Delta_1^\# \xrightarrow[\#\#]{E_1} \Delta_3$ then there is one marked constraint in $\Delta_1^\#$ and therefore there is an isomorphism I mapping Δ_3 to Δ_5 and $E_4 = I; E_5; E_6$. Otherwise, $\Delta_1^\# \xrightarrow[\text{z}]{E_1} \Delta_3$ and $E_5 = \square$; and by Lemma 4 there is an E_8 such that $\Delta_3 \xrightarrow[\text{z}]{E_8} \Delta_7$ and $E_4 = E_8; E_7$. \square

Lemma 8 (Strip lemma). *Cf. Lemma 11.1.9 in [Bar84].*

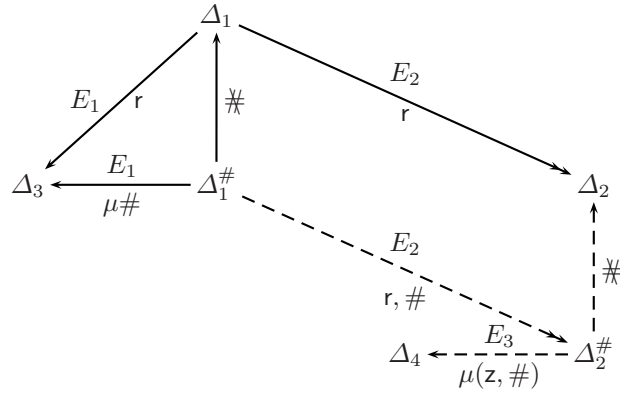
$$\begin{array}{ccc}
 & \Delta_1 & \\
 E_1 \swarrow \text{r} & & \searrow E_2 \text{r} \\
 \Delta_3 & & \Delta_2 \\
 & \xrightarrow[\text{r}]{E_4} & \\
 & \Delta_4 & \\
 & \swarrow E_3 \text{r} & \\
 & \Delta_4 &
 \end{array}$$

and $E_1; E_4 = E_2; E_3$.

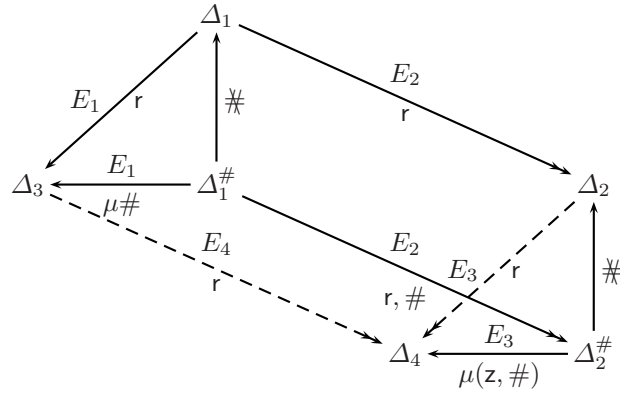
Proof. Consider the top-left direct reduction. For every T-rewriting or E-rewriting direct reduction $\xrightarrow[\text{r}]{E_1}$, there is a $\xrightarrow[\#\#]{E_1}$ reduction from the marking of Δ_1 to the unmarked Δ_3 where the constraint used by runifier to generate E_1 is the only marked constraint; For every Z-rewriting direct reduction, mark no constraints and the exact same reduction exists using Z-reduction.

$$\begin{array}{ccc}
 & \Delta_1 & \\
 E_1 \swarrow \text{r} & & \searrow E_2 \text{r} \\
 \Delta_3 & & \Delta_2 \\
 & \xrightarrow[\text{z, \#}]{E_1} & \\
 & \Delta_1^\# & \\
 & \uparrow \#\# & \\
 & \Delta_1 &
 \end{array}$$

By Lemma 1 there is a mixed reduction that simulates the \Rightarrow_r reduction to Δ_2 ;
 By Lemma 5 $\Delta_2^\#$ has an unmarked normal form using mixed Z-rewriting and marked rewriting:

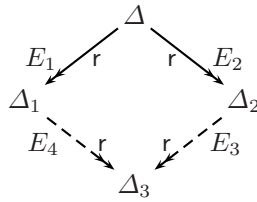


By Lemma 3 there is a reduction corresponding to the normalising mixed reduction to Δ_4 ; Lemma 6 completes the square.



□

Theorem 4 (Confluence modulo isomorphism). Cf. Lemma 11.1.10 in [Bar84].



and $E_1; E_4 = E_2; E_3$.

Proof. From Lemma 8 by a diagram chase.

□

4 Well-named constraints

This section provides a restriction on constraints such that the irreducible constraint forms never arise. Section 4.1 defines a language of constraint shapes based on the idea of charging every type positively or negatively. Section 4.2 adds variable occurrence restrictions to form the definition of a well-named constraint set. Section 4.3 shows that reduction is closed on well-named constraint sets and that unsolved well-named sets are reducible.

4.1 Polar Types and Constraints

Note 30 *One approach to avoiding irreducible constraints is to only permit E-variables to wrap restricted types; that invariant will be preserved by (R E-unify) (and (R T-unify) and (R Z-unify)). However, for the typing inference application it is desirable to allow E-variables to wrap \sqcap . In this case, the type opposite must be either a restricted type or an E-variable that wraps a restricted type. This is the approach we formalise by giving a restricted language of polar (charged) types and constraints.*

Note 31 *The basic idea is to designate the arbitrarily constructed types as negative and the restricted or E-variable-wrapped restricted types as positive and then insist that every constraint must have a positive and a negative side. To guarantee that simplification preserves this regime, one argument of a positive \rightarrow constructor must be positive and the other negative, and for a negative \rightarrow the polarities must be reversed.*

Note 32 *To guarantee that reduction preserves the regime, only positive (resp., negative) types can be substituted for positive (resp., negative) variables; this is handled by imposing variable occurrence restrictions in the next section.*

Note 33 *Charging a type or constraint adds a charge superscript to each variable within its prefix. Charged types and constraints themselves are also considered positive or negative. The following definition specifies the syntax of the polar types and constraints. We also make use of multisets of charged variable vectors.*

Definition 28 (Polar definitions).

$$\begin{array}{ll}
 c \in & \text{Charge} ::= + \mid - & (\text{charges}) \\
 \bar{\rho} \in & \text{PosType}^{\rightarrow} ::= \alpha^+ \mid \sigma \rightarrow \rho & (\text{restricted positively charged types}) \\
 \rho \in & \text{PosType} ::= e^+ \bar{\rho} \mid \bar{\rho} & (\text{positively charged types}) \\
 \sigma \in & \text{NegType} ::= \alpha^- \mid \rho \rightarrow \sigma \mid \omega \mid e^- \sigma \mid \sigma \sqcap \sigma_1 & (\text{negatively charged types}) \\
 \gamma \in & \text{PolarConstraint} ::= \rho \dot{=} \sigma \mid e \gamma & (\text{charged constraints}) \\
 \Gamma \in & \mathbb{P}(\text{PolarConstraint}) & (\text{charged constraint sets})
 \end{array}$$

The discharge operator: $|\Gamma|$ gives a constraint Δ , identical to Γ with no charges. \square

Note 34 A charge c can be positive or negative. For our purposes the benefit of marking variables with one of two charge symbols is that we can impose a condition which checks that no T -variable occurs more than twice, and no E -variable occurs more than once in certain positions (the positions designated positive).⁵ Every polar constraint has a positive and a negative side: negative types are essentially unrestricted apart from the polarity requirement on \rightarrow ; positive types exclude \sqcap , ω , and any more than one wrapping E -variable, they also have a polarity requirement on \rightarrow which complements negative arrows. These restrictions are essential to rule out situations where differing extractions of expansion constructors could lead to non-confluence.

Note 35 The inverse of discharging — charging — is not a function: some types and constraints do not charge (e.g., $\omega \rightarrow \omega$) and some types charge positively and negatively (e.g., \mathfrak{a}). So discharging is total, non-injective and non-surjective.

4.2 Well-Named Constraints

Note 36 The following variable occurrence restrictions are needed to guarantee preservation of the polar shape by reduction.

1. Every T -variable occurs at most twice, once positively and once negatively, in any namespace. Then T -rewriting constraint reduction preserves polarities as any other occurrence of $\vec{e}\alpha$ in an unsolved single constraint belongs to the same sort as the type substituted for it.
2. Every E -variable occurs at most once positively in any namespace. Then E -rewriting constraint reduction preserves polarities because all other occurrences of $\vec{e}e$ are at negative type positions or wrapping a constraint.

Note 37 To formalise these conditions we use the following `vpaths` function to form the polar multiset Ψ of all E -variable paths ψ to relevant variables in a polar constraint set Γ .

Notation 5 (Multiset notation) Enumerated multisets are delimited by angle brackets; the empty multiset is $\langle \rangle$; multiset membership uses the symbol \in ; the multiset union operator is \uplus . The multiset difference operator $\Psi \setminus \Psi_1$ removes an occurrence of each element of Ψ_1 from Ψ , if there is one. Let the sequence constructor \cdot distribute such that $v \cdot \langle \psi_1, \dots, \psi_n \rangle = \langle v \cdot \psi_1, \dots, v \cdot \psi_n \rangle$. \square

Definition 29 (Variable paths, vpaths). By induction on polar types, constraints and constraint sets:

⁵ In other work on intersection typing inference using polar types, such as [Jim00,Car02,KW04], the polarities take on deeper meanings. For instance, the *negative* occurrence of a T -variable refers to its binding occurrence; the *positive* occurrence refers to its free, or “usage”, occurrence.

$$\begin{aligned}
\text{vpaths}(\alpha^c) &= \langle \alpha^c \rangle \\
\text{vpaths}(e^+ \rho) &= e \cdot \text{vpaths}(\rho) \uplus \langle e^+ \rangle \\
\text{vpaths}(e^- \sigma) &= e \cdot \text{vpaths}(\sigma) \\
\text{vpaths}(\rho \rightarrow \sigma) &= \text{vpaths}(\rho) \uplus \text{vpaths}(\sigma) \\
\text{vpaths}(\sigma \rightarrow \rho) &= \text{vpaths}(\sigma) \uplus \text{vpaths}(\rho) \\
\text{vpaths}(\sigma \cap \sigma_1) &= \text{vpaths}(\sigma) \uplus \text{vpaths}(\sigma_1) \\
\text{vpaths}(\omega) &= \langle \rangle \\
\text{vpaths}(e \gamma) &= e \cdot \text{vpaths}(\gamma) \\
\text{vpaths}(\emptyset) &= \langle \rangle \\
\text{vpaths}(\{\gamma\} \cup \Gamma) &= \text{vpaths}(\gamma) \uplus \text{vpaths}(\Gamma)
\end{aligned}$$

□

Definition 30 (Well-named constraint sets, $\text{wellnamed} : \text{CstrSet} \rightarrow \text{Bool}$).
 $\text{wellnamed}(\Delta \cup \Delta_1)$ iff

- Δ_1 is solved: $\text{solved}(\Delta_1)$;
- Δ charges: *there is a Γ such that $|\Gamma| = \Delta$;*
- T-variables occur positively at most once, and negatively at most once, in any namespace: *for all c, \bar{e}, α , and Ψ : $\text{vpaths}(\Gamma) \neq \Psi \uplus \langle \bar{e} \alpha^c, \bar{e} \alpha^c \rangle$;*
- E-variables occur positively at most once, in any namespace: *for all \bar{e}, e , and Ψ : $\text{vpaths}(\Gamma) \neq \Psi \uplus \langle \bar{e} e^+, \bar{e} e^+ \rangle$.* □

4.3 Preservation of Well-Namedness

Note 38 *Reduction of a well-named constraint set with \Rightarrow_r results in a well-named constraint set. Simplification preserves well-namedness (Lemma 10); this is not such a grand statement as only outermost \rightarrow constructors in well-named constraints can be factored. Lemma 9 expresses a property used in the proofs and Definition 31 finds the E-variable path to each substitution in an expansion.*

Lemma 9 (Partitions of well-named constraint sets). $\text{wellnamed}(\Delta)$ and $\Delta = \Delta_1 \uplus \Delta_2$ iff there are Γ_1 and Γ_2 such that $|\Gamma_1| = \Delta_1$ and $|\Gamma_2| = \Delta_2$ and $\text{vpaths}(\Gamma_1) \cap \text{vpaths}(\Gamma_2) = \emptyset$.

Proof. By induction on Δ . □

Lemma 10 (Simplification preserves wellnamed). *If $\text{wellnamed}(\Delta)$ then $\text{wellnamed}(\text{factor}(\Delta))$.*

Proof. By induction on Δ we show that (1) $\text{vpaths}(\text{factor}(\Delta)) \subseteq \text{vpaths}(\Delta)$ and (2) $\text{wellnamed}(\Delta)$.

- Case \emptyset . Then $\text{simplify}(\Delta) = \emptyset$.
- Case $\Delta_1 \cup \{\bar{\Delta}\}$. Proceed by cases on $\bar{\Delta}$:

- Case $\vec{e}(\tau_1 \rightarrow \tau_2 \dot{\vdash} \tau_3 \rightarrow \tau_4)$. Then $\bar{\Delta} = |\gamma|$, and $\gamma = \vec{e}(\sigma_1 \rightarrow \rho_2 \dot{\vdash} \rho_3 \rightarrow \sigma_4)$, and $\text{vpaths}(\gamma) = \text{vpaths}(\sigma_1) \cup \text{vpaths}(\rho_2) \cup \text{vpaths}(\rho_3) \cup \text{vpaths}(\sigma_4)$. Let $\bar{\Delta}_1 = \vec{e}(\tau_1 \dot{\vdash} \tau_3)$, and $\bar{\Delta}_2 = \vec{e}(\tau_2 \dot{\vdash} \tau_4)$, thus $\bar{\Delta}_1$ and $\bar{\Delta}_2$ are well-named, and $\text{vpaths}(\bar{\Delta}_1) \cup \text{vpaths}(\bar{\Delta}_2) \subseteq \bar{\Delta}$ (note that $\bar{\Delta}_1$ or $\bar{\Delta}_2$ may be solved). By I.H., $\text{vpaths}(\text{simplify}(\bar{\Delta}_i)) \subseteq \text{vpaths}(\bar{\Delta}_i)$, and $\text{wellnamed}(\text{simplify}(\bar{\Delta}_i))$. The lemma follows by the transitivity of \subseteq .
- Case $\vec{e}(\tau_1 \cap \tau_2 \dot{\vdash} \tau_3 \rightarrow \tau_4)$ is not possible for chargeable constraints.
- Case otherwise. Then $\text{simplify}(\bar{\Delta}) = \bar{\Delta}$.

By I.H., $\text{vpaths}(\text{simplify}(\Delta_1)) \subseteq \text{vpaths}(\Delta_1)$, so (1) and (2) follow by Lemma 9 and the transitivity of \subseteq . \square

Definition 31 (E-paths of an expansion, $\text{eExtract} : \text{Expansion} \rightarrow \mathbb{P}(\text{E-Variable} \times \text{Substitution})$). *By induction on expansions:*

$$\begin{aligned}
\text{eExtract}(S) &= \{(\epsilon, S)\} \\
\text{eExtract}(\omega) &= \emptyset \\
\text{eExtract}(e E) &= \{(e \cdot \vec{e}, S) \mid (\vec{e}, S) \in \text{eExtract}(E)\} \\
\text{eExtract}(E \cap E_1) &= \text{eExtract}(E) \cup \text{eExtract}(E_1)
\end{aligned}$$

\square

Lemma 11 (Reduction preserves wellnamed). *If wellnamed(Δ) and $\Delta \xrightarrow[\tau]{E} \Delta_1$ then wellnamed(Δ_1).*

Proof. Let $\Delta_2 = \text{factor}(\Delta)$. Then wellnamed(Δ_2) by Lemma 10. By the definition of $\xrightarrow[\tau]{E}$, there are $\vec{e}, \tau_i, E', \Delta_3, \bar{\Delta}$ such that $\bar{\Delta} = \vec{e}(\tau_1 \dot{\vdash} \tau_2)$, and $\{\bar{\Delta}\} \cup \Delta_3 = \Delta_2$, and $(\bar{\Delta}, \text{vars}(\Delta)) \xrightarrow{\text{runifier}} \vec{e}/E'$. Proceed by case analysis of E' .

- Case $\{\alpha \mapsto \tau\}$ (R T-unify) .
Then $\text{vpaths}([E] \Delta_3) \subseteq \text{vpaths}(\Delta_2)$ and wellnamed($[E] \Delta_3$) by structural induction on Δ_3 . $\text{solved}([E] \bar{\Delta})$ implies that $\text{vpaths}([E] \bar{\Delta}) = \langle \rangle$ and wellnamed($[E] \bar{\Delta}$) and therefore wellnamed($[E] \Delta_2$). Then wellnamed(Δ_1) by Lemma 10.
- Case $\{e \mapsto E''\}$ (R E-unify) .
Let $\{\tau_1, \tau_2\} = \{e\bar{\tau}, \tau\}$, and $\Psi = \vec{e} \cdot e \cdot \text{vpaths}(\Delta_2)$. Let $\{(\vec{e}_i, S_i)\}_{i=1}^n = \text{eExtract}(E'')$. By structural induction on Δ_1 , using Lemma 10: $\text{vpaths}([E] \Delta_2) = \text{vpaths}(\Delta_2) \setminus \Psi \uplus \langle \vec{e} \cdot \vec{e}_i \cdot [S_i] \psi \mid \vec{e} \cdot e \cdot \psi \in \Psi \text{ and } 1 \leq i \leq n \rangle$ and wellnamed(Δ_1).
- Case ω (R Z-unify) .
Then $\text{vpaths}([E] \Delta_2) \subseteq \text{vpaths}(\Delta_2)$ and wellnamed($[E] \Delta_2$) by structural induction on Δ_2 . Then wellnamed(Δ_1) by Lemma 10. \square

Note 39 *For well-named constraint sets, reduction enjoys the following progress property. This means that no normalising reduction fails to unify a constraint. But note it is much weaker than a confluence result: we have not yet shown that if a constraint Δ has a normalising reduction then there is no reduction $\Delta \xrightarrow[\tau]{E} \Delta_1$ such that Δ_1 has no normalising reductions; and we have not shown that all the normal forms of Δ are equal.*

Theorem 6 (Progress of reduction). *If $\text{wellnamed}(\Delta)$ and $\text{not solved}(\Delta)$ then there is a Δ_1 such that $\Delta \xRightarrow{\tau} \Delta_1$.*

Proof. By cases on any $\vec{e}(\tau_1 \dot{=} \tau_2) \in \text{simplify}(\Delta)$:

- if $\tau_1 = e\bar{\tau}$ or $\tau_2 = e'\bar{\tau}'$ then Δ is reducible by E-rewriting;
- if $\tau_1 = \alpha$ and $\alpha \notin \text{ovs}(\tau_2)$, or $\tau_2 = \alpha'$ and $\alpha' \notin \text{ovs}(\tau_1)$, then Δ is reducible by T-rewriting;
- if $\tau_1 = \alpha$ and $\alpha \in \text{ovs}(\tau_2)$, or $\tau_2 = \alpha'$ and $\alpha' \in \text{ovs}(\tau_1)$, then Δ is reducible by Z-rewriting;
- if $\tau_1 = \tau_3 \rightarrow \tau_4$ and $\tau_2 = \omega$ or $\tau_2 = \tau_5 \cap \tau_6$, or $\tau_2 = \tau_7 \rightarrow \tau_8$ and $\tau_1 = \omega$ or $\tau_1 = \tau_9 \cap \tau_{10}$, then Δ is reducible by Z-rewriting;
- forms where $\tau_1 = \tau_3 \rightarrow \tau_4$ and $\tau_2 = \tau_5 \rightarrow \tau_6$ do not occur in simplified constraint sets. \square

5 Finite Reductions

Definition 32 (Terminating rewrite system). *Rewrite relation $\xRightarrow{\tau} \subseteq SET \times SET$ is terminating iff for every $mem \in SET$ there is an $n \in \mathbb{N}$ such that the length of every reduction $mem \xRightarrow{\tau} mem'$ is at most n . Where the length of a reduction is the number of direct reductions it comprises.* \square

Note 40 *The $\xRightarrow{\tau}$ system is not terminating (see [BK04] for examples, and the proof it simulates β -reduction of λ -terms and hence has infinite reductions). The rest of this section shows that the following systems, formed by partitioning $\xRightarrow{\tau}$ into the three parts defined by the runifier rules, terminate on well-named constraint sets.*

5.1 T-rewriting and Z-rewriting terminate

Note 41 *T-rewriting and Z-rewriting — separately or together — are terminating on all constraint. For T-rewriting, a very simple linear ordering is sufficient to show termination. The ordered pair of number of distinct T-variables (counting occurrences of the same T-variable in distinct namespaces as distinct) and constraint size is decreased.*

Definition 33 (Distinct-variable and size measure, dvsm).
 $\text{dvsm}(\Delta \cup \Delta_1) = (|\{\psi \in \text{vpaths}(\Delta)\}|, \text{size}(\Delta))$ where $\text{solved}(\Delta_1)$ and $\text{not solved}(\Delta)$. \square

Lemma 12 (Size under simplification). $\text{size}(\Delta) \geq \text{size}(\text{factor}(\Delta))$.

Proof. By structural induction on Δ . \square

Theorem 7 (T-rewriting and Z-rewriting terminate). *The systems $\xRightarrow{\tau}$, \xRightarrow{z} and $\xRightarrow{\tau, z}$ are terminating.*

Proof. If $\Delta \xrightarrow[\tau]{\bar{e}/\{\alpha \mapsto \tau\}} \Delta'$ then $\{\psi \in \text{vpaths}(\Delta')\} \subseteq (\{\psi \in \text{vpaths}(\Delta)\} \setminus \{\bar{e} \cdot \alpha\})$ by structural induction on Δ' .

If $\Delta \xrightarrow[z]{E=\bar{e}/\omega} \Delta'$ then $\{\psi \in \text{vpaths}(\Delta')\} \subseteq \{\psi \in \text{vpaths}(\Delta)\}$ and $\text{size}([E] \Delta) < \text{size}(\Delta)$ by structural induction on Δ' . By Lemma 12, $\text{size}(\Delta) \geq \text{size}([e] \Delta)$. \square

Note 42 *On well-named constraint sets we can use size by itself as the termination order because each T-variable occurs at most twice in any namespace. On arbitrary constraints this is not true, but it is still true that $\text{size}(\Delta)$ bounds the length of reductions of Δ because $|\{\psi \in \text{vpaths}(\Delta)\}|$ is bounded by the size of Δ . T-rewriting does not decrease the even simpler measure of the constraint set cardinality, because this can be increased by simplification.*

5.2 E-rewriting terminates

Note 43 *For E-rewriting the termination proof is harder. E-rewrite steps do not reduce the constraint set size in any useful sense; they do eliminate one E-variable from one namespace, but they can create many more in other namespaces.*

The restriction to well-named constraint sets offers a fairly simple decreasing measure. When an E-variable e is expanded, the “subject constraint” breaks into several shallower constraints; so let some “height” measure be part of the measure.

Elsewhere in the set e may occur either as part of the prefix of some constraints or within it. If within the prefix, the expansion can increase the constraint size and the depth of syntactic nesting. However, we can count blocks of negative E-variables (and \sqcap constructors, but not \rightarrow constructors) as a single unit for the purpose of height counting, because they are treated as a unit by E-rewriting.

If e is in the prefix the expansion can make several copies of the constraints wrapped by e , but it will not change their heights (which could be greater than the height of the directly affected constraint). We can count these constraints as being at a lower “level”, so their proliferation does not increase the overall measure provided that the measure of the remnants of the higher-level directly affected constraint decreases. Thus we use a multiset ordering [DM79] in the proof of Theorem 8.

The level measure is derived from the position of a constraint in the level ordering of the whole constraint set.

Definition 34 (Level ordering, levOrder).

$$\text{levOrder}(\Delta) = \left\{ (\bar{\Delta}_1, \bar{\Delta}_2) \left| \begin{array}{l} \{\bar{\Delta}_1, \bar{\Delta}_2\} \subseteq \Delta \text{ and } \bar{\Delta}_1 = \bar{e}_1 \bar{e}_2 e \bar{\Delta}_3 \text{ and} \\ \bar{\Delta}_2 = \bar{e}_1 (\tau_1 \dot{=} \tau_2) \text{ and} \\ \bar{e}_2 e \text{ is in the variables of } \tau_1 \text{ or } \tau_2 \end{array} \right. \right\}$$

\square

Note 44 As the prefix of $\bar{\Delta}_1$ is always longer than the prefix of $\bar{\Delta}_2$ in the construction of levOrder , the transitive closure of $\text{levOrder}(\Delta)$ is irreflexive.

Intuitively, the level of $\bar{\Delta}$ is roughly the maximum prefix length of all constraints in Δ , minus the prefix length of $\bar{\Delta}$.

Definition 35 (Level and height multiset measure).

$$\text{lhmm}(\Delta) = \{ (l(\bar{\Delta}, \Delta), h(\bar{\Delta})) \mid \bar{\Delta} \in \Delta \}$$

Where $l(\bar{\Delta}_1, \Delta)$ is: 0 if there is no $(\bar{\Delta}_2, \bar{\Delta}_1)$ in $\text{levOrder}(\Delta)$; and $n+1$ if n is the maximum level of any constraint $\bar{\Delta}_2$ s.t. $(\bar{\Delta}_2, \bar{\Delta}_1)$ is in $\text{levOrder}(\Delta)$. And:

$$\begin{aligned} h(\bar{e}(\tau_1 \dot{\simeq} \tau_2)) &= h(\tau_1) + h(\tau_2) \\ h(\alpha) &= 0 \\ h(\tau_1 \rightarrow \tau_2) &= 1 + \max\{h(\tau_1), h(\tau_2)\} \\ h(\tau) &= 1 + h'(\tau) \text{ if } \tau \notin \text{Type}^\rightarrow \\ h'(\bar{\tau}) &= h(\bar{\tau}) \\ h'(\omega) &= 0 \\ h'(e\tau) &= h'(\tau) \\ h'(\tau_1 \cap \tau_2) &= \max\{h'(\tau_1), h'(\tau_2)\} \end{aligned}$$

□

Theorem 8 (E-rewriting of well-named constraint sets is terminating).

The system \xRightarrow{e} is terminating.

Proof. If $\Delta \xRightarrow{S} \Delta'$ and $\Delta = \text{factor}(\Delta)$ then $\text{lhmm}(\Delta) >_m (\Delta')$. Where the ordered pair multiset ordering $>_m$ is defined $X_1 \cup X_2 >_m X_1 \cup X_3$ if X_1 are multisets and $X_2 \neq \emptyset$ and for all $x \in X_3$ there is an $x' \in X_2$ such that $x' > x$. And the ordered pair ordering is $(a, b) > (c, d)$ if $a > c$, or $a = c$ and $b > d$.

Let $\Delta = \Delta_1 \uplus \{\bar{e}(e\bar{\tau} \dot{\simeq} \tau)\}$ and $\Delta' = \text{simplify}([S] \Delta)$ and $S = \bar{e}/e := E$ and $\text{eExtract}(E) = \{(\bar{e}_i, S_i)\}_{i=1}^n$. Partition Δ_1 into three as follows.

1. The e -in-prefix constraints: $\Delta_p = \{\bar{e}e\bar{\Delta}'_i \in \Delta_1\}$
2. The e -below-prefix constraints:
 $\Delta_b = \{\bar{\Delta} \in \Delta_1 \mid \bar{\Delta} = \bar{e}_1(\tau_1 \dot{\simeq} \tau_2) \text{ and } \bar{e}_2 e \in \text{vpaths}(\tau_1 \dot{\simeq} \tau_2) \text{ and } \bar{e}_1 \bar{e}_2 = \bar{e}\}$
3. The rest: $\Delta_r = \Delta_1 \setminus (\Delta_p \setminus \Delta_b)$

Let $M_p = \text{lhmm}(\Delta_p)$, $M_b = \text{lhmm}(\Delta_b)$, $M_r = \text{lhmm}(\Delta_r)$, and $(l, h) = \text{lhmm}(\{\bar{e}(e\bar{\tau} \dot{\simeq} \tau)\})$.

So $\Delta' = \Delta_r \cup \Delta_{p'} \cup \text{simplify}([S] \Delta_b \cup \{\bar{e}(E\bar{\Delta} \dot{\simeq} \tau)\})$
where $\Delta_{p'} = \{\bar{e}\bar{e}_i[S_i]\bar{\Delta} \mid \bar{e}e\bar{\Delta} \in \Delta_p \text{ and } 1 \leq i \leq n\}$ and $\Delta_{b'} = \{[S]\bar{\Delta} \mid \bar{\Delta} \in \Delta_b\}$.
The above definition of Δ' gives $\Delta_{p'}$ in simplified form: each Δ_p constraint becomes n ; M_r is unchanged as it is unaffected by S and Δ is simplified in the statement of this theorem; simplification is applied to the subject constraint and those in $\Delta_{b'}$.

The level order of every constraint is preserved or transmitted to its simplify-descendants. The measures update as follows.

- M_r is unchanged.
- Each e -in-prefix constraint measure is replaced by n copies of itself: $M_{p'} = \bigcup_{i=1}^n M_p$ (this is after simplify). For the height this is obvious: the prefix does not count towards height. For the level, we have seen that each copy can retain the level of the original.
- For the e -below-prefix constraints, including the subject constraint, before simplify the height measure is unchanged or reduced because $h([e := E] e \tau) = 0$, if $E = \omega$; and $h(e \tau)$, otherwise. Simplification either leaves a constraint unchanged or breaks it into m with smaller heights. At least the subject constraint is broken down: (l, h) in $\text{lhmm}(\Delta)$ becomes at most $\{(l, h - 1)\}_{i=1}^n$ in $\text{lhmm}(\Delta')$. \square

Note 45 *There are further interesting questions on termination. In particular, we do not know whether E -rewriting on unrestricted constraints is terminating. We do know that a simpler version of E -rewriting can non-terminate on non-well-named constraint sets. But the example relies on the E -rewriting not introducing fresh renamings (see [BK04]), and terminates with the E -rewriting presented here. The examples of non-termination that arise from the correspondence to β -reduction use both T -rewriting and E -rewriting.*

6 No Occur Checks

By extending the well-named predicate we can safely omit the occur checks in (R T-unify) and (R E-unify), and omit (R Z-unify) entirely, while keeping the confluence, progress, and termination properties. Section 6.1 defines the variable graph of a constraint, inspired by condition (d) of the System I type inference algorithm’s “progress invariant” [KW04]. Section 6.2 defines the acyclic well-named constraint sets and shows that they are closed under reduction.

6.1 Variable Graphs

Note 46 *To eliminate the occur checks we require that for all T -rewriting constraint forms $\alpha \doteq \bar{\tau}$ we require $\alpha \notin \text{ovs}(\bar{\tau})$ and for all E -rewriting constraint forms $e \bar{\tau} \doteq \tau$ we require $e \notin \text{ovs}(\tau)$.*

Note 47 *To guarantee the outer variables of every single constraint LHS and RHS disjoint needs a stronger invariant because reduction can easily break outer variable disjointness in general. For example:*

$$\{a_1 \doteq a_2, a_2 \doteq a_3 \rightarrow a_1\} \xrightarrow[r]{\{a_2 \mapsto a_3 \rightarrow a_1\}} \{a_1 \doteq a_3 \rightarrow a_1\}$$

So every variable v that could become outer in one side of a constraint with variable v' outer in its other side must be distinct from v' . This condition can be formulated as outer variable-graph acyclicity.

Note 48 The variable graph $\text{ovgraph}(\Delta)$ of a constraint is a relation (whose elements are the graph arcs) over variable sequences (the graph vertices). If two variables do occur outer on opposite sides of the same constraint, or might occur on opposite sides after rewriting and simplification, they are connected in the graph g (that is, they are an element of the transitive closure g^+). The following notation helps give a simple definition of the variable graph.

Convention 9 (Negative type shorthand) Let $\overline{e\rho \rightarrow^n \sigma}$ abbreviate the negative type $e_1^+ \rho_1 \rightarrow (\cdots \rightarrow (e_n^+ \rho_n \rightarrow \sigma) \cdots)$ where there are no ρ', σ' such that $\sigma = \rho' \rightarrow \sigma'$. If $n = 0$ the outer constructor is not \rightarrow . \square

Definition 36 (Outer variable graph of a constraint, ovgraph).

$$\begin{aligned}
\text{ovgraph} &: \text{PolarConstraint} \rightarrow \mathbb{P}(\text{Variable}^* \times \text{Variable}^*) \\
\text{ovgraph}(\Gamma) &= \{(\vec{e}_i v_1, \vec{e}_i v_2) \mid \gamma \in \Gamma \text{ and } (v_1, v_2) \in \text{ovgraph}'(\gamma)\} \\
\text{ovgraph}'(\alpha^+ \doteq \overline{e\rho \rightarrow^n \alpha'^-}) &= \{\alpha\} \times (\{\alpha'\} \cup \{e_1 \mid n \geq 1\}) \quad (1) \\
\text{ovgraph}'(\overline{(\overline{e\rho \rightarrow^m \sigma})} \rightarrow \bar{\pi} \doteq \overline{e'\rho' \rightarrow^n \alpha'}) &= \text{ovs}(|\sigma|) \times (\{\alpha'\} \cup \{e'_1 \mid n \geq 1\}) \quad (2) \\
\text{ovgraph}'(e^c \tau \leq \overline{e\rho \rightarrow^n \sigma}) &= \text{ovs}(|\sigma|) \times \{e\} \quad (3) \\
\text{ovgraph}'(\bar{\rho} \doteq \omega) &= \text{ovgraph}'(\bar{\rho} \doteq \sigma_1 \cap \sigma_2) = \emptyset \quad (4)
\end{aligned}$$

\square

Note 49 Rules (1) and (2) in the definition of ovgraph show the general form of well-named T -rewriting forms and unsimplified well-named forms with \rightarrow outermost on both sides. All the E -variables e , e_i and e'_i in these forms are positive and therefore mutually distinct. The graph pairs the remaining positive-side outer variables with the remaining negative-side outer variables. E -variable e_1 in (1) and e'_1 in (2) needs including so that simplification of constraints with \rightarrow constructors outermost preserve acyclicity.

Rule (3) “is” the occur check for E -rewriting forms. It pairs the negative-side outer variables with the positive e .

Rule (4) applies to Z -rewriting forms. Z -rewriting will only be needed when there is a constructor class.

Note 50 Preservation of graph acyclicity is harder to prove using a more obvious definition like

$$\text{ovgraph}'(\rho \doteq \sigma) = \text{ovs}(\rho) \times \text{ovs}(\sigma)$$

where all positively charging E -variables are included. Also, omitting all positive E -variables does not work.

Note 51 The T -variable pairs in the graph provide the occur check as long as T -rewriting is used: any variable that could become outer opposite \mathbf{a}_1 after a series of T -rewrites is reachable from \mathbf{a}_1 in the graph. Returning to the earlier example:

$$\text{ovgraph}(\{\mathbf{a}_1 \doteq \mathbf{a}_2, \mathbf{a}_2 \doteq \mathbf{a}_3 \rightarrow \mathbf{a}_1\}) = \{(\mathbf{a}_1, \mathbf{a}_2), (\mathbf{a}_2, \mathbf{a}_3), (\mathbf{a}_2, \mathbf{a}_1)\}$$

Similarly, the E -variable pairs provide the occur check for E -rewriting.

Note 52 *The role of the mixed variable type pairs, and the reason for form (3) arcs going negative to positive, can be illustrated by considering a reduction that causes arrow simplification:*

$$\begin{aligned}
& \text{ovgraph}(\{e_1 a_1 \doteq e_2 a_2 \cap e_3 a_3, e_1 a_4 \rightarrow a_5 \doteq a_6, a_6 \doteq e_7 a_7 \rightarrow a_8\}) \\
&= \{(e_2, e_1), (e_3, e_1), (e_1, a_6), (a_6, e_7), (a_6, a_8)\} \\
& \text{ovgraph}(\{e_2 a_4 \cap e_3 a_4 \rightarrow a_5 \doteq a_6, a_6 \doteq e_7 a_7 \rightarrow a_8\}) \\
&= \{(e_2, a_6), (e_3, a_6), (a_6, e_7), (a_6, a_8)\} \\
& \text{ovgraph}(\{e_2 a_4 \cap e_3 a_4 \rightarrow a_5 \doteq e_7 a_7 \rightarrow a_8\}) \\
&= \{(e_2, e_7), (e_3, e_7), (e_2, a_8), (e_3, a_8)\} \\
& \text{ovgraph}(\{e_7 a_7 \doteq e_2 a_4 \cap e_3 a_4, a_5 \doteq a_8\}) \\
&= \{(e_2, e_7), (e_3, e_7), (a_5, a_8)\}
\end{aligned}$$

The first reduction illustrates the reason for negative-to-positive in E-rewriting form (3) constraints. The last step is a simplification, not a reduction, it illustrates the effect of the domain-type reversal of simplify on the graph shape: note that the two preserved arcs do not change direction, so the argument that acyclicity is preserved is relatively simple.

Definition 37 (Graph vertices, V). $V(g) = \bigcup_{i=1}^n \{v_i, v'_i\}$ where $g = \{(v_i, v'_i)\}_{i=1}^n$. \square

Note 53 *Variable graphs always partition into a number of disjoint graphs: $g = g_1 \uplus \dots \uplus g_n$, one for each prefix \vec{e} of a vertex of the form $\vec{e}v$, as the prefixes are equal for the source and target vertices of every graph arc. Note also that positive T-variables only occur as arc sources but all other variables can be sources or targets in the graph. Variable sequences ending in a T-variable will only occur in the graphs of at most two single constraints owing the wellnamed restriction but those ending in an E-variable can be in the graphs of many single constraints.*

The following lemma can be used to show that a local transformation on any directed graph encoded as a relation preserves graph acyclicity.

Lemma 13 (Acyclicity preserving graph transformation). *If (1) $g_1 \uplus g_2$ is acyclic and (2) g_3 is acyclic and (3) $(\vec{v}_1 \in \text{dom}g_3 \cap V(g_1)) \wedge (\vec{v}_1, \vec{v}_2) \in g_3^+ \wedge \vec{v}_2 \in V(g_1) \Rightarrow (\vec{v}_1, \vec{v}_2) \in g_2^+$ then $g_1 \cup g_3$ is acyclic.*

Proof. If (a) $(\vec{v}_1, \vec{v}_2) \in g_1^+$ or (b) $(\vec{v}_1, \vec{v}_2) \in g_3^+$ then $\vec{v}_1 \neq \vec{v}_2$ by (1) or (2). If $(\vec{v}_1, \vec{v}_2) \in g_1^+ \wedge (\vec{v}_2, \vec{v}_3) \in g_3^+$ then (c) $\vec{v}_1 \neq \vec{v}_2$ or else (d) $\vec{v}_3 \in V(g_1)$ and $(\vec{v}_2, \vec{v}_3) \in g_2^+$ by (3). Similarly, if $(\vec{v}_1, \vec{v}_2) \in g_3^+ \wedge (\vec{v}_2, \vec{v}_3) \in g_1^+$, (e) $\vec{v}_1 \notin V(g_1)$ or (f) $(\vec{v}_1, \vec{v}_2) \in g_2^+$.

If $(\vec{v}_1, \vec{v}_2) \in (g_1 \cup g_3)^+$ does not have one of these 6 forms then it results from: some number of (d)s then an (a), these are all in $(g_1 \cup g_2)^+$; some (d)s then a (c) so $\vec{v}_1 \neq \vec{v}_2$; similarly for some (f)s then a (b) or an (e). \square

6.2 Acyclic Well-Named Constraint Sets

Note 54 *We can prove that variable graph acyclicity is preserved by both simplification and reduction for the following set of acyclic well-named constraint*

sets. This places a further restriction on the shape of constraints, in addition to demanding well-namedness and graph acyclicity. The extra restriction is that the RHS argument of a negative \rightarrow constructor must be a restricted negative type and the RHS argument of a positive \rightarrow must have an E-variable outermost. The difficulty this avoids is with constraints such as:

$$\sigma \rightarrow \rho \doteq \rho' \rightarrow \sigma'$$

If Σ' is an unrestricted negative type its outer namespace could contain negative E-variables which must be disjoint from the outer variables of ρ and which the graph must include. Proving preservation of acyclicity by reduction and simplification is more difficult if arbitrary negative types are permitted. Indeed, a stronger graph definition would be needed to prevent forms like $\mathbf{a}_1 \rightarrow \mathbf{e}_1 \mathbf{a}_1 \doteq \mathbf{a}_2 \rightarrow \mathbf{e}_1 \mathbf{a}_1$, and the graph approach may not work with arbitrary negative types.

Definition 38 (Acyclic Well-Named).

$$\begin{array}{lll} \bar{\pi} \in & \text{PType}^{\rightarrow} ::= \alpha^+ \mid \nu \rightarrow \bar{\pi} & (\text{restricted pos types}) \\ \pi \in & \text{PType} ::= e^+ \bar{\pi} & (\text{pos types}) \\ \bar{\nu} \in & \text{NType}^{\rightarrow} ::= \alpha^- \mid \pi \rightarrow \bar{\nu} & (\text{restricted neg types}) \\ \nu \in & \text{NType} ::= \bar{\nu} \mid \omega \mid e^- \nu \mid \nu \cap \nu_1 & (\text{neg types}) \\ \beta \in \text{PNConstraint} ::= \pi \doteq \nu \mid \bar{\pi} \doteq \bar{\nu} \mid e \beta & (\text{pn constraints}) \\ B \in \mathbb{P}(\text{PNConstraint}) & & (\text{pn constraint sets}) \end{array}$$

$\text{verywellnamed}(\Delta \cup \Delta_1)$ iff

- Δ_1 is solved: $\text{solved}(\Delta_1)$;
- Δ pn-charges: for all Δ there is a B such that $|B| = \Delta$;
- Δ is well-named: $\text{wellnamed}(\Delta)$;
- Δ is acyclic: $\text{ovgraph}(B)^+$ is irreflexive. □

Lemma 14 (Simplification preserves acyclic well-named). If $\text{verywellnamed}(\Delta)$ then $\text{verywellnamed}(\text{factor}(\Delta))$.

Proof. Because Δ is well-named, it suffices to consider constraints of the following form.

$$(\overline{e \pi \rightarrow^m \alpha'}) \rightarrow e \bar{\pi} \doteq e'_0 \pi'_0 \rightarrow (\overline{e' \pi' \rightarrow^n \alpha})$$

These simplify into (no deeper simplification is possible):

$$\left\{ e'_0 \pi'_0 \doteq \overline{e \pi \rightarrow^m \alpha'}, e \bar{\pi} \doteq \overline{e' \pi' \rightarrow^n \alpha} \right\}$$

The transformation induced on the variable graph is:

$$\{(\alpha', e'_0), (\alpha', \alpha)\} \Rightarrow \{(\alpha', e'_0), (\alpha, e)\}$$

The removal of arcs does not compromise acyclicity. The addition of the arc (α, e) cannot introduce a cycle as α is a negative occurrence and its positive occurrence can only be a source in the graph. □

Theorem 10 (Reduction preserves acyclic well-named). *If $\text{verywellnamed}(\Delta_1)$ and $\Delta_1 = \text{factor}(\Delta_1)$ and $\Delta_1 \xrightarrow[r]{E} \Delta_2$ then $\text{verywellnamed}(\Delta_2)$.*

Proof. By cases on the \rightarrow rules and β where $\Delta_1 = \Delta_3 \cup \Delta_4 \cup \{\vec{e}\bar{\Delta}\}$ and $\text{solved}(\Delta_3)$ and $|B_4 = \Delta_4$ and $|\beta| = \bar{\Delta}$.

- Case (R T-unify) where $\beta = \alpha^+ \doteq \bar{\nu}$ or $\beta = \bar{\pi} \doteq \alpha^-$, and $\vec{e}\alpha \notin \mathbf{V}(\text{ovgraph}(B_4))$. Then $\Delta_2 = ([E] \Delta_3) \cup \Delta_4 \cup \{[E] \vec{e}\bar{\Delta}\}$.
- Case (R T-unify) where $\beta = \alpha^+ \doteq \overline{e\rho \rightarrow^m} \alpha_1^-$ and $\vec{e}\alpha \in \mathbf{V}(\text{ovgraph}(B_4))$ and $E = \vec{e}/\{\alpha \mapsto \bar{\tau}\}$.

Then $\Delta_2 = \Delta_3 \cup [E] \Delta_4 \cup \{\overline{e|\rho| \rightarrow^m} \alpha_1 \doteq \overline{e|\rho| \rightarrow^m} \alpha_1\}$. and $\{\vec{e}\beta'\} \in B_4$ and $\alpha^- \in \text{ovs}(\beta')$. The polar shape and occurrence restrictions are preserved as this α^- is the only other α in namespace \vec{e} . Proceed by case analysis of β' .

- $\alpha_2 \doteq \overline{e'\pi' \rightarrow^n} \alpha$. Graph transformation: $\{(\alpha_2, \alpha), (\alpha_2, e'_1), (\alpha, e_1), (\alpha, \alpha_1)\} \Rightarrow \{(\alpha_2, e'_1), (\alpha_2, e_1), (\alpha_2, \alpha_1)\}$; Omit the e_1 pairs if $m = 0$ and omit the e'_1 pairs if $n = 0$.
- $(\overline{e'\pi' \rightarrow^n} \alpha) \rightarrow \pi \doteq \overline{e''\bar{\pi}'' \rightarrow^p} \alpha_2$. Graph transformation: $\{(\alpha, e''_1), (\alpha, \alpha_2), (\alpha, e_1), (\alpha, \alpha_1)\} \Rightarrow \{(\alpha_1, e''_1), (\alpha_1, \alpha_2)\}$; Omit the e''_1 if $p = 0$ and omit the e_1 if $m = 0$; this cannot introduce a cycle as the other α_1 is positive so it can only be a source in the rest of the graph.
- Case (R T-unify) where $\beta = \alpha_1^+ \doteq \alpha^-$ and $E = \vec{e}/\{\alpha \mapsto \alpha_1\}$ is very similar to the previous case.
- Case (R T-unify) where $\beta = (\overline{e\bar{\pi} \rightarrow^m} \alpha_1^-) \rightarrow \pi \doteq \alpha^-$ and $\vec{e}\alpha \in \mathbf{V}(\text{ovgraph}(B_4))$ and $E = \vec{e}/\{\alpha \mapsto \bar{\tau}\}$.

Then $\Delta_2 = \Delta_3 \cup [E] \Delta_4 \cup \{\overline{e|\rho| \rightarrow^m} \alpha_1 \doteq \overline{e|\rho| \rightarrow^m} \alpha_1\}$. and $\{\vec{e}\beta'\} \in B_4$ and $\alpha^+ \in \text{ovs}(\beta')$. The polar shape and occurrence restrictions are preserved as this α^+ is the only other α in namespace \vec{e} . So $\beta' = \alpha \doteq \overline{e'\pi' \rightarrow^n} \alpha_2$. Graph transformation:

- $\{(\alpha_1, \alpha), (\alpha, e'_1), (\alpha, \alpha_2)\} \Rightarrow \{(\alpha_1, e'_1), (\alpha_1, \alpha_2)\}$; Omit the e'_1 pairs if $n = 0$.
- Case (R E-unify) where $\beta = e\bar{\pi} \doteq \nu$ and $\{(\vec{e}_i, R_i)\}_{i=1}^n = \mathbf{eExtract}(|\nu|)$. Let $\text{ovgraph}(\Delta_1) = \bigcup_{i=1}^n g_{\vec{e}\vec{e}_i} \cup g_{\vec{e}e} \cup g'$ where $g_{\vec{e}}$ is the partition of g for variables in namespace \vec{e} .

Then $\text{ovgraph}(\Delta_2) = \bigcup_{i=1}^n (g_{\vec{e}\vec{e}_i} \cup g''_i) \cup g' \cup \text{ovgraph}([E] \beta)$.

where $g''_i = \{(\vec{e}f_i(v), \vec{e}f_i(v')) \mid (\vec{e}v, \vec{e}v') \in g_{\vec{e}e}\}$ and $f_i(\alpha) = R_i(\alpha)$ and $f_i(e) = e'$ if $R_i(e) = e' \square$. For each i : $g_{\vec{e}\vec{e}_i}$ and g''_i are disjoint by the definition of the fresh renamings R_i . In each of the n partitions of $\text{ovgraph}([E] \beta)$ all arcs either point from variables in $g_{\vec{e}\vec{e}_i}$ to variables in g''_i , or vice-versa therefore $\text{ovgraph}(\Delta_2)$ is acyclic. \square

7 Application to System E

Section 7.1 explains the various minor differences between the presentation of unification with E-variables and $\xrightarrow[r]{E}$ used here and the formulations we use for typing inference in System E. Having established that there are no significant differences, Section 7.2 shows that the constraints used for exact call-by-name

and call-by-value typing inference in System E in [BCKW04] are acyclic well-named and therefore inherit the progress, confluence and termination properties we have established.

7.1 Differences From System E

Note 55 *There are a minor notational differences between our presentation and System E. None of these make any difference for the results presented in this report.*

- *System E Substitutions are defined by a grammar. They are finite lists of variable assignments terminated by \square . Whereas here they are functions, which may have infinite supports. There may be more than one mapping for any variable in a System E substitution, all those after the first are ignored garbage.*
- *Constraints are called single, or singular, constraints in System E. They are constructed with the asymmetric \leq , instead of our symmetric \doteq , because it can be used with subtyping constraints.*
- *Constraint sets are called constraints in System E, the empty constraint set is denoted ω and the constraint set $\{\bar{\Delta}\} \uplus \Delta$ can be written $\bar{\Delta} \cap \Delta$. The \cap constraint constructor is not idempotent so a constraint may contain repeated single constraints.*

Note 56 *In System E with reflexive subtyping, the following equality laws are allowed (because they are sound in the sense that applying an expansion to equal entities produces equal results and because they form a large equality which is convenient for many purposes) for both types Y and constraints Y .*

1. *Association: $Y_1 \cap (Y_2 \cap Y_3) = Y_1 \cap Y_2 \cap Y_3$.*
2. *Commutation: $Y_1 \cap Y_2 = Y_1 \cap Y_2$;*
3. *Absorption: $Y \cap \omega = \omega$.*
4. *Distribution: $e(Y_1 \cap Y_2) = e Y_1 \cap e Y_2$.*
5. *Distribution: $e \omega = \omega$.*

These additions do not change any of our results.

Note 57 *In System E we use a slight variation on \Rightarrow_r for constraint solving. This omits (R Z-unify) because constraints are always acyclic well-named, as shown in the next section. It also allows its (U T-unify) to be used on constraints of the form $\tau \leq \alpha$. Owing to the restrictions of well-named such constraints will always be of the form $\bar{\tau} \leq \alpha$ or $e \bar{\tau} \leq \alpha$. Therefore reduction with \Rightarrow_r keeps the confluence, progress and termination properties.*

Definition 39 (System E constraint reduction, $\xrightarrow[r]{S}$).

$\Delta \xrightarrow[u]{\bar{e}/S} \text{factor}([\bar{e}/S] \Delta)$ if $\bar{e} \bar{\Delta} \cap \Delta_1 = \text{factor}(\Delta)$ and $\bar{\Delta} \xrightarrow{\text{unifier}} S$ where:

$\tau_1 \leq \tau_2 \xrightarrow{\text{unifier}} \alpha := \tau$ if $\{\tau_1, \tau_2\} = \{\alpha, \tau\}$ (U T-unify)

$e \tau \leq \tau_1 \xrightarrow{\text{unifier}} e := E$ if $\tau_1 \notin \text{T-Variable}$ and $(\tau_1, \text{vars}(\Delta)) \xrightarrow{\text{retract}} (E, \mathcal{V}')$ (U E-unify)

□

7.2 Application to System E

Note 58 *The System E exact typing inference procedure uses a relation uSkN to map a term of the λ -calculus to a typing skeleton (a skeleton summarises a typing derivation) for its call-by-name exact typing analysis. Another relation, uSkV maps a term to a typing skeleton for its call-by-value exact typing analysis. Skeletons encodes the typing problem for the term as a type environment, a result type and a constraint set. Applying a unifier of the constraint set to the skeleton produces a System E typing derivation for the term. Here we give the basic λ -calculus and type environment definitions needed to generate initial constraint sets. Then we show that initial constraint sets are acyclic well-named, therefore when constraint set reduction is being used for typing inference it has all the beneficial properties we have investigated.*

Definition 40 (λ -calculus terms). *Term variables and terms are the least sets satisfying:*

$$\begin{aligned} \text{Term-Variable} &= \{x_i \mid i \in \mathbb{K}\} \text{ where } \mathbb{K} \text{ is an arbitrary countable set of indices} \\ \text{Var-HNF} &\supseteq \text{Term-Variable} \cup \{U @ M \mid U \in \text{Var-HNF} \text{ and } M \in \text{Term}\} \\ \text{Value} &\supseteq \text{Var-HNF} \cup \{\lambda x. M \mid x \in \text{Term-Variable} \text{ and } M \in \text{Term}\} \\ \text{Term} &\supseteq \text{Value} \cup \{M_1 @ M_2 \mid M_i \in \text{Term}\} \end{aligned}$$

□

Definition 41 (Type environments).

1. A type environment is a total function $A : \text{TermVar} \rightarrow \text{Type}$ which maps finitely many term variables to non- ω types.
2. The empty environment $()$ is defined $\{x \mapsto \omega \mid x \in \text{TermVar}\}$ and $(x_1 \mapsto \tau_1, \dots, x_n \mapsto \tau_n)$ abbreviates $() [x_1 \mapsto \tau_1] \dots [x_n \mapsto \tau_n]$.⁶
3. environment intersection: $A \cap A' = \{x \mapsto \tau \mid \{A(x), A'(x)\} = \{\omega, \tau\}\} \cup \{x \mapsto \tau \cap \tau' \mid \{A(x), A'(x)\} \neq \{\omega, \tau\}\}$.
4. Environment expansion:
 $[E] A = \{x \mapsto [E] A(x) \mid A(x) \neq \omega\} \cup \{x \mapsto \omega \mid A(x) = \omega\}$.
5. E-variable application: $e A = [e \square] A$. □

Definition 42 (Term to initial call-by- x , uSkx). *This inductive definition of uSkx maps terms to triples containing a type environment, a type and the initial constraint (using the full System E skeleton formulation is unnecessary for our purposes). Each case assumes $M_i \xrightarrow{\text{uSkx}} \langle A_i, \tau_i, \Delta_i \rangle$ for subterm M_i .*

$$\begin{aligned} x &\xrightarrow{\text{uSkV}} \langle (x : \alpha), \alpha, \emptyset \rangle \\ \lambda x. M_1 &\xrightarrow{\text{uSkV}} \langle e A[x : \omega], e A(x) \rightarrow e \tau, e \Delta \rangle && \text{if } M_1 \in \text{Value} \\ \lambda x. M_1 &\xrightarrow{\text{uSkV}} \langle A[x : \omega], A(x) \rightarrow \tau, \Delta \rangle && \text{if } M_1 \notin \text{Value} \\ M_1 @ M_2 &\xrightarrow{\text{uSkV}} \langle A_1 \cap e A_2, \alpha, \Delta_1 \uplus e \Delta_2 \uplus \{\tau_1 \doteq e \tau_2 \rightarrow \alpha\} \rangle \\ &&& \text{if } \alpha, e \notin \text{ovs}(Q_1) \text{ and } M_2 \in \text{Value} \\ M_1 @ M_2 &\xrightarrow{\text{uSkV}} \langle A_1 \cap A_2, \alpha, \Delta_1 \uplus \Delta_2 \uplus \{\tau_1 \doteq \tau_2 \rightarrow \alpha\} \rangle \\ &&& \text{if } \{\{\alpha\}, \text{ovs}(Q_1), \text{ovs}(Q_2)\} \text{ are pairwise disjoint and } M_2 \notin \text{Value} \end{aligned}$$

⁶ The function modification operator $A[x \mapsto \tau]$ gives a function which is the same as A for every variable except x which it maps to τ .

$$\begin{aligned}
& x \xrightarrow{\text{uSkN}} \langle (x : \alpha), \alpha, \emptyset \rangle \\
& \lambda x. M \xrightarrow{\text{uSkN}} \langle A[x : \omega], A(x) \rightarrow \tau, \Delta \rangle \\
M_1 @ M_2 & \xrightarrow{\text{uSkN}} \langle A_1 \cap e A_2, \alpha, \Delta_1 \uplus e \Delta_2 \uplus \{\tau_1 \doteq e \tau_2 \rightarrow \alpha\} \rangle \quad \text{if } e, \alpha \notin \text{ovs}(\Delta_1)
\end{aligned}$$

□

Theorem 11. *Initial constraints are acyclic well-named If $M \xrightarrow{\text{uSkx}} \langle A, \tau, \Delta \rangle$ then $\text{Acyclic}(\Delta)$.*

Proof. By induction on M the following properties hold. If $M \xrightarrow{\text{uSkx}} \langle (x_1 : \tau_1, \dots, x_n : \tau_n), \tau, \Delta \rangle$ then there are $\Phi, B, \bar{\pi}$ and ν_i such that:

1. $|B| = \Delta$
2. $|\nu_i| = \tau_i$
3. $|\bar{\pi}| = \tau$
4. $\Phi = \{\text{vpaths}(\nu_i)\}_{i=1}^n \cup \text{vpaths}(\bar{\pi}) \cup \text{vpaths}(B)$ satisfies the occurrence conditions of Definition 30.
5. $\bar{\pi}$ contains no negative arrows
6. each ν_i contain no arrows.
7. $\text{ovgraph}(B)$ is acyclic.

(1) to (3) show that the initial judgement has a polar constraint and a polar typing as required, (4) implies $\text{wellnamed}(B)$, (5) and (6) are help to guarantee that a chargeable structure is produced.

- Case x : Let $B = \emptyset, \bar{\pi} = \mathbf{a}_i^+, \nu_1 = \mathbf{a}_i^-$.
 1. $|B| = \emptyset$;
 2. $|\nu_1| = \mathbf{a}_i$;
 3. $|\bar{\pi}| = \mathbf{a}_i$;
 4. $\Phi = \{\mathbf{a}_i^-, \mathbf{a}_i^+\}$ satisfies the occurrence conditions;
 5. \mathbf{a}_i^+ contains no arrows;
 6. \mathbf{a}_i^- contains no arrows;
 7. $\text{ovgraph}(\emptyset) = \emptyset$ is acyclic.
- Case $\lambda x. M$. The properties hold for $M \xrightarrow{\text{uSkx}} \langle (x : \tau_x, A), \tau, \Delta \rangle$ for some $\Phi, B, \bar{\pi}, \nu_x$ and $\nu_j (1 \leq j \leq n)$ by I.H.
 1. $|B| = \Delta$; or $|e B| = e \Delta$;
 2. $|\nu_j| = \tau_j$; or $|e \nu_j| = e \tau_j$;
 3. $|\nu_x \rightarrow \bar{\pi}| = \tau_x \rightarrow \tau$; or $|e^- \nu_x \rightarrow e^+ \bar{\pi}| = e \tau_x \rightarrow e \tau$;
 4. $\bigcup_{i=1}^n \text{vpaths}(\nu_i) \cup \text{vpaths}(\bar{\pi}) \cup \text{vpaths}(B) = \Phi$ therefore it satisfies the occurrence conditions;
 5. ν_x contains no arrows and $\bar{\pi}$ no negative arrows so $\nu_x \rightarrow \bar{\pi}$ or $e^- \nu_x \rightarrow e^+ \bar{\pi}$ has no negative arrows;
 6. ν_i or $e \nu_i$ contains no arrows.
 7. $\text{ovgraph}(\Delta)$ or $\text{ovgraph}(e \Delta)$ remains acyclic.
- Case $M @ N$. Analogous to the previous case. □

References

- [Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.
- [BCKW04] A. Bakewell, S. Carlier, A. J. Kfoury, and J. B. Wells. Exact intersection typing inference and call-by-name evaluation. Technical report, Department of Computer Science, Boston University, Dec. 2004.
- [BK04] A. Bakewell and A. J. Kfoury. Unification with expansion variables. Technical report, Department of Computer Science, Boston University, Dec. 2004.
- [Car02] S. Carlier. Polar type inference with intersection types and ω . In *Proceedings of the 2nd Workshop on Intersection Types and Related Systems*, 2002.
- [CPWK04] S. Carlier, J. Polakow, J. B. Wells, and A. J. Kfoury. System E: Expansion variables for flexible typing with linear and non-linear types and intersection types. In *Programming Languages & Systems, 13th European Symp. Programming*, vol. 2986 of *LNCS*, pp. 294–309. Springer-Verlag, 2004.
- [DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, Aug. 1979.
- [Jim00] T. Jim. A polar type system. In J. D. P. Rolim, A. Z. Broder, A. Corradini, R. Gorrieri, R. Heckel, J. Hromkovic, U. Vaccaro, and J. B. Wells, eds., *ICALP Workshops 2000: Proceedings of the Satellite Workshops of the 27th International Colloquium on Automata, Languages, and Programming*, vol. 8 of *Proceedings in Informatics*, pp. 323–338, Geneva, Switzerland, July 2000. Carleton Scientific.
- [KW99] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Conf. Rec. POPL '99: 26th ACM Symp. Princ. of Prog. Langs.*, pp. 161–174, 1999. Superseded by [KW04].
- [KW03] A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. Supersedes [KW99], Aug. 2003.
- [KW04] A. J. Kfoury and J. B. Wells. Principality and type inference for intersection types using expansion variables. *Theoret. Comput. Sci.*, 311(1–3):1–70, 2004. Supersedes [KW99]. For omitted proofs, see the longer report [KW03].