Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

Łukasz Czajka

University of Innsbruck

7 November 2015

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

1. Bracket abstraction.

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

1. Bracket abstraction.
2. Experimental results.

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

- Goal: translating HOL to first-order logic.

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

- Goal: translating HOL to first-order logic.
- Requires elimination of lambda-abstraction.

Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions

- Goal: translating HOL to first-order logic.
- Requires elimination of lambda-abstraction.
- We replace lambda-lifting in HOLyHammer with various bracket abstraction algorithms and see how this affects the success rate of first-order ATPs on translated problems.

Bracket abstraction

Univariate ($[]_A. : V \times \mathrm{CL}(B) \to \mathrm{CL}(B)$) or multivariate ($[]_A. : V^n \times \Lambda \to \mathrm{CL}(B)$).

Bracket abstraction

Univariate ($[]_A. : V \times \mathrm{CL}(B) \to \mathrm{CL}(B)$) or multivariate ($[]_A. : V^n \times \Lambda \to \mathrm{CL}(B)$).
For univariate – induced translation:

$$\begin{aligned} \mathrm{H}_A(x) &= x \\ \mathrm{H}_A(st) &= (\mathrm{H}_A(s))(\mathrm{H}_A(t)) \\ \mathrm{H}_A(\lambda x.t) &= [x]_A.\mathrm{H}_A(t) \end{aligned}$$

Bracket abstraction

Univariate ($[]_A. : V \times \mathrm{CL}(B) \to \mathrm{CL}(B)$) or multivariate ($[]_A. : V^n \times \Lambda \to \mathrm{CL}(B)$).
For univariate – induced translation:

$$\begin{aligned} \mathrm{H}_A(x) &= x \\ \mathrm{H}_A(st) &= (\mathrm{H}_A(s))(\mathrm{H}_A(t)) \\ \mathrm{H}_A(\lambda x.t) &= [x]_A.\mathrm{H}_A(t) \end{aligned}$$

For multivariate: $\mathrm{H}_A t = []_A.t$ and
$[x_1, \ldots, x_n]_A.\lambda y.t = [x_1, \ldots, x_n, y]_A.t$.

Bracket abstraction

The simple algorithm (fab) of Curry

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$$

Bracket abstraction

The simple algorithm (fab) of Curry

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$$

$$\begin{aligned} \mathsf{S} &= \lambda xyz.xz(yz) \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{I} &= \lambda x.x \end{aligned}$$

Bracket abstraction

The simple algorithm (fab) of Curry

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$$

$$\begin{aligned} \mathsf{S} &= \lambda xyz.xz(yz) \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{I} &= \lambda x.x \end{aligned}$$

$$\begin{aligned} [x]_{(\mathrm{fab})}.st &= \mathsf{S}([x]_{(\mathrm{fab})}.s)([x]_{(\mathrm{fab})}.t) \\ [x]_{(\mathrm{fab})}.x &= \mathsf{I} \\ [x]_{(\mathrm{fab})}.t &= \mathsf{K}t \end{aligned}$$

Bracket abstraction

The simple algorithm (fab) of Curry

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$$

$$\begin{aligned} \mathsf{S} &= \lambda xyz.xz(yz) \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{I} &= \lambda x.x \end{aligned}$$

$$\begin{aligned} [x]_{(\mathrm{fab})}.st &= \mathsf{S}([x]_{(\mathrm{fab})}.s)([x]_{(\mathrm{fab})}.t) \\ [x]_{(\mathrm{fab})}.x &= \mathsf{I} \\ [x]_{(\mathrm{fab})}.t &= \mathsf{K}t \end{aligned}$$

Worst case translation size: exponential in the size of the input term.

Bracket abstraction

A slightly better algorithm (abf)

$$
\begin{array}{rcll}
[x]_{(\mathrm{abf})}.x & = & \mathsf{I} & \\
[x]_{(\mathrm{abf})}.t & = & \mathsf{K}t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_{(\mathrm{abf})}.st & = & \mathsf{S}([x]_{(\mathrm{abf})}.s)([x]_{(\mathrm{abf})}.t) &
\end{array}
$$

Bracket abstraction

A slightly better algorithm (abf)

$$
\begin{array}{rcll}
[x]_{(\mathrm{abf})}.x & = & \mathsf{I} & \\
[x]_{(\mathrm{abf})}.t & = & \mathsf{K}t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_{(\mathrm{abf})}.st & = & \mathsf{S}([x]_{(\mathrm{abf})}.s)([x]_{(\mathrm{abf})}.t) &
\end{array}
$$

Worst case translation size: $O(n^3)$.

Bracket abstraction

Schönfinkel's algorithm S

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}\}$$

Bracket abstraction

Schönfinkel's algorithm S

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}\}$$

$$\begin{aligned} \mathsf{B} &= \lambda xyz.x(yz) \\ \mathsf{C} &= \lambda xyz.xzy \end{aligned}$$

Bracket abstraction

Schönfinkel's algorithm S

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}\}$$

$$\begin{aligned} \mathsf{B} &= \lambda xyz.x(yz) \\ \mathsf{C} &= \lambda xyz.xzy \end{aligned}$$

$$\begin{aligned} [x]_S.t &= \mathsf{K}t && \text{if } x \notin \mathrm{FV}(t) \\ [x]_S.x &= \mathsf{I} \\ [x]_S.sx &= s && \text{if } x \notin \mathrm{FV}(s) \\ [x]_S.st &= \mathsf{B}s([x]_S.t) && \text{if } x \notin \mathrm{FV}(s) \\ [x]_S.st &= \mathsf{C}([x]_S.s)t && \text{if } x \notin \mathrm{FV}(t) \\ [x]_S.st &= \mathsf{S}([x]_S.s)([x]_S.t) \end{aligned}$$

Bracket abstraction

Schönfinkel's algorithm S

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}\}$$

$$\begin{aligned} \mathsf{B} &= \lambda xyz.x(yz) \\ \mathsf{C} &= \lambda xyz.xzy \end{aligned}$$

$$\begin{aligned} [x]_S.t &= \mathsf{K}t && \text{if } x \notin \mathrm{FV}(t) \\ [x]_S.x &= \mathsf{I} \\ [x]_S.sx &= s && \text{if } x \notin \mathrm{FV}(s) \\ [x]_S.st &= \mathsf{B}s([x]_S.t) && \text{if } x \notin \mathrm{FV}(s) \\ [x]_S.st &= \mathsf{C}([x]_S.s)t && \text{if } x \notin \mathrm{FV}(t) \\ [x]_S.st &= \mathsf{S}([x]_S.s)([x]_S.t) \end{aligned}$$

Worst case translation size: $O(n^3)$ but with a smaller constant than (abf).

Bracket abstraction

Turner's algorithm T

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{S}', \mathsf{B}', \mathsf{C}'\}$$

Bracket abstraction

Turner's algorithm T

$$B = \{\mathsf{S}, \mathsf{K}, \mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{S}', \mathsf{B}', \mathsf{C}'\}$$

$$\begin{aligned} \mathsf{S}' &= \lambda kxyz.k(xz)(yz) \\ \mathsf{B}' &= \lambda kxyz.kx(yz) \\ \mathsf{C}' &= \lambda kxyz.k(xz)y \end{aligned}$$

Bracket abstraction

Turner's algorithm T

$$
\begin{array}{rcll}
[x]_T.t & = & \mathsf{K}t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_T.x & = & \mathsf{I} & \\
[x]_T.sx & = & s & \text{if } x \notin \mathrm{FV}(s) \\
[x]_T.uxt & = & \mathsf{C}ut & \text{if } x \notin \mathrm{FV}(ut) \\
[x]_T.uxt & = & \mathsf{S}u([x]_T.t) & \text{if } x \notin \mathrm{FV}(u) \\
[x]_T.ust & = & \mathsf{B}'us([x]_T.t) & \text{if } x \notin \mathrm{FV}(us) \\
[x]_T.ust & = & \mathsf{C}'u([x]_T.s)t & \text{if } x \notin \mathrm{FV}(ut) \\
[x]_T.ust & = & \mathsf{S}'u([x]_T.s)([x]_T.t) & \text{if } x \notin \mathrm{FV}(u) \\
[x]_T.st & = & \mathsf{B}s([x]_T.t) & \text{if } x \notin \mathrm{FV}(s) \\
[x]_T.st & = & \mathsf{C}([x]_T.s)t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_T.st & = & \mathsf{S}([x]_T.s)([x]_T.t) &
\end{array}
$$

Bracket abstraction

Turner's algorithm T

$$
\begin{array}{rcll}
[x]_T.t & = & \mathsf{K}t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_T.x & = & \mathsf{I} & \\
[x]_T.sx & = & s & \text{if } x \notin \mathrm{FV}(s) \\
[x]_T.uxt & = & \mathsf{C}ut & \text{if } x \notin \mathrm{FV}(ut) \\
[x]_T.uxt & = & \mathsf{S}u([x]_T.t) & \text{if } x \notin \mathrm{FV}(u) \\
[x]_T.ust & = & \mathsf{B}'us([x]_T.t) & \text{if } x \notin \mathrm{FV}(us) \\
[x]_T.ust & = & \mathsf{C}'u([x]_T.s)t & \text{if } x \notin \mathrm{FV}(ut) \\
[x]_T.ust & = & \mathsf{S}'u([x]_T.s)([x]_T.t) & \text{if } x \notin \mathrm{FV}(u) \\
[x]_T.st & = & \mathsf{B}s([x]_T.t) & \text{if } x \notin \mathrm{FV}(s) \\
[x]_T.st & = & \mathsf{C}([x]_T.s)t & \text{if } x \notin \mathrm{FV}(t) \\
[x]_T.st & = & \mathsf{S}([x]_T.s)([x]_T.t) &
\end{array}
$$

Worst case translation size: $O(n^2)$.

Bracket abstraction

Turner's algorithm T

$$x_0, x_1, x_2, x_3 \in \mathrm{FV}(s) \cap \mathrm{FV}(t)$$

Bracket abstraction

Turner's algorithm T

$$x_0, x_1, x_2, x_3 \in \mathrm{FV}(s) \cap \mathrm{FV}(t)$$

$$[x_3]_T.st \quad = \quad \mathsf{S}([x_3]_T.s)([x_3]_T.t)$$

Bracket abstraction

Turner's algorithm T

$$x_0, x_1, x_2, x_3 \in \mathrm{FV}(s) \cap \mathrm{FV}(t)$$

$$\begin{array}{rcl} [x_3]_T.st & = & \mathsf{S}([x_3]_T.s)([x_3]_T.t) \\ [x_2, x_3]_T.st & = & \mathsf{S}'\mathsf{S}([x_2, x_3]_T.s)([x_2, x_3]_T.t) \end{array}$$

Bracket abstraction

Turner's algorithm T

$$x_0, x_1, x_2, x_3 \in \mathrm{FV}(s) \cap \mathrm{FV}(t)$$

$$\begin{aligned}
[x_3]_T.st &= \mathsf{S}([x_3]_T.s)([x_3]_T.t) \\
[x_2, x_3]_T.st &= \mathsf{S}'\mathsf{S}([x_2, x_3]_T.s)([x_2, x_3]_T.t) \\
[x_1, x_2, x_3]_T.st &= \mathsf{S}'(\mathsf{S}'\mathsf{S})([x_1, x_2, x_3]_T.s)([x_1, x_2, x_3]_T.t)
\end{aligned}$$

Bracket abstraction

Turner's algorithm T

$$x_0, x_1, x_2, x_3 \in \mathrm{FV}(s) \cap \mathrm{FV}(t)$$

$$\begin{aligned}
[x_3]_T.st &= \mathsf{S}([x_3]_T.s)([x_3]_T.t) \\
[x_2, x_3]_T.st &= \mathsf{S}'\mathsf{S}([x_2, x_3]_T.s)([x_2, x_3]_T.t) \\
[x_1, x_2, x_3]_T.st &= \mathsf{S}'(\mathsf{S}'\mathsf{S})([x_1, x_2, x_3]_T.s)([x_1, x_2, x_3]_T.t) \\
[x_0, x_1, x_2, x_3]_T.st &= \mathsf{S}'(\mathsf{S}'(\mathsf{S}'\mathsf{S}))([x_0, x_1, x_2, x_3]_T.s)([x_0, x_1, x_2, x_3]_T.t)
\end{aligned}$$

Director strings

- Directors: ^ / \ -.

Director strings

- Directors: ^ / \ -.
- Application nodes decorated with strings of directors.

Director strings

- Directors: ^ / \ -.
- Application nodes decorated with strings of directors.
- Abstraction algorithm (informal):

$$
\begin{array}{rcl}
\lambda x.M_x N_x & \rightarrow & \hat{}((\lambda x.M_x)(\lambda x.N_x)) \\
\lambda x.M_x N & \rightarrow & /((\lambda x.M_x)N) \\
\lambda x.M N_x & \rightarrow & \backslash(M(\lambda x.N_x)) \\
\lambda x.MN & \rightarrow & -(MN) \\
\lambda x.x & \rightarrow & \mathsf{I} \\
\lambda x.y & \rightarrow & \mathsf{K}y
\end{array}
$$

Director strings

Examples

USE THE BLACKBOARD

Lambda-lifting

Supercombinators

Definition

A *supercombinator* is a closed lambda-term of the form $\lambda x_1 \ldots x_n.t$ where $n \geq 0$, the term t is not a lambda-abstraction, and each lambda-abstraction appearing in t is a supercombinator.

Lambda-lifting

Lambda-lifting L

$B =$ all supercombinators

Lambda-lifting

Lambda-lifting L

$B =$ all supercombinators

$$
\begin{aligned}
\mathrm{H}_L(x) &= x \\
\mathrm{H}_L(t_1\, t_2) &= (\mathrm{H}_L(t_1))(\mathrm{H}_L(t_2)) \\
[x_1, \ldots, x_m]_L.\lambda y.t &= [x_1, \ldots, x_m, y]_L.t \\
[x_1, \ldots, x_m]_L.t &= (\lambda y_1 \ldots y_n x_1 \ldots x_m.\mathrm{H}_L(t))\, y_1 \ldots y_n \\
&\quad \text{where } \{y_1, \ldots, y_n\} = \mathrm{FV}(t) \setminus \{x_1, \ldots, x_m\}
\end{aligned}
$$

Abdali's algorithm

Abdali's (modified) algorithm A

$$B = \{\mathsf{B}_n^m, \mathsf{E}_n^m, \mathsf{S}_n^{k,m} \mid n, m, k \in \mathbb{N}\}$$

Abdali's algorithm

Abdali's (modified) algorithm A

$$B = \{\mathsf{B}_n^m, \mathsf{E}_n^m, \mathsf{S}_n^{k,m} \mid n, m, k \in \mathbb{N}\}$$

$$\begin{aligned}
\mathsf{B}_n^m &= \lambda x y_1 \ldots y_n z_1 \ldots z_m . x(y_1 z_1 \ldots z_m) \ldots (y_n z_1 \ldots z_m) \\
\mathsf{S}_n^{k,m} &= \lambda y_1 \ldots y_n z_1 \ldots z_k u v_1 \ldots v_m . u(y_1 z_1 \ldots z_k u v_1 \ldots v_m) \ldots (y_n z_1 \ldots z_k u v_1 \ldots v_m) \\
\mathsf{E}_n^m &= \lambda x y_1 \ldots y_n z_1 \ldots z_m . x z_1 \ldots z_m (y_1 z_1 \ldots z_m) \ldots (y_n z_1 \ldots z_m)
\end{aligned}$$

Abdali's algorithm

Abdali's (modified) algorithm A

$$
\begin{array}{rcl}
\mathrm{H}_{\boldsymbol{A}}(x) & = & x \\
\mathrm{H}_{\boldsymbol{A}}(t_1\, t_2) & = & (\mathrm{H}_{\boldsymbol{A}}(t_1))(\mathrm{H}_{\boldsymbol{A}}(t_2)) \\
\mathrm{H}_{\boldsymbol{A}}(\lambda x.t) & = & [x]_{\boldsymbol{A}}.t \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.t & = & \mathsf{B}_0^i([x_i, \ldots, x_m]_{\boldsymbol{A}}.t) \\
 & & \text{if } i > 1, x_i \in \mathrm{FV}(t), x_j \notin \mathrm{FV}(t) \text{ for } j < i \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.ux_m & = & [x_1, \ldots, x_{m-1}]_{\boldsymbol{A}}.u \\
 & & \text{if } x_m \notin \mathrm{FV}(u) \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.ux_m t_1 \ldots t_n & = & \mathsf{E}_n^m([x_1, \ldots, x_{m-1}]_{\boldsymbol{A}}.u)([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_1) \ldots ([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_n) \\
 & & \text{if } x_m \notin \mathrm{FV}(u) \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.x_i t_1 \ldots t_n & = & \mathsf{S}_n^{i-1,m-i}([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_1) \ldots ([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_n) \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.st_1 \ldots t_n & = & \mathsf{B}_n^m(\mathrm{H}_{\boldsymbol{A}}(s))([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_1) \ldots ([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_n) \\
 & & \text{if } \{x_1, \ldots, x_m\} \cap \mathrm{FV}(s) = \emptyset, \{x_1, \ldots, x_m\} \cap \mathrm{FV}(t_1) \neq \emptyset \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.\lambda y.s & = & [x_1, \ldots, x_m, y]_{\boldsymbol{A}}.s \\
[x_1, \ldots, x_m]_{\boldsymbol{A}}.(\lambda y.s)t_1 \ldots t_n & = & \mathsf{E}_n^m([x_1, \ldots, x_m, y]_{\boldsymbol{A}}.s)([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_1) \ldots ([x_1, \ldots, x_m]_{\boldsymbol{A}}.t_n)
\end{array}
$$

Abdali + director strings

Algorithm *D*

$$B = \{\mathsf{D}_l, \mathsf{U}_l^k \mid k \in \mathbb{N}, l \in L_m, m \in \mathbb{N}\}$$

Abdali + director strings

Algorithm D

$$B = \{\mathsf{D}_l, \mathsf{U}_l^k \mid k \in \mathbb{N}, l \in L_m, m \in \mathbb{N}\}$$

$$\begin{aligned}
\mathsf{D}_{\langle \nu_0, \ldots, \nu_n \rangle m} &= \lambda x y_1 \ldots y_n z_1 \ldots z_m . x \{z_1\}^{\nu_0(1)} \ldots \{z_m\}^{\nu_0(m)} (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1 \\
\mathsf{U}^k_{\langle \nu_1, \ldots, \nu_n \rangle m} &= \lambda y_1 \ldots y_n z_1 \ldots z_m . z_k (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1\}^{\nu_n(1)} \ldots \{z_m\}^{\nu_n(m)})
\end{aligned}$$

Abdali + director strings

Algorithm *D*

$$B = \{\mathsf{D}_l, \mathsf{U}_l^k \mid k \in \mathbb{N}, l \in L_m, m \in \mathbb{N}\}$$

$$\begin{aligned}
\mathsf{D}_{\langle \nu_0, \ldots, \nu_n \rangle m} &= \lambda x y_1 \ldots y_n z_1 \ldots z_m . x \{z_1\}^{\nu_0(1)} \ldots \{z_m\}^{\nu_0(m)} (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1 \\
\mathsf{U}^k_{\langle \nu_1, \ldots, \nu_n \rangle m} &= \lambda y_1 \ldots y_n z_1 \ldots z_m . z_k (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1\}^{\nu_n(1)} \ldots \{z_m\}^{\nu_n(m)})
\end{aligned}$$

The notation $\{t\}^b$ should be treated as t if $b = 1$, or omitted if $b = 0$.

Abdali + director strings

Algorithm *D*

$$B = \{\mathsf{D}_l, \mathsf{U}_l^k \mid k \in \mathbb{N}, l \in L_m, m \in \mathbb{N}\}$$

$$\begin{array}{lcl} \mathsf{D}_{\langle \nu_0, \ldots, \nu_n \rangle m} & = & \lambda x y_1 \ldots y_n z_1 \ldots z_m . x \{z_1\}^{\nu_0(1)} \ldots \{z_m\}^{\nu_0(m)} (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1 \\ \mathsf{U}^k_{\langle \nu_1, \ldots, \nu_n \rangle m} & = & \lambda y_1 \ldots y_n z_1 \ldots z_m . z_k (y_1 \{z_1\}^{\nu_1(1)} \ldots \{z_m\}^{\nu_1(m)}) \ldots (y_n \{z_1\}^{\nu_n(1)} \ldots \{z_m\}^{\nu_n(m)}) \end{array}$$

The notation $\{t\}^b$ should be treated as t if $b = 1$, or omitted if $b = 0$.

$$\mathsf{D}_{\langle 010,101 \rangle_3} = \lambda x y z_1 z_2 z_3 . x z_2 (y z_1 z_3).$$

Abdali + director strings

Algorithm D

$$
\begin{array}{rcl}
\mathrm{H}_{\boldsymbol{D}}(x) & = & x \\
\mathrm{H}_{\boldsymbol{D}}(t_1 t_2) & = & (\mathrm{H}_{\boldsymbol{D}}(t_1))(\mathrm{H}_{\boldsymbol{D}}(t_2)) \\
\mathrm{H}_{\boldsymbol{D}}(\lambda x.t) & = & [x]_{\boldsymbol{D}}.t \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.t & = & \mathrm{D}_{\langle 0^m\rangle_m}([x_i,\ldots,x_m]_{\boldsymbol{D}}.t) \\
 & & \text{if } i>1, x_i \in \mathrm{FV}(t), x_j \notin \mathrm{FV}(t) \text{ for } j<i \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.ux_m & = & [x_1,\ldots,x_{m-1}]_{\boldsymbol{D}}.u \\
 & & \text{if } x_m \notin \mathrm{FV}(u) \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.ux_m t_1\ldots t_n & = & \mathrm{D}_{\langle \nu_0,\nu_1,\ldots,\nu_n\rangle_m} u' t'_1\ldots t'_n \\
 & & \text{where } x_m \notin \mathrm{FV}(u), \\
 & & u' = [\{x_1\}^{\nu_0(1)},\ldots,\{x_{m-1}\}^{\nu_0(m-1)}]_{\boldsymbol{D}}.u, \\
 & & t'_i = [\{x_1\}^{\nu_i(1)},\ldots,\{x_m\}^{\nu_i(m)}]_{\boldsymbol{D}}.t_i \text{ for } i=1,\ldots,n \\
 & & \nu_0(m)=1, \nu_0(j)=1 \text{ iff } x_j \in \mathrm{FV}(u), \text{ for } j=1,\ldots,m-1 \\
 & & \nu_i(j)=1 \text{ iff } x_j \in \mathrm{FV}(t_i), \text{ for } i=1,\ldots,n, j=1,\ldots,m, \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.x_i t_1\ldots t_n & = & \mathrm{U}^i_{\langle \nu_1,\ldots,\nu_n\rangle_m} t'_1\ldots t'_n \\
 & & \text{where } t'_i = [\{x_1\}^{\nu_i(1)},\ldots,\{x_m\}^{\nu_i(m)}]_{\boldsymbol{D}}.t_i \text{ for } i=1,\ldots,n \\
 & & \nu_i(j)=1 \text{ iff } x_j \in \mathrm{FV}(t_i), \text{ for } i=1,\ldots,n, j=1,\ldots,m, \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.st_1\ldots t_n & = & \mathrm{D}_{\langle 0^m,\nu_1,\ldots,\nu_n\rangle_m}(\mathrm{H}_{\boldsymbol{D}}(s))t'_1\ldots t'_m \\
 & & \text{where } \{x_1,\ldots,x_m\}\cap \mathrm{FV}(s)=\emptyset, \\
 & & \{x_1,\ldots,x_m\}\cap \mathrm{FV}(t_1)\neq\emptyset, \\
 & & t'_i = [\{x_1\}^{\nu_i(1)},\ldots,\{x_m\}^{\nu_i(m)}]_{\boldsymbol{D}}.t_i \text{ for } i=1,\ldots,n \\
 & & \nu_i(j)=1 \text{ iff } x_j \in \mathrm{FV}(t_i), \text{ for } i=1,\ldots,n, j=1,\ldots,m, \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.\lambda y.s & = & [x_1,\ldots,x_m,y]_{\boldsymbol{D}}.s \\
[x_1,\ldots,x_m]_{\boldsymbol{D}}.(\lambda y.s)t_1\ldots t_n & = & \mathrm{D}_{\langle \nu_0,\ldots,\nu_n\rangle_m} s' t'_1\ldots t'_n \\
 & & \text{where } s' = ([\{x_1\}^{\nu_0(1)},\ldots,\{x_m\}^{\nu_0(m)},y]_{\boldsymbol{D}}.s) \\
 & & t'_i = [\{x_1\}^{\nu_i(1)},\ldots,\{x_m\}^{\nu_i(m)}]_{\boldsymbol{D}}.t_i \text{ for } i=1,\ldots,n \\
 & & \nu_0(j)=1 \text{ iff } x_j \in \mathrm{FV}(\lambda y.s), \text{ for } j=1,\ldots,m \\
 & & \nu_i(j)=1 \text{ iff } x_j \in \mathrm{FV}(t_i), \text{ for } i=1,\ldots,n, j=1,\ldots,m
\end{array}
$$

Implementation

Preprocessing

- Every occurence of $\lambda x.tx$ where $x \notin \mathrm{FV}(t)$ is replaced by t.

Implementation

Preprocessing

- Every occurence of $\lambda x.tx$ where $x \notin \mathrm{FV}(t)$ is replaced by t.
- Any equation between a constant and a lambda-expression is translated directly to another equation. For example, the formula $h = \lambda x.fx(gx)$ is translated to $\forall x.hx = fx(gx)$.

Implementation

Preprocessing

- Every occurence of $\lambda x.tx$ where $x \notin \mathrm{FV}(t)$ is replaced by t.
- Any equation between a constant and a lambda-expression is translated directly to another equation. For example, the formula $h = \lambda x.fx(gx)$ is translated to $\forall x.hx = fx(gx)$.
- All terms are reduced to β-normal form.

Experimental results

Algorithm	Solved%	Solved
Abdali + director strings D	28.738	1903
Abdali A	28.511	1888
Turner T	27.167	1799
Schönfinkel S	25.899	1715
Lambda-lifting $L2$	23.694	1569
Lambda-lifting $L1$	10.556	699
any	35.246	2334

Total 6622 problems. Only problems with non-trivial lambda-abstractions (i.e. those that cannot be eliminated by the preprocessing) in the goal or one of the dependencies.

Experimental results

Algorithm	Solved%	Solved
Abdali + director strings D	28.738	1903
Abdali A	28.511	1888
Turner T	27.167	1799
Schönfinkel S	25.899	1715
Lambda-lifting $L2$	23.694	1569
Lambda-lifting $L1$	10.556	699
any	35.246	2334

Total 6622 problems. Only problems with non-trivial lambda-abstractions (i.e. those that cannot be eliminated by the preprocessing) in the goal or one of the dependencies.

Cumulative results for provers: CVC version 4, E version 1.8, Vampire version 3.0, Z3 version 4.0 and SPASS version 3.5.