Nominal Unification with Triangular Substitutions

Ramana Kumar   Michael Norrish

11 July 2010

THE AUSTRALIAN NATIONAL UNIVERSITY

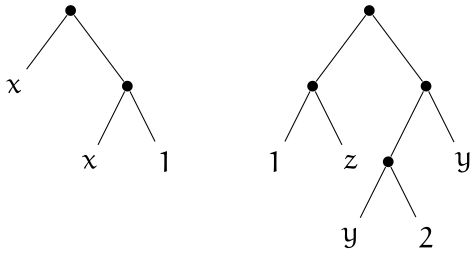
NICTA

Outline

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$\langle x, \langle x, 1\rangle\rangle =_? \langle\langle 1, z\rangle, \langle\langle y, 2\rangle, y\rangle\rangle$

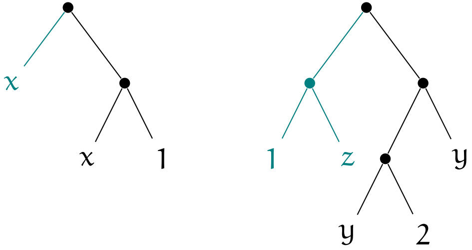


Accumulator: $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$x =_? \langle 1, z \rangle$

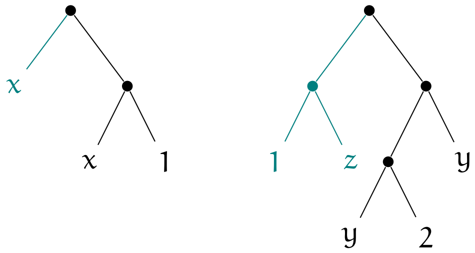


Accumulator: $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$x =_? \langle 1, z \rangle$

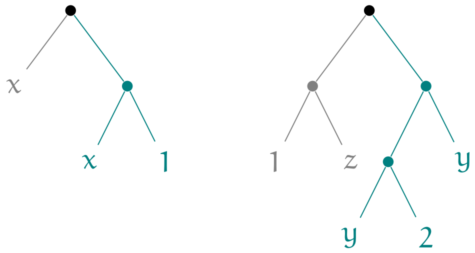


Accumulator: $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$\langle x, 1 \rangle =_? \langle \langle y, 2 \rangle, y \rangle$

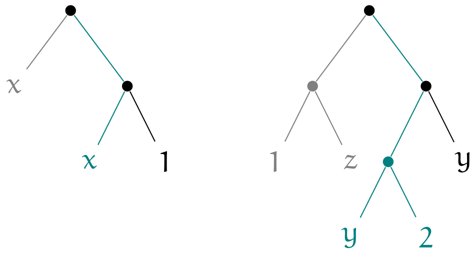


Accumulator: $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$x =_? \langle y, 2 \rangle$

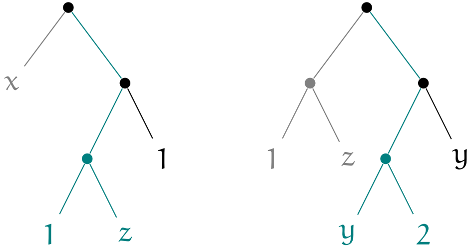


Accumulator: $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$\langle 1, z \rangle =_? \langle y, 2 \rangle$

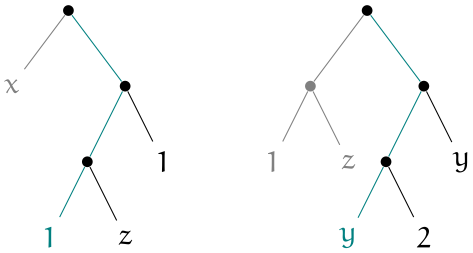


Accumulator: $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$\langle 1, z \rangle =_? \langle y, 2 \rangle$

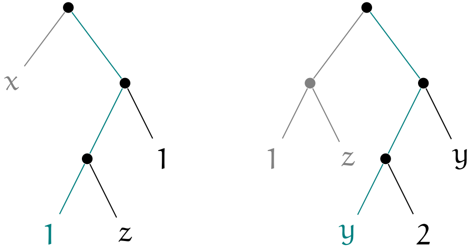


Accumulator: $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$1 =_? y$

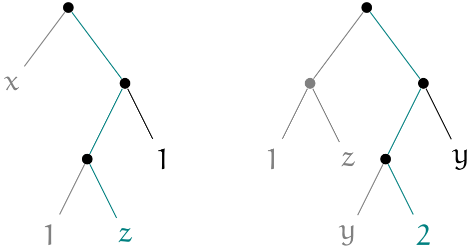


Accumulator: $y \mapsto 1,\ x \mapsto \langle 1, z \rangle,\ \emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$z =_? 2$

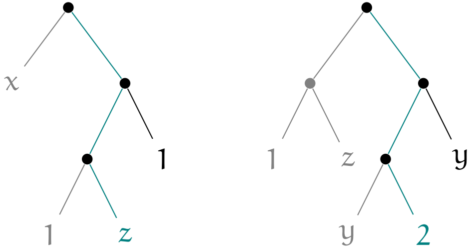


Accumulator: $y \mapsto 1$, $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$z =_? 2$

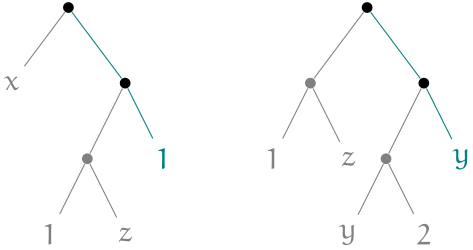


Accumulator: $z \mapsto 2$, $y \mapsto 1$, $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$1 =_? y$

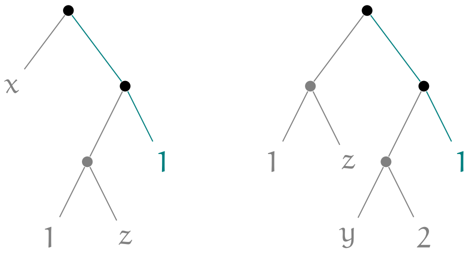


Accumulator: $z \mapsto 2$, $y \mapsto 1$, $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

$1 =_? 1$

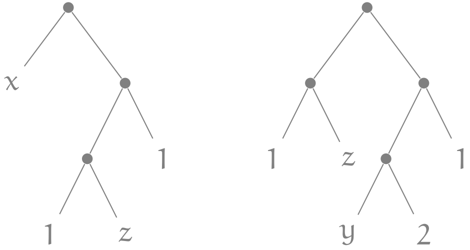


Accumulator: $z \mapsto 2$, $y \mapsto 1$, $x \mapsto \langle 1, z \rangle$, $\emptyset$

Unifying Terms to Build a Triangular Substitution

Do these terms unify?

Yes.



Accumulator: $z \mapsto 2$, $y \mapsto 1$, $x \mapsto \langle 1, z \rangle$, $\emptyset$

Triangular Substitutions

Unifying $\langle\langle x, 2\rangle, x\rangle$ and $\langle y, \langle 1, z\rangle\rangle$

- Triangular: $x \mapsto \langle 1, z\rangle,\ y \mapsto \langle x, 2\rangle$
- Idempotent: $x \mapsto \langle 1, z\rangle,\ y \mapsto \langle\langle 1, z\rangle, 2\rangle$

Composing with the result of unifying z and 3

- Triangular: $z \mapsto 3, x \mapsto \langle 1, z\rangle,\ y \mapsto \langle x, 2\rangle$
- Idempotent: $x \mapsto \langle 1, 3\rangle,\ y \mapsto \langle\langle 1, 3\rangle, 2\rangle,\ z \mapsto 3$

Advantages:
 persistence, memory sharing, no copying, can be much smaller.

Backtracking with Triangular Substitutions

$x = 1$ or $x = 2$ $y = \langle 1, x \rangle$ or $y = \langle x, 2 \rangle$ $\langle w, \langle x, z \rangle \rangle = \langle x, y \rangle$
initial subst

Backtracking with Triangular Substitutions

$x = 1$ or $x = 2$

$y = \langle 1, x \rangle$ or $y = \langle x, 2 \rangle$

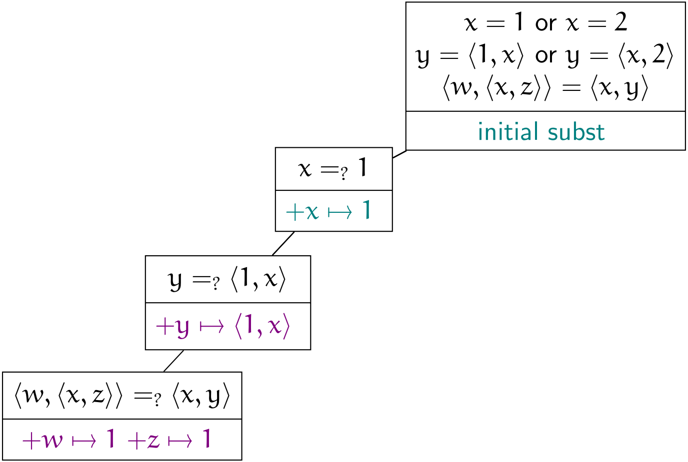
$\langle w, \langle x, z \rangle \rangle = \langle x, y \rangle$

initial subst

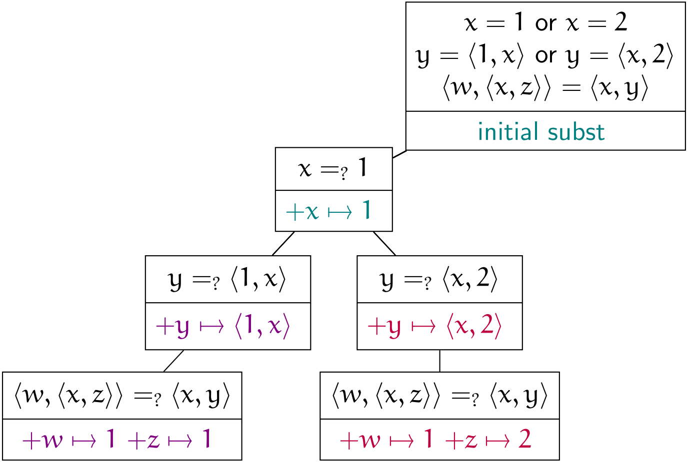
$x =_? 1$

$+x \mapsto 1$

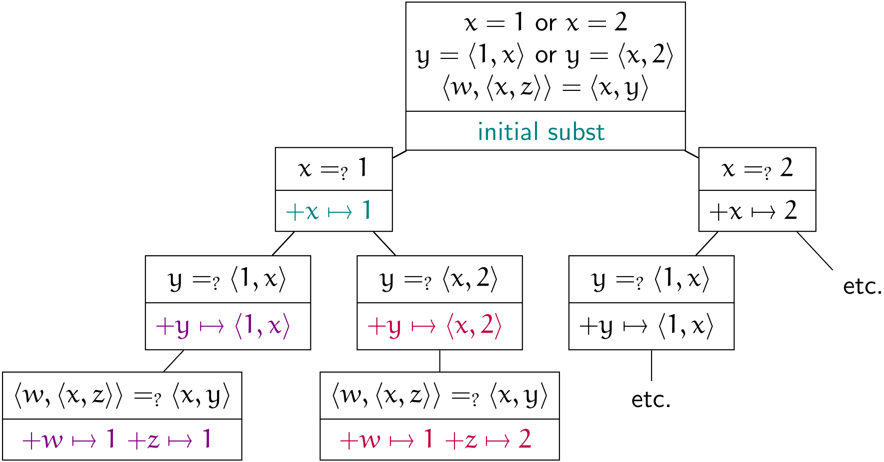
Backtracking with Triangular Substitutions



Backtracking with Triangular Substitutions



Backtracking with Triangular Substitutions



Applying Triangular Substitutions

Let $\sigma = \{z \mapsto 3, y \mapsto \langle z, 1\rangle, x \mapsto y\}$, $t = x$.

One-step	Homomorphic lift of map from variables to terms. (Unused. Result would be y.)
Full	Repeated one-step. $\sigma \triangleleft t = \langle 3, 1\rangle$.
Walk	Repeats only on top-level variables. `walk` $\sigma\ t = \langle z, 1\rangle$.

Well-formedness

Applying $\{z \mapsto x, y \mapsto z, x \mapsto y\}$ to x loops indefinitely.

Disallow substitutions with loops:

- Non-looping substitutions are *well-formed* (`wfs`).
- $\{z \mapsto \langle x, 3\rangle, y \mapsto z, x \mapsto y\}$ is not well-formed.
- $\vdash \texttt{wfs}\ \sigma \iff \exists n.\, \sigma^n$ is idempotent.

Outline

Nominal Terms

Terms t form an algebraic type of five constructors:

- **Nom** Object-level names/atoms a_1, $a_2 \ldots$
- **Const** Arbitrary non-name data c_1, $c_2 \ldots$
- **Susp** A suspension: a meta-variable coupled with a permutation (over noms). $\pi_1 \cdot V_1$, $\pi_2 \cdot V_2 \ldots$
- **Tie** A binding: a name (bound variable) coupled with a term (the body). $[a_1]t_1$, $[a_2]t_2 \ldots$
- **Pair** A pair of terms. $\langle t_{11}, t_{12} \rangle$, $\langle t_{21}, t_{22} \rangle \ldots$

We don't quotient by α-equivalence on ties.

Freshness Constraints

A pair $(a \# V)$

meaning: V should not be bound to a term with a free.

Example

Q. $[a_1]V_1 =_? [a_2]V_2$

A. Yes: $\sigma = \{V_1 \mapsto (a_1\ a_2) \cdot V_2\}$, $\nabla = \{(a_1 \# V_2)\}$.

V_1	V_2	Verdict
a_3	a_3	✓
a_1	a_2	✓
a_2	a_1	✗

Freshness and Equivalence Judgements

Define $\nabla \vdash a \# t$ and $\nabla \vdash t_1 \approx t_2$.

Interesting rules

$$\frac{}{\nabla \vdash a \# [a]t} \qquad \frac{(\pi^{-1}(a) \# V) \in \nabla}{\nabla \vdash a \# \pi{\cdot}V}$$

$$\frac{\nabla \vdash t_1 \approx (a_1\ a_2){\cdot}t_2 \quad \nabla \vdash a_1 \# t_2}{\nabla \vdash [a_1]t_1 \approx [a_2]t_2}$$

Outline

Nominal Unification

Nominal unification works in two phases.

The first phase is similar to first-order unification.

The second phase checks freshness constraints.

Unification Algorithm 1

```
unify1 (σ,∇) t₁ t₂ =
  case (walk σ t₁, walk σ t₂) of
    (a₁, a₂) → if a₁ = a₂ then (σ,∇) else fail
    (c₁, c₂) → if c₁ = c₂ then (σ,∇) else fail
    (π₁·V₁, π₂·V₂) →
      if V₁ = V₂ then
        unify_eq_vars (dis_set π₁ π₂) V₁ (σ,∇)
      else
        add_bdg π₁ V₁ (π₂·V₂) (σ,∇)
    (π₁·V₁, t₂) → add_bdg π₁ V₁ t₂ (σ,∇)
    (t₁, π₂·V₂) → add_bdg π₂ V₂ t₁ (σ,∇)
    ...
```

(add_bdg does an *occurs check*.)

(unify_eq_vars adds freshness constraints.)

Unification Algorithm 2

```
unify1 (σ,∇) t₁ t₂ =
  case (walk σ t₁, walk σ t₂) of
    ...
    ([a₁]t₁, [a₂]t₂) →
      if a₁ = a₂ then unify1 (σ,∇) t₁ t₂
      else do
        ∇′ ← term_fcs a₁ (σ ◁ t₂);
        unify1 (σ, ∇ ∪ ∇′) t₁ ((a₁ a₂) · t₂)

    (⟨t₁₁,t₁₂⟩, ⟨t₂₁,t₂₂⟩) → do
      (σ′,∇′) ← unify1 (σ,∇) t₁₁ t₂₁;
      unify1 (σ′,∇′) t₁₂ t₂₂

    _ → fail
```

(term_fcs a t generates the minimal set of freshness constraints to make a fresh for t.)

Verifying Freshness Constraints

`unify2` σ ∇ generates a minimal ∇', such that for each $(a \# V)$ in ∇ we have

$$\nabla' \vdash a \# (\sigma \triangleleft V)$$

`unify2` might:

- add constraints ($\sigma = \{V \mapsto \langle V_2, V_3 \rangle\}$, $\nabla = (a \# V)$)
- remove constraints ($\sigma = \{V \mapsto a_1\}$, $\nabla = \{(a_2 \# V)\}$)
- fail ($\sigma = \{V \mapsto a_1\}$, $\nabla = \{(a_1 \# V)\}$)

Putting the Two Phases Together

```
unify (σ, ∇) t₁ t₂ = do
  (σ′, ∇′) ← unify1 (σ, ∇) t₁ t₂;
  ∇″ ← unify2 σ′ ∇′;
  (σ′, ∇″)
```

Unification Example

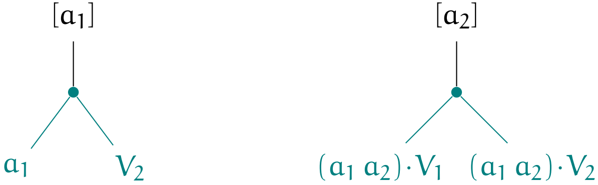
$[a_1]\langle a_1, V_2\rangle =_? [a_2]\langle V_1, V_2\rangle$

Substitution σ: $\emptyset$

Freshnesses ∇: $\emptyset$

Unification Example

$$\langle a_1, V_2 \rangle =_? (a_1\ a_2)\cdot\langle V_1, V_2 \rangle$$

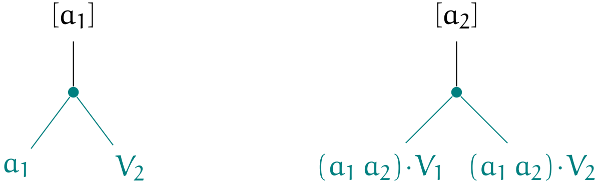


Substitution σ: $\emptyset$

Freshnesses ∇: $\emptyset$

Unification Example

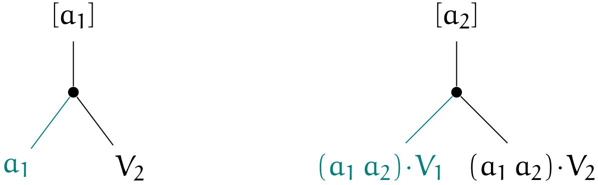
$\langle a_1, V_2 \rangle =_? (a_1\ a_2)\cdot\langle V_1, V_2 \rangle$



Substitution σ:	$\emptyset$
Freshnesses ∇:	$(a_1 \# V_1)$, $(a_1 \# V_2)$, $\emptyset$

Unification Example

$a_1 =_? (a_1\ a_2)\cdot V_1$

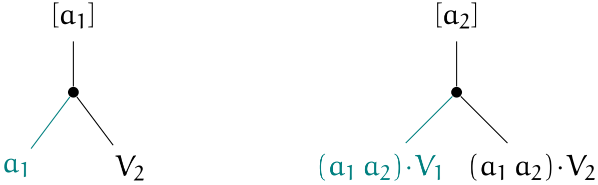


Substitution σ: $\emptyset$

Freshnesses ∇: $(a_1 \# V_1),\ (a_1 \# V_2),\ \emptyset$

Unification Example

$a_1 =_? (a_1\ a_2)\cdot V_1$



Substitution σ:	$V_1 \mapsto a_2$, $\emptyset$
Freshnesses ∇:	$(a_1 \# V_1)$, $(a_1 \# V_2)$, $\emptyset$

Unification Example

$V_2 =_? (a_1\ a_2)\cdot V_2$

Substitution σ:	$V_1 \mapsto a_2,\ \emptyset$
Freshnesses ∇:	$(a_1 \# V_1),\ (a_1 \# V_2),\ \emptyset$

Unification Example

$V_2 =_? (a_1\ a_2) \cdot V_2$

Substitution σ: $V_1 \mapsto a_2,\ \emptyset$

Freshnesses ∇: $(a_2 \# V_2),\ (a_1 \# V_1),\ (a_1 \# V_2),\ \emptyset$

Unification Example

Verify $(a_2 \# (\sigma \triangleleft V_2))$, $(a_1 \# (\sigma \triangleleft V_1))$, $(a_1 \# (\sigma \triangleleft V_2))$.

Substitution σ: $V_1 \mapsto a_2, \emptyset$

Freshnesses ∇: $(a_2 \# V_2)$, $(a_1 \# V_1)$, $(a_1 \# V_2)$, $\emptyset$

Unification Example

Verify $(a_2 \# V_2)$, $(a_1 \# a_2)$, $(a_1 \# V_2)$.

Substitution σ: $V_1 \mapsto a_2, \emptyset$

Freshnesses ∇: $(a_2 \# V_2), (a_1 \# V_1), (a_1 \# V_2), \emptyset$

Unification Example

Verify $(a_2 \# V_2)$, $(a_1 \# a_2)$, $(a_1 \# V_2)$.

Substitution σ:		$V_1 \mapsto a_2$, $\emptyset$
Freshnesses ∇:	$(a_2 \# V_2)$,	$(a_1 \# V_2)$, $\emptyset$

Unification Example

Done.

Substitution σ:		$V_1 \mapsto a_2$, $\emptyset$
Freshnesses ∇:	$(a_2 \# V_2)$,	$(a_1 \# V_2)$, $\emptyset$

Outline

Nested Recursion Under walk

```
unify1 (σ,∇) t₁ t₂ =
  case (walk σ t₁, walk σ t₂) of
    ...
    (⟨t₁₁,t₁₂⟩, ⟨t₂₁,t₂₂⟩) →
      do (σ′,∇′) ← unify1 (σ,∇) t₁₁ t₂₁;
         unify1 (σ′,∇′) t₁₂ t₂₂
    ...
```

Need a well-founded relation R such that:

- $R(((\sigma, \nabla), t_{11}, t_{21}),\ ((\sigma, \nabla), t_1, t_2))$ and
- $R(((\sigma', \nabla'), t_{12}, t_{22}),\ ((\sigma, \nabla), t_1, t_2))$ (ugh!)
- where walk $\sigma\ t_1 = \langle t_{11}, t_{12} \rangle$ and walk $\sigma\ t_2 = \langle t_{21}, t_{22} \rangle$.

Termination Relation for `unify1`

Our R, which ignores the freshness environment (∇):

$$
\begin{array}{ll}
\multicolumn{2}{l}{\texttt{UTR}((\sigma', t_{12}, t_{22}),\ (\sigma, t_1, t_2)) \iff} \\
 & \texttt{wfs}\ \sigma' \qquad (\implies \texttt{wfs}\ \sigma) \\
\wedge & \sigma \sqsubseteq \sigma' \\
\wedge & \mathcal{V}(\sigma', t_{12}, t_{22}) \subseteq \mathcal{V}(\sigma, t_1, t_2) \\
\wedge & |\sigma' \triangleleft t_{12}| < |\sigma' \triangleleft t_1|
\end{array}
$$

This is not a lexicographic combination.

Termination 1: UTR is Well-Founded

Our R, which ignores the freshness environment (∇):

$$\texttt{UTR}((\sigma', t_{12}, t_{22}),\ (\sigma, t_1, t_2)) \iff$$

$$\begin{array}{ll} & \texttt{wfs}\ \sigma' \qquad (\implies \texttt{wfs}\ \sigma) \\ \wedge & \sigma \sqsubseteq \sigma' \\ \wedge & \mathcal{V}(\sigma', t_{12}, t_{22}) \subseteq \mathcal{V}(\sigma, t_1, t_2) \\ \wedge & |\sigma' \triangleleft t_{12}| < |\sigma' \triangleleft t_1| \end{array}$$

UTR sequence: $\ldots, (\sigma'', t_{122}, t_{222}), (\sigma', t_{12}, t_{22}), (\sigma, t_1, t_2)$

- $\mathcal{V}(\sigma, t_1, t_2)$ reaches a fixpoint: finite subsets.
- σ' reaches a fixpoint: new bindings drawn from fixed set.
- Decreasing sequence of $|\sigma' \triangleleft t_1|$ must stop.

Termination 2: UTR Between Recursive Calls

Need to show:

- UTR$((\sigma, t_{11}, t_{21}),\ (\sigma, t_1, t_2))$ and
- UTR$((????, t_{12}, t_{22}),\ (\sigma, t_1, t_2))$
- where walk $\sigma\ t_1 = \langle t_{11}, t_{12} \rangle$ and walk $\sigma\ t_2 = \langle t_{21}, t_{22} \rangle$.

Difficulty:

- The $????$.

Solution:

- See the paper!

Outline

Correctness

If `unify` produces a result, it is a nominal unifier (soundness).

If the arguments are unifiable, `unify` succeeds (completeness).

If `unify` produces a result, it is most-general (generality).

Correctness in an Accumulator-Passing World

What does "soundness" (say) mean for

$$\texttt{unify}\ (\sigma, \nabla)\ t_1\ t_2 = (\sigma', \nabla')$$

For the substitutions:

$$\texttt{unify}\ (\sigma, \nabla)\ t_1\ t_2\ \simeq\ \texttt{unify}\ (\emptyset, \nabla)\ (\sigma \triangleleft t_1)\ (\sigma \triangleleft t_2)$$

The freshness environment (∇) imposes additional (potentially unsatisfiable) requirements.

Soundness

$$\vdash \ \texttt{unify}\,(\sigma, \nabla)\ t_1\ t_2 = (\sigma', \nabla') \implies$$

$$\nabla' \vdash \sigma' \triangleleft (\sigma \triangleleft t_1) \ \approx \ \sigma' \triangleleft (\sigma \triangleleft t_2)$$

Alternatively: we know that $\sigma \sqsubseteq \sigma'$, so conclusion can be

$$\nabla' \vdash \sigma' \triangleleft t_1 \ \approx \ \sigma' \triangleleft t_2$$

Completeness

$$\vdash \quad \nabla_u \vdash \sigma_u \triangleleft (\sigma \triangleleft t_1) \;\approx\; \sigma_u \triangleleft (\sigma \triangleleft t_2) \implies$$

$$\exists \sigma'. \forall \nabla.$$

$$\nabla \text{ consistent with } \sigma' \implies$$

$$\exists \nabla'. \, \texttt{unify}\, (\sigma, \nabla)\; t_1\; t_2 = (\sigma', \nabla')$$

Can simplify, by taking (universally quantified) $\nabla = \emptyset$, concluding:

$$\exists \sigma' \, \nabla'. \, \texttt{unify}\, (\sigma, \emptyset)\; t_1\; t_2 = (\sigma', \nabla')$$

Or even:

$$\exists \sigma' \, \nabla'. \, \texttt{unify}\, (\emptyset, \emptyset)\; (\sigma \triangleleft t_1)\; (\sigma \triangleleft t_2) = (\sigma', \nabla')$$

Generality

Generality of `unify`'s returned substitution:

$$\vdash \ \texttt{unify}\,(\sigma, \nabla)\ t_1\ t_2 = (\sigma', \nabla') \ \wedge$$
$$\nabla_u \vdash \sigma_u \triangleleft (\sigma \triangleleft t_1) \ \approx \ \sigma_u \triangleleft (\sigma \triangleleft t_2) \ \Longrightarrow$$
$$\exists \sigma_w.\ \forall t.$$
$$\nabla_u \vdash \sigma_w \triangleleft (\sigma' \triangleleft (\sigma \triangleleft t)) \ \approx \ \sigma_u \triangleleft (\sigma \triangleleft t)$$

The witness σ_w can be σ_u.

We also know that $\sigma' \triangleleft (\sigma \triangleleft t) = \sigma' \triangleleft t$, since $\sigma \sqsubseteq \sigma'$.

Generality

Generality of `unify`'s returned freshness environment:

$$\vdash\ \texttt{unify}\,(\sigma,\nabla)\ t_1\ t_2 = (\sigma',\nabla')\ \wedge$$

$$\nabla_u \vdash \sigma_u \lhd (\sigma \lhd t_1)\ \approx\ \sigma_u \lhd (\sigma \lhd t_2)\ \Longrightarrow$$

$$\forall (a \# V) \in \nabla'.$$

$(a \# V)$ is due to something in ∇, or

$$\nabla_u \vdash a \# (\sigma_u \lhd V)$$

("due to something" is long when captured formally.)

Conclusion

Formal treatment of triangular substitutions

- well-formedness condition, different forms of application.

Nominal unification in accumulator-passing style

- implementable, efficient.

Termination

- unusual termination relation, same as first-order case.

Correctness

- soundness, completeness, generality, for accumulator-style.