Exploiting Uniformity in Substitution: the Nuprl Term Model

Abhishek Anand

Cornell University

Variable bindings

$$\int_a^b x^2 dx \qquad\qquad \frac{d}{dx} B$$

$\lambda x.x$

```
match t with
  pair x y ⇒ B
end
```

$$\forall x : A.B \qquad\qquad \exists x : A.B$$

$$\frac{A : Type \qquad x : A \vdash B : Type}{(x : A) \to B : Type}$$

Variable bindings

$$\int_a^b x^2 dx$$

$$\frac{d}{dx} B$$

abstract over syntax

$$\lambda x.x$$

```
match t with
  pair x y ⇒ B
end
```

$$\forall x : A.B$$

$$\exists x : A.B$$

$$\frac{A : \mathit{Type} \qquad x : A \vdash B : \mathit{Type}}{(x : A) \to B : \mathit{Type}}$$

The untyped λ calculus

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
```

The untyped λ calculus

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
```

lam x b represents $\lambda x.b$

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
  lam x b represents λx.b
Fixpoint fvars (t:Term) : list Var :=
match t with
| vterm v ⇒ [v]
| app f a ⇒ fvars f ++ fvars a
| lam x b ⇒ fvars b -- [x]
```

Extensions

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
```

Extensions

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
| all : Var → Term → Term        ∀x.B
| exi : Var → Term → Term        ∃x.B
```

Extensions

```
Inductive Term : Set :=
| vterm : Var → Term
| app : Term → Term → Term
| lam : Var → Term → Term
| all : Var → Term → Term        ∀x.B
| exi : Var → Term → Term        ∃x.B

Fixpoint fvars (t:Term) : list Var :=
match t with
| vterm v ⇒ [v]
| app f a ⇒ fvars f ++ fvars a
| lam x b ⇒ fvars b -- [x]
| all x b ⇒ fvars b -- [x]
| exi x b ⇒ fvars b -- [x]
```

A Uniform Treatment

Parametrize over a collection of operators

A Uniform Treatment

Parametrize over a collection of operators

$\lambda\ x.x$ $\qquad\qquad$ $\forall x.B$

A Uniform Treatment

Parametrize over a collection of operators

$\lambda\ x.x$ $\qquad$ $\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

Parametrize over a collection of operators

$\lambda\ x.x$

$\forall x.B$

```
match t with
  pair x y => B
end
```

A Uniform Treatment

Parametrize over a collection of operators

$\lambda\ (x.x)$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

Parametrize over a collection of operators

$\lambda\ x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

A Uniform Treatment

Parametrize over a collection of operators

multiple variables
bound simultaneously

$\lambda\, x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

A Uniform Treatment

Parametrize over a collection of operators

multiple variables
bound simultaneously

$\lambda\ x.x$

$\forall x.B$

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

```
match t with
  pair x y ⇒ B
end
```

A Uniform Treatment

Parametrize over a collection of operators

multiple variables
bound simultaneously

$\lambda\ x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

multiple
bound terms

A Uniform Treatment

$\{Opid\colon \texttt{Type};\quad \text{signature}\colon\ Opid \to \text{list nat}\}$

multiple variables
bound simultaneously

$\lambda\ x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

multiple
bound terms

A Uniform Treatment

$\{Opid\colon \texttt{Type};\quad \text{signature: } Opid \to \text{list nat}\}$

multiple variables
bound simultaneously

$\lambda\ x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

[1;1]

A Uniform Treatment

$\{ \mathit{Opid} \colon \texttt{Type}; \quad \text{signature} \colon \mathit{Opid} \to \text{list nat} \}$

[2]

$\lambda\, x.x$

$\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

[1;1]

A Uniform Treatment

$\{Opid\colon \texttt{Type};\quad \text{signature: } Opid \to \text{list nat}\}$

[1]

[2]

$\lambda\ x.x$ $\qquad$ $\forall x.B$

```
match t with
  pair x y ⇒ B
end
```

```
match t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

[1;1]

A Uniform Treatment : Formally

```
Context {Var Opid : Type}.

⋮

Inductive Term : Type :=
|
|
with BTerm : Type :=
|                                  .
```

A Uniform Treatment : Formally

```
Context {Var Opid : Type}.

⋮

Inductive Term : Type :=
| vterm: Var → Term
|
with BTerm : Type :=
|                                    .
```

A Uniform Treatment : Formally

```
Context {Var Opid : Type}.

⋮

Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
|                                   .
```

A Uniform Treatment : Formally

```
Context {Var Opid : Type}.

⋮

Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

$\lambda x.y$

oterm lam [bterm [x] (vterm y)]

```
Inductive Term  : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm  → Term
with BTerm : Type :=
| bterm:  list Var → Term  → BTerm.
```

A Uniform Treatment : Formally

$\lambda x.y$

oterm lam [bterm [x] (vterm y)]

$\forall x.y$

oterm forall [bterm [x] (vterm y)]

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with
  pair x y ⇒ B
end
```

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with
  pair x y ⇒ B
end
```

oterm
  (MATCH [pair])
  [bterm [x;y] B]

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with          oterm
  | inl x ⇒ L            (MATCH [inl;inr])
  | inr x ⇒ R            [bterm [x] L;  bterm [x] R]
end
```

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

```
oterm
   (MATCH [inl;inr])
   [bterm [x] L;  bterm [x] R]
```

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with
  | inl x ⇒ L
  | inr x ⇒ R
end
```

```
oterm
  (MATCH )
  [bterm inl [x] L;
   bterm inr [x] R]
```

```
Inductive Term : Type :=
| vterm: Var → Term
| oterm: Opid → list BTerm → Term
with BTerm : Type :=
| bterm: list Var → Term → BTerm.
```

A Uniform Treatment : Formally

```
MATCH t with          oterm
  | inl x ⇒ L           (MATCH )
  | inr x ⇒ R           [bterm inl [x] L;
end                      bterm inr [x] R]
```

```
Inductive Term (Tag: Type) : Type :=
| vterm: Var → Term
| oterm: Opid → list (BTerm Tag) → Term
with BTerm (Tag: Type): Type :=
| bterm: Tag → list Var → Term Tag → BTerm.
```

```
Fixpoint fvars (t:Term): list Var :=
  match t with
  | vterm v ⇒ [v]
  | oterm op bts ⇒ flat_map fvars_bterm bts
  end
 with fvars_bterm (bt : BTerm) : list Var :=
  match bt with
  | bterm lv t ⇒ (fvars t) -- lv
  end.
```

Bound Variables

```
Fixpoint bvars (t : Term) : list Var :=
  match t with
  | vterm v ⇒ []
  | oterm op bts ⇒ flat_map bvars_bterm bts
  end
 with bvars_bterm (bt : BTerm) : list Var :=
  match bt with
  | bterm lv t ⇒ (bvars t) ++ lv
  end.
```

```
Substitution := list (Var × Term).

Fixpoint subst_aux (t : Term)
      (sub : Substitution) : Term :=
  match nt with
  | vterm var ⇒
      match sub_find sub var with
      | Some t ⇒ t
      | None ⇒ nt
      end
  | oterm op bts ⇒
      oterm op (map (subst_bterm sub) bts)
```

Unsafe substitution : bound terms

```
with subst_bterm (sub : Substitution)
    (bt : BTerm) {struct bt}: BTerm :=
  match bt with
  | bterm lv t ⇒
      bterm lv (subst_aux t (sub_filter sub lv))
  end.
```

Unsafe substitution : bound terms

```
with subst_bterm (sub : Substitution)
    (bt : BTerm) {struct bt}: BTerm :=
  match bt with
  | bterm lv t ⇒
      bterm lv (subst_aux t (sub_filter sub lv))
  end.
```

Unsafe substitution : bound terms

```
with subst_bterm (sub : Substitution)
    (bt : BTerm) {struct bt}: BTerm :=
  match bt with
  | bterm lv t ⇒
      bterm lv (subst_aux t (sub_filter sub lv))
  end.
```

```
with subst_bterm (sub : Substitution)
    (bt : BTerm) {struct bt}: BTerm :=
  match alpha_ren (sfvars sub) bt with
  | bterm lv t ⇒
        bterm lv (subst_aux t (sub_filter sub lv))
  end.
```

Unsafe substitution : bound terms

$$\texttt{with}\ \mathrm{subst\_bterm}\ (sub : \mathrm{Substitution})$$
$$(bt : \mathsf{BTerm})\ \{\texttt{struct}\ bt\} : \mathsf{BTerm} :=$$
$$\texttt{match}\ \mathrm{alpha\_ren}\ (\mathrm{sfvars}\ sub)\ bt\ \texttt{with}$$
$$\mid \mathsf{bterm}\ lv\ t \Rightarrow$$
$$\mathsf{bterm}\ lv\ (\mathrm{subst\_aux}\ t\ (\mathrm{sub\_filter}\ sub\ lv))$$
$$\texttt{end}.$$

$$\mathrm{sfvars} : \mathrm{Substitution} \to \mathsf{list}\ Var :=$$
$$\mathrm{flat\_map}\ (\mathrm{fvars} \circ \pi_2)$$

Unsafe substitution : bound terms

```
with subst_bterm (sub : Substitution)
    (bt : BTerm) {struct bt}: BTerm :=
  match alpha_ren (sfvars sub) bt with
  | bterm lv t ⇒
      bterm lv (subst_aux t (sub_filter sub lv))
  end.

sfvars : Substitution → list Var :=
  flat_map (fvars ∘ π2)
```

Capture-avoiding substitution

```
Definition subst (t : Term)
    (sub : Substitution) : Term :=
subst_aux (alpha_ren (sfvars sub) t) sub.
```

α equality

$$
\begin{array}{l}
\texttt{Inductive}\ \mathsf{alpha\_eq} : \mathrm{Term} \rightarrow \mathrm{Term} \rightarrow \texttt{Prop} := \\
\mid \mathsf{alv} : \forall\, (v{:}\mathrm{Var}),\ (\mathsf{vterm}\ v) \equiv_\alpha (\mathsf{vterm}\ v) \\
\mid \mathsf{alo} : \forall\, (op{:}\ \mathsf{Opid})\ (lbt1\ lbt2 : \mathsf{list\ BTerm}), \\
\quad \mathrm{length}\ lbt1 = \mathrm{length}\ lbt2 \\
\quad \rightarrow (\forall\, n, \\
\qquad\quad n < \mathrm{length}\ lbt1 \\
\qquad\quad \rightarrow (lbt1[n]) \equiv_\alpha (lbt2[n])) \\
\quad \rightarrow (\mathsf{oterm}\ op\ lbt1) \equiv_\alpha (\mathsf{oterm}\ op\ lbt2)
\end{array}
$$

α equality : bound terms

```
with alpha_eq_bterm : BTerm → BTerm → Prop :=
 | alb : ∀ (lv1 lv2 lv: list Var) (t1 t2 : Term) ,
     disjoint lv (all_vars t1 ++ all_vars t2)
     → length lv1 = length lv2
     → length lv1 = length lv
     → no_repeats lv
     → (subst t1 (var_ren lv1 lv))
          ≡α (subst t2 (var_ren lv2 lv))
     → (bterm lv1 t1) ≡α (bterm lv2 t2)
```

α equality : bound terms

```
with alpha_eq_bterm : BTerm → BTerm → Prop :=
 | alb : ∀ (lv1 lv2 lv: list Var) (t1 t2 : Term) ,
     disjoint lv (all_vars t1 ++ all_vars t2)
     → length lv1 = length lv2
     → length lv1 = length lv
     → no_repeats lv
     → (subst t1 (var_ren lv1 lv))
          ≡α (subst t2 (var_ren lv2 lv))
     → (bterm lv1 t1) ≡α (bterm lv2 t2)

var_ren (lvi lvf : list Var) : Substitution :=
  zip lvi (map vterm lvf).
```

Proved Properties

`Context` ($a\ b$: Term) ($sa\ sb\ s$: $\mathrm{Substitution}$).

Proved Properties

`Context` $(a\ b : \mathsf{Term})\ (sa\ sb\ s : \mathrm{Substitution})$.

$a =_\alpha b \to sa =_\alpha sb \to \mathrm{subst}\ a\ sa =_\alpha \mathrm{subst}\ b\ sb$.

Proved Properties

$\texttt{Context}\ (a\ b : \mathsf{Term})\ (sa\ sb\ s : \mathrm{Substitution}).$

$a =_\alpha b \to sa =_\alpha sb \to \mathrm{subst}\ a\ sa =_\alpha \mathrm{subst}\ b\ sb.$

$\mathrm{fvars}\ (\mathrm{subst}\ a\ s)$
$\approx \texttt{(}\mathrm{fvars}\ a\ \texttt{--}\ \mathrm{Dom}\ s\texttt{)}\ \texttt{++}\ \mathrm{sfvars}\ (\mathrm{keepFirst}\ s\ (\mathrm{fvars}\ a)).$

Proved Properties

Context $(a\ b : \mathrm{Term})\ (sa\ sb\ s : \mathrm{Substitution})$.

$a =_\alpha b \to sa =_\alpha sb \to \mathrm{subst}\ a\ sa =_\alpha \mathrm{subst}\ b\ sb$.

$\mathrm{fvars}\ (\mathrm{subst}\ a\ s)$
$\approx (\mathrm{fvars}\ a\ \text{--}\ \mathrm{Dom}\ s)\ {+}{+}\ \mathrm{sfvars}\ (\mathrm{keepFirst}\ s\ (\mathrm{fvars}\ a))$.

$\mathrm{disjoint}\ (\mathrm{sfvars}\ sa)\ (\mathrm{Dom}\ sb)$
$\to \mathrm{disjoint}\ (\mathrm{sfvars}\ sb)\ (\mathrm{Dom}\ sa)$
$\to \mathrm{disjoint}\ (\mathrm{Dom}\ sa)\ (\mathrm{Dom}\ sb)$
$\to \mathrm{subst}\ (\mathrm{subst}\ a\ sa)\ sb =_\alpha \mathrm{subst}\ (\mathrm{subst}\ a\ sb)\ sa$.

Proved Properties

$\texttt{Context}\ (a\ b : \mathrm{Term})\ (sa\ sb\ s : \mathrm{Substitution}).$

$a =_\alpha b \to sa =_\alpha sb \to \mathrm{subst}\ a\ sa =_\alpha \mathrm{subst}\ b\ sb.$

$\mathrm{fvars}\ (\mathrm{subst}\ a\ s)$
$\approx (\mathrm{fvars}\ a \texttt{ -- } \mathrm{Dom}\ s) \texttt{ ++ } \mathrm{sfvars}\ (\mathrm{keepFirst}\ s\ (\mathrm{fvars}\ a)).$

$\mathrm{disjoint}\ (\mathrm{sfvars}\ sa)\ (\mathrm{Dom}\ sb)$
$\to \mathrm{disjoint}\ (\mathrm{sfvars}\ sb)\ (\mathrm{Dom}\ sa)$
$\to \mathrm{disjoint}\ (\mathrm{Dom}\ sa)\ (\mathrm{Dom}\ sb)$
$\to \mathrm{subst}\ (\mathrm{subst}\ a\ sa)\ sb =_\alpha \mathrm{subst}\ (\mathrm{subst}\ a\ sb)\ sa.$

$\mathrm{subst}\ (\mathrm{subst}\ a\ sa)\ sb$
$=_\alpha \mathrm{subst}\ a\ ((\mathrm{SubstSub}\ sa\ sb)\texttt{++}sb).$

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};


}.
```

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};
  freshVars : ∀ (n:nat) (avoid : list Var), list Var;

}.
```

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};
  freshVars : ∀ (n:nat) (avoid : list Var), list Var;
  freshCorrect: ...
}.
```

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};


}.
```

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};


}.
```

free variables, bound variables, unsafe substitution,

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};


}.
```

free variables, bound variables, unsafe substitution, ~~safe substitution~~

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};



}.
```

free variables, bound variables, unsafe substitution, ~~safe substitution~~ ,alpha equality

Variables

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};



}.
```

free variables, bound variables, unsafe substitution, ~~safe substitution~~ ,alpha equality

equivariance : $\forall \pi, \pi\ (f\ x) = f\ (\pi\ x)$

Andrew M. Pitts. "Nominal logic: A first order theory of names and binding". In: *TACS*. Springer, 2001

α equality of bound terms: simpler definition

```
with alpha_eq_bterm : BTerm → BTerm → Prop :=
 | alb : ∀ (lv1 lv2 lv: list Var) (t1 t2 : Term) ,
     disjoint lv (all_vars t1 ++ all_vars t2)
     → length lv1 = length lv2
     → length lv1 = length lv
     → no_repeats lv
     → (subst t1 (var_ren lv1 lv))
          ≡α (subst t2 (var_ren lv2 lv))
     → (bterm lv1 t1) ≡α (bterm lv2 t2)
```

α equality of bound terms: simpler definition

```
with alpha_eq_bterm : BTerm → BTerm → Prop :=
 | alb : ∀ (lv1 lv2 lv: list Var) (t1 t2 : Term) ,
     disjoint lv (all_vars t1 ++ all_vars t2)
     → length lv1 = length lv2
     → length lv1 = length lv
     → no_repeats lv
     → (subst_aux t1 (var_ren lv1 lv))
          ≡α (subst_aux t2 (var_ren lv2 lv))
     → (bterm lv1 t1) ≡α (bterm lv2 t2)
```

α equality of bound terms: simpler definition

```
with alpha_eq_bterm : BTerm → BTerm → Prop :=
 | alb : ∀ (lv1 lv2 lv: list Var) (t1 t2 : Term) ,
     disjoint lv (all_vars t1 ++ all_vars t2)
     → length lv1 = length lv2
     → length lv1 = length lv
     → no_repeats lv
     →  (π(lv1,lv) t1)
          ≡α  (π(lv2,lv) t2)
     → (bterm lv1 t1) ≡α (bterm lv2 t2)
```

Beyond Substitution

- $t \Downarrow_n v$
- defines $\sim$, a computational equivalence
- conditions for congruence of $\sim$

Douglas J. Howe. "Equality in Lazy Computation Systems". In: *LICS*. 1989

Douglas J. Howe. "Proving Congruence of Bisimulation in Functional Programming Languages". In: *Inf. Comput.* 124.2 (1996)

Limitations

- non-linear structure in pattern variables

Limitations

- non-linear structure in pattern variables

```
Inductive btree: Set :=
  leaf : btree | bnode : btree → btree → btree.


Fixpoint btsize (bt: btree) :nat :=
match bt with
  | leaf ⇒ 1
  | bnode t leaf ⇒ (btsize t)+1
  | bnode t (bnode ta tb)
  ⇒ (btsize t)+(btsize ta)+(btsize tb)
end.
```

Limitations

- non-linear structure in pattern variables

```
Inductive btree: Set :=
   leaf : btree | bnode : btree → btree → btree.


Fixpoint btsize (bt: btree) :nat :=
match bt with
   | leaf ⇒ 1
   | bnode t leaf ⇒ (btsize t)+1
   | bnode t (bnode ta tb)
   ⇒ (btsize t)+(btsize ta)+(btsize tb)
end.
```

Limitations

- non-linear structure in pattern variables

  CFGV:

```
Fixpoint btsize (bt: btree) :nat :=
match bt with
  | leaf ⇒ 1
  | bnode t leaf ⇒ (btsize t)+1
  | bnode t (bnode ta tb)
  ⇒ (btsize t)+(btsize ta)+(btsize tb)
end.
```

Limitations

- non-linear structure in pattern variables

  CFGV: Abhishek Anand and Vincent Rahli. "A Generic Approach to Proofs about Substitution". In: LFMTP. Vienna, Austria: ACM, July 2014

```
Fixpoint btsize (bt: btree) :nat :=
match bt with
  | leaf ⇒ 1
  | bnode t leaf ⇒ (btsize t)+1
  | bnode t (bnode ta tb)
  ⇒ (btsize t)+(btsize ta)+(btsize tb)
end.
```

Limitations

- non-linear structure in pattern variables

  CFGV: Abhishek Anand and Vincent Rahli. "A Generic Approach to Proofs about Substitution". In: LFMTP. Vienna, Austria: ACM, July 2014

$$t := x \mid \lambda x : t.t \mid (t\ t)$$

Limitations

- non-linear structure in pattern variables

  CFGV: Abhishek Anand and Vincent Rahli. "A Generic Approach to Proofs about Substitution". In: LFMTP. Vienna, Austria: ACM, July 2014

$$t := x \mid \lambda x : t.t \mid (t\ t)$$

- non-linear structure in pattern variables

  CFGV: Abhishek Anand and Vincent Rahli. "A Generic Approach to Proofs about Substitution". In: LFMTP. Vienna, Austria: ACM, July 2014

$$t := x \mid \lambda x : t.t \mid (t\ t)$$

- complex restrictions: e.g. A-normal form

Future Work

- Automation

  disjoint _ _

  subset _ _

  Domenico Cantone, Eugenio Omodeo, and Alberto Policriti.
  Set Theory for Computing. Monographs in Computer Science. New York, NY: Springer New York, 2001

Future Work

- Automation

  disjoint _ _

  subset _ _

  Domenico Cantone, Eugenio Omodeo, and Alberto Policriti. *Set Theory for Computing*. Monographs in Computer Science. New York, NY: Springer New York, 2001

- De Bruijn Indices

Conclusion

- an abstract representation of terms
- define substitution and prove its properties *once and for all*
- `https://github.com/aa755/SquiggleEq`

Rewriting Support

$x =_\alpha y$

Rewriting Support

$x =_\alpha y$
$\mathrm{disjoint}\ (\mathrm{fvars}\ (\mathrm{subst}\ x\ \mathit{sub}))\ \mathit{lv}$

Rewriting Support

$x =_\alpha y$
$\mathrm{disjoint}\ (\mathrm{fvars}\ (\mathrm{subst}\ y\ sub))\ lv$

Well-formedness of terms

```
Class GenericTermSig : Type :=
{
  Opid : Set;
  OpBindings : Opid → list nat;
  opid_dec : ∀ x y : Opid, {x = y} + {x ≠ y};
}.
```

Well-formedness : proved properties

$\lambda(x.y)$

```
oterm lam [bterm [x] (vterm y)                    ]
Inductive nt_wf: Term → Prop:=
| wfvt: ∀ nv : Var, nt_wf (vterm nv)
| wfot: ∀ (o: Opid) (lbt: list BTerm),
     (∀ b, b ∈ lbt → bt_wf b)
     → map (num_bvars) lbt = OpBindings o
     → nt_wf (oterm o lbt)
with bt_wf : BTerm → Prop :=
| wfbt : ∀ (lv : list Var) (t: Term),
     nt_wf nt → bt_wf (bterm lv nt).
```

Well-formedness : proved properties

$\lambda(x.y)$

oterm lam [bterm [x] (vterm y)bterm [z] (vterm w)]

Inductive nt_wf: Term → Prop:=
| wfvt: ∀ *nv* : Var, nt_wf (vterm *nv*)
| wfot: ∀ (*o*: Opid) (*lbt*: list BTerm),
    (∀ *b*, *b* ∈ *lbt* → bt_wf *b*)
    → map (num_bvars) *lbt* = OpBindings *o*
    → nt_wf (oterm *o* *lbt*)
with bt_wf : BTerm → Prop :=
| wfbt : ∀ (*lv* : list Var) (*t*: Term),
    nt_wf *nt* → bt_wf (bterm *lv* *nt*).

Well-formedness : proved properties

$\lambda(x.y)(z.w)$

oterm lam [bterm [x] (vterm y)]

```
Inductive nt_wf: Term → Prop:=
| wfvt: ∀ nv : Var, nt_wf (vterm nv)
| wfot: ∀ (o: Opid) (lbt: list BTerm),
    (∀ b, b ∈ lbt → bt_wf b)
    → map (num_bvars) lbt = OpBindings o
    → nt_wf (oterm o lbt)
with bt_wf : BTerm → Prop :=
| wfbt : ∀ (lv : list Var) (t: Term),
    nt_wf nt → bt_wf (bterm lv nt).
```

VarType

```
Class VarType (Var :Type) := {
  deq : ∀ x y : Var, {x = y} + {x ≠ y};
  freshVars : ∀ (n:nat) (avoid : list Var), list Var;
  freshCorrect: ∀ (n:nat) (avoid : list Var),
    let lf:= (freshVars n avoid) in
      no_repeats lf ∧ disjoint lf avoid ∧ length lf = n
}.
```