定理証明支援系Coq上での対話的スタック指向プログラミング

坂口和彦

筑波大学 情報学群情報科学類

2014/6/19 IPSJ-PRO 2014-1

スタック指向プログラミングとは

- スタックの操作を基本としてプログラムを記述する方法
- スタック指向プログラミング言語: PostScript, Forth, etc.

スタック指向プログラミングの例

PostScript で整数 n、プログラム p が積まれた状態で以下のプログラムを実行すると、p が n 回実行される

```
{
  2 index 0 eq {
    pop pop pop
  }{
    [ 2 index /exec cvx 6 5 roll
      1 sub 6 4 roll dup /exec cvx ] cvx exec
  } ifelse
} dup exec
```

スタック指向プログラミングの例

PostScript で整数 n、プログラム p が積まれた状態で以下のプログラムを実行すると、p が n 回実行される

```
{
  2 index 0 eq {
    pop pop pop
  }{
    [ 2 index /exec cvx 6 5 roll
      1 sub 6 4 roll dup /exec cvx ] cvx exec
  } ifelse
} dup exec
```

本研究の目的

- スタック指向プログラミングは難しい
  - スタック上の値の並び、数、インデックス等を間違える
  - 値の並びをメモしながらプログラムを書き進めるのも手間が大きい
- これを簡単にするプログラミング環境が欲しい

研究の概要

- 定理証明支援系 Coq を使って、対話的スタック指向プログラミング環境を開発した
  - プログラムのそれぞれの時点でのスタックの状態が一般的にどうなっているかを表示できる
  - プログラムが書けると同時に (仕様を満たすことの) 証明も付いてくる
  - プログラムの仕様やスタックの状態の表現には、Coq が持っている論理や Coq 上で定義した関数が使える
- https://github.com/pi8027/formalized-postscript

対象とする言語: PS0

PS0 の命令、操作的意味論の定義

$$
\begin{array}{lll}
i_1, i_2 ::= & \mathsf{pop} & (v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i}) \\
\mid & \mathsf{copy} & (v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{swap} & (v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{cons} & (v_1 v_2 \overline{v}, \mathsf{cons}\ \overline{i}) \mapsto (\mathsf{pair}(v_2, v_1)\, \overline{v}, \overline{i}) \\
\mid & \mathsf{quote} & (v_1 \overline{v}, \mathsf{quote}\ \overline{i}) \mapsto (\mathsf{push}(v_1)\, \overline{v}, \overline{i}) \\
\mid & \mathsf{exec} & (v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i}) \\
\mid & \mathsf{push}(i_1) & (\overline{v}, \mathsf{push}(v_1)\, \overline{i}) \mapsto (v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{pair}(i_1, i_2) & (\overline{v}, \mathsf{pair}(i_1, i_2)\ \overline{i}) \mapsto (\overline{v}, i_1 i_2 \overline{i})
\end{array}
$$

対象とする言語: PS0

PS0 の命令、操作的意味論の定義

$$
\begin{array}{lll}
i_1, i_2 ::= & \mathsf{pop} & (v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i}) \\
\mid & \mathsf{copy} & (v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{swap} & (v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{cons} & (v_1 v_2 \overline{v}, \mathsf{cons}\ \overline{i}) \mapsto (\mathsf{pair}(v_2, v_1)\, \overline{v}, \overline{i}) \\
\mid & \mathsf{quote} & (v_1 \overline{v}, \mathsf{quote}\ \overline{i}) \mapsto (\mathsf{push}(v_1)\, \overline{v}, \overline{i}) \\
\mid & \mathsf{exec} & (v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i}) \\
\mid & \mathsf{push}(i_1) & (\overline{v}, \mathsf{push}(v_1)\, \overline{i}) \mapsto (v_1 \overline{v}, \overline{i}) \\
\mid & \mathsf{pair}(i_1, i_2) & (\overline{v}, \mathsf{pair}(i_1, i_2)\ \overline{i}) \mapsto (\overline{v}, i_1 i_2 \overline{i})
\end{array}
$$

対象とする言語: PS0

$$(v_1 \ldots v_n, i_1 \ldots i_n)$$

- 値もプログラムも命令を使って表現する
- プログラムの実行中の状態は、2 つの命令列の組で表す
  - スタック
  - 残りのプログラム

対象とする言語: PS0

$$(v_1 \ldots v_n, i_1 \ldots i_n)$$

- 値もプログラムも命令を使って表現する
- プログラムの実行中の状態は、2 つの命令列の組で表す
  - スタック
  - 残りのプログラム

対象とする言語: PS0

$$(v_1 \ldots v_n, i_1 \ldots i_n)$$

- 値もプログラムも命令を使って表現する
- プログラムの実行中の状態は、2 つの命令列の組で表す
  - スタック
  - 残りのプログラム

pop, copy, swap, exec 命令

- $(v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i})$
  pop 命令は、スタックの先頭の値を捨てる
- $(v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i})$
  copy 命令は、スタックの先頭の値を取り出し、その値をスタックの先頭に 2 回積む
- $(v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i})$
  swap 命令は、スタックの先頭の 2 つの値の順番を入れ替える
- $(v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i})$
  exec 命令は、スタックの先頭の値を取り出し、その値をプログラムの先頭に追加する

pop, copy, swap, exec 命令

- $(v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i})$
  pop 命令は、スタックの先頭の値を捨てる
- $(v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i})$
  copy 命令は、スタックの先頭の値を取り出し、その値をスタックの先頭に 2 回積む
- $(v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i})$
  swap 命令は、スタックの先頭の 2 つの値の順番を入れ替える
- $(v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i})$
  exec 命令は、スタックの先頭の値を取り出し、その値をプログラムの先頭に追加する

pop, copy, swap, exec 命令

- $(v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i})$
  pop 命令は、スタックの先頭の値を捨てる
- $(v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i})$
  copy 命令は、スタックの先頭の値を取り出し、その値をスタックの先頭に 2 回積む
- $(v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i})$
  swap 命令は、スタックの先頭の 2 つの値の順番を入れ替える
- $(v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i})$
  exec 命令は、スタックの先頭の値を取り出し、その値をプログラムの先頭に追加する

pop, copy, swap, exec 命令

- $(v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i})$
  pop 命令は、スタックの先頭の値を捨てる
- $(v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i})$
  copy 命令は、スタックの先頭の値を取り出し、その値をスタックの先頭に 2 回積む
- $(v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i})$
  swap 命令は、スタックの先頭の 2 つの値の順番を入れ替える
- $(v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i})$
  exec 命令は、スタックの先頭の値を取り出し、その値をプログラムの先頭に追加する

pop, copy, swap, exec 命令

- $(v_1 \overline{v}, \mathsf{pop}\ \overline{i}) \mapsto (\overline{v}, \overline{i})$
  pop 命令は、スタックの先頭の値を捨てる
- $(v_1 \overline{v}, \mathsf{copy}\ \overline{i}) \mapsto (v_1 v_1 \overline{v}, \overline{i})$
  copy 命令は、スタックの先頭の値を取り出し、その値をスタックの先頭に 2 回積む
- $(v_1 v_2 \overline{v}, \mathsf{swap}\ \overline{i}) \mapsto (v_2 v_1 \overline{v}, \overline{i})$
  swap 命令は、スタックの先頭の 2 つの値の順番を入れ替える
- $(v_1 \overline{v}, \mathsf{exec}\ \overline{i}) \mapsto (\overline{v}, v_1 \overline{i})$
  exec 命令は、スタックの先頭の値を取り出し、その値をプログラムの先頭に追加する

$\mathsf{pair}(i_1, i_2)$, cons 命令

- $(\overline{v}, \mathsf{pair}(i_1, i_2)\ \overline{i}) \Mapsto (\overline{v}, i_1 i_2 \overline{i})$
  $\mathsf{pair}(i_1, i_2)$ 命令は、プログラムの先頭に i_1, i_2 を追加する
- $(v_1 v_2 \overline{v}, \mathsf{cons}\ \overline{i}) \Mapsto (\mathsf{pair}(v_2, v_1)\, \overline{v}, \overline{i})$
  cons 命令は、スタックの先頭の 2 つの値を取り出し、pair を使って組にした値をスタックの先頭に積む

$\mathsf{pair}(i_1, i_2)$, cons 命令

- $(\overline{v}, \mathsf{pair}(i_1, i_2)\ \overline{i}) \Mapsto (\overline{v}, i_1 i_2 \overline{i})$
  $\mathsf{pair}(i_1, i_2)$ 命令は、プログラムの先頭に i_1, i_2 を追加する
- $(v_1 v_2 \overline{v}, \mathsf{cons}\ \overline{i}) \Mapsto (\mathsf{pair}(v_2, v_1)\ \overline{v}, \overline{i})$
  cons 命令は、スタックの先頭の 2 つの値を取り出し、pair を使って組にした値をスタックの先頭に積む

$\mathsf{pair}(i_1, i_2)$, cons 命令

- $(\overline{v}, \mathsf{pair}(i_1, i_2)\ \overline{i}) \Mapsto (\overline{v}, i_1 i_2 \overline{i})$
  $\mathsf{pair}(i_1, i_2)$ 命令は、プログラムの先頭に i_1, i_2 を追加する
- $(v_1 v_2 \overline{v}, \mathsf{cons}\ \overline{i}) \Mapsto (\mathsf{pair}(v_2, v_1)\ \overline{v}, \overline{i})$
  cons 命令は、スタックの先頭の 2 つの値を取り出し、pair を使って組にした値をスタックの先頭に積む

$\mathsf{push}(i)$, quote 命令

- $(\overline{v}, \mathsf{push}(v_1)\,\overline{i}) \Mapsto (v_1\overline{v}, \overline{i})$
  $\mathsf{push}(v_1)$ 命令は、スタックの先頭に v_1 を積む
- $(v_1\overline{v}, \mathsf{quote}\ \overline{i}) \Mapsto (\mathsf{push}(v_1)\,\overline{v}, \overline{i})$
  quote 命令は、スタックの先頭の値 v_1 を取り出し、
  $\mathsf{push}(v_1)$ をスタックの先頭に積む

push(i), quote 命令

- $(\overline{v}, \mathsf{push}(v_1)\,\overline{i}) \Mapsto (v_1\overline{v}, \overline{i})$
  $\mathsf{push}(v_1)$ 命令は、スタックの先頭に v_1 を積む
- $(v_1\overline{v}, \mathsf{quote}\ \overline{i}) \Mapsto (\mathsf{push}(v_1)\,\overline{v}, \overline{i})$
  quote 命令は、スタックの先頭の値 v_1 を取り出し、
  $\mathsf{push}(v_1)$ をスタックの先頭に積む

push(i), quote 命令

- $(\overline{v}, \mathsf{push}(v_1)\,\overline{i}) \Mapsto (v_1\overline{v}, \overline{i})$
  push(v_1) 命令は、スタックの先頭に v_1 を積む
- $(v_1\overline{v}, \mathsf{quote}\ \overline{i}) \Mapsto (\mathsf{push}(v_1)\,\overline{v}, \overline{i})$
  quote 命令は、スタックの先頭の値 v_1 を取り出し、
  push(v_1) をスタックの先頭に積む

PS0プログラミング

- PS0の命令は、どれも単体ではスタックの3番目以降の値に対する操作が書けない
- スタックの先頭の2要素を保持しつつ、3番目の値を変更するにはどうするか
- スタックの先頭の3要素の順序を逆順にするにはどうするか

値の組を作る

$\mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2))$ を実行すると、i_1 と i_2 がスタックに積まれる

$$(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots)$$

値の組を作る

$\mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2))$ を実行すると、i_1 と i_2 がスタックに積まれる

$$
\begin{aligned}
&(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots) \\
\mapsto &(\ldots, \mathsf{push}(i_1)\, \mathsf{push}(i_2) \ldots)
\end{aligned}
$$

値の組を作る

$\mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2))$ を実行すると、i_1 と i_2 がスタックに積まれる

$$
\begin{aligned}
&(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots) \\
\mapsto &(\ldots, \mathsf{push}(i_1)\, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_1 \ldots, \mathsf{push}(i_2) \ldots)
\end{aligned}
$$

値の組を作る

pair(push(i_1), push(i_2)) を実行すると、i_1 と i_2 がスタックに積まれる

$$
\begin{aligned}
&(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots) \\
\mapsto &(\ldots, \mathsf{push}(i_1)\, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_1 \ldots, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_2\ i_1 \ldots, \ldots)
\end{aligned}
$$

値の組を作る

$\mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2))$ を実行すると、i_1 と i_2 がスタックに積まれる

$$
\begin{aligned}
&(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots) \\
\mapsto &(\ldots, \mathsf{push}(i_1)\, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_1 \ldots, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_2\, i_1 \ldots, \ldots)
\end{aligned}
$$

- この形の命令を作ることで、2 つの命令を元の命令に復元可能な組にできる
- 元とは逆の順番で並んでいることに注意

値の組を作る

pair(push(i_1), push(i_2)) を実行すると、i_1 と i_2 がスタックに積まれる

$$
\begin{aligned}
&(\ldots, \mathsf{pair}(\mathsf{push}(i_1), \mathsf{push}(i_2)) \ldots) \\
\mapsto &(\ldots, \mathsf{push}(i_1)\, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_1 \ldots, \mathsf{push}(i_2) \ldots) \\
\mapsto &(i_2\ i_1 \ldots, \ldots)
\end{aligned}
$$

- この形の命令を作ることで、2 つの命令を元の命令に復元可能な組にできる
- 元とは逆の順番で並んでいることに注意

例: スタック上の3つの値を逆順にする

$(a\ b\ c\ \overline{v},\ \ldots)$

例: スタック上の3つの値を逆順にする

$$(a\ b\ c\ \overline{v}, \mathrm{quote}\ \ldots)$$
$$\mapsto (\mathrm{push}(a)\ b\ c\ \overline{v},\ \ldots)$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ \ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ \ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v},\ \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ \ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ \ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ \ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \ \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
& (a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto & (\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto & (b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto & (\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto & (\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto & (c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ \overline{v}, \ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ \overline{v}, \mathsf{exec}\ldots) \\
\mapsto &(\overline{v}, \mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ \overline{v}, \mathsf{exec}\ldots) \\
\mapsto &(\overline{v}, \mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ldots) \\
\mapsto &(\overline{v}, \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \mathsf{push}(c)\ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
&(a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto &(c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto &(\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto &(\mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ \overline{v}, \mathsf{exec}\ldots) \\
\mapsto &(\overline{v}, \mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ldots) \\
\mapsto &(\overline{v}, \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \mathsf{push}(c)\ldots) \\
\mapsto^* &(b\ a\ \overline{v}, \mathsf{push}(c)\ldots)
\end{aligned}
$$

例: スタック上の3つの値を逆順にする

$$
\begin{aligned}
& (a\ b\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto & (\mathsf{push}(a)\ b\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto & (b\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto & (\mathsf{push}(b)\ \mathsf{push}(a)\ c\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto & (\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ c\ \overline{v}, \mathsf{swap}\ldots) \\
\mapsto & (c\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{quote}\ldots) \\
\mapsto & (\mathsf{push}(c)\ \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \overline{v}, \mathsf{cons}\ldots) \\
\mapsto & (\mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ \overline{v}, \mathsf{exec}\ldots) \\
\mapsto & (\overline{v}, \mathsf{pair}(\mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b)), \mathsf{push}(c))\ldots) \\
\mapsto & (\overline{v}, \mathsf{pair}(\mathsf{push}(a), \mathsf{push}(b))\ \mathsf{push}(c)\ldots) \\
\mapsto^{*} & (b\ a\ \overline{v}, \mathsf{push}(c)\ldots) \mapsto (c\ b\ a\ \overline{v}, \ldots)
\end{aligned}
$$

CoqでのPS0プログラムの検証

直前の例に証明を付けてみる

```
Definition rev3 : seq inst :=
  [:: instquote; instswap; instquote; instcons;
      instswap; instquote; instcons; instexec].

Goal forall i1 i2 i3 vs cs,
  (i3 :: i2 :: i1 :: vs, rev3) |=>*
  (i1 :: i2 :: i3 :: vs, [::]]).
```

証明の自動化: evalauto タクティク

- $s_1 \mapsto^* s_2$ を証明するとき、s_1 のプログラムの先頭を見ると最初に適用すべきルールが分かる
- このルールの適用を繰り返すことで、証明の部分的な自動化ができる
- 証明可能な状態から evalauto によって証明不能な状態に陥ることは無い

対話的プログラミング

1. スタックの形を見る
2. 次に何を実行すれば良いかを考える

の繰り返しでプログラムを記述したい

- 最初に書くのは仕様だけで、プログラムは書かない
- 証明を進めるごとにそれに対応するプログラムを埋めていく
- 証明からプログラムが取り出せる

対話的プログラミング

1. スタックの形を見る
2. 次に何を実行すれば良いかを考える

の繰り返しでプログラムを記述したい

- 最初に書くのは仕様だけで、プログラムは書かない
- 証明を進めるごとにそれに対応するプログラムを埋めていく
- 証明からプログラムが取り出せる

対話的プログラミングの例

$$(a\,b\,c\,\overline{v}, ?p_1) \mapsto\!\!\Rightarrow^* (c\,b\,a\,\overline{v}, \epsilon)$$

を証明する (? は Coq の存在変数、ϵ は空列)

$\mapsto\!\!\Rightarrow^*$ を分割する

$$(a\,b\,c\,\overline{v}, ?p_1) \mapsto\!\!\Rightarrow^* ?s_1 \wedge ?s_1 \mapsto\!\!\Rightarrow^* (c\,b\,a\,\overline{v}, \epsilon)$$

$\wedge$ の左側に quote の実行規則を適用

$$(\mathrm{push}(a)\,b\,c\,\overline{v}, ?p_2) \mapsto\!\!\Rightarrow^* (c\,b\,a\,\overline{v}, \epsilon)$$

以降も同様の方法で対話的にプログラムを記述できる

対話的プログラミングの例

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

を証明する (? は Coq の存在変数、ϵ は空列)
$\Mapsto^*$ を分割する

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* ?s_1 \wedge ?s_1 \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

$\wedge$ の左側に quote の実行規則を適用

$$(\mathrm{push}(a)\,b\,c\,\overline{v}, ?p_2) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

以降も同様の方法で対話的にプログラムを記述できる

対話的プログラミングの例

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

を証明する (? は Coq の存在変数、ϵ は空列)
$\Mapsto^*$ を分割する

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* ?s_1 \wedge ?s_1 \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

$\wedge$ の左側に quote の実行規則を適用

$$(\mathrm{push}(a)\,b\,c\,\overline{v}, ?p_2) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

以降も同様の方法で対話的にプログラムを記述できる

対話的プログラミングの例

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

を証明する (? は Coq の存在変数、ϵ は空列)
$\Mapsto^*$ を分割する

$$(a\,b\,c\,\overline{v}, ?p_1) \Mapsto^* ?s_1 \wedge ?s_1 \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

$\wedge$ の左側に quote の実行規則を適用

$$(\mathrm{push}(a)\,b\,c\,\overline{v}, ?p_2) \Mapsto^* (c\,b\,a\,\overline{v}, \epsilon)$$

以降も同様の方法で対話的にプログラムを記述できる

データ型の定義

- 実用的なプログラムを記述するためには、自然数やブール値を扱えることが必須
- 自然数型を定義するための準備として、以下の定義を導入する

$$\mathsf{nop} = \mathsf{pair}(\mathsf{push}(\mathsf{pop}), \mathsf{pop})$$
$$i^n = \underbrace{\mathsf{pair}(\mathsf{pair}(\ldots \mathsf{pair}}_{n}(\mathsf{nop}, i) \ldots, i), i)$$

直感的には、i^n は i を n 回実行する命令になっている

データ型の定義

「命令 i_1 が自然数 n を表現する」を「i_1 を実行することでスタックの先頭の i_2 が ${i_2}^n$ に置き換わる」で定義する:

$$\forall i_2\, \overline{v}\, \overline{i}.(i_2\, \overline{v}, i_1\, \overline{i}) \mapsto^* ({i_2}^n\, \overline{v}, \overline{c})$$

自然数 n を表す命令を実行した後に exec を実行すると、最初にスタックの先頭にあった命令を n 回実行できる

データ型の定義

「命令 i_1 が自然数 n を表現する」を「i_1 を実行することでスタックの先頭の i_2 が ${i_2}^n$ に置き換わる」で定義する:

$$\forall i_2\, \overline{v}\, \overline{i}.(i_2\, \overline{v}, i_1\, \overline{i}) \mapsto^* ({i_2}^n\, \overline{v}, \overline{c})$$

自然数 n を表す命令を実行した後に exec を実行すると、最初にスタックの先頭にあった命令を n 回実行できる

繰り返しを含むプログラムの記述

- 自然数を定義したので、加減乗除や GCD を書きたい
- Coq の帰納法を使うことで、問題無く記述できる

PostScript への変換

- PS0 プログラムを PostScript プログラムに変換し、実行できる
- GNU M4 を使って PostScript プログラムに PS0 プログラム (Coq の項) を埋め込めるようにした

まとめ

- Coq 上でスタック指向言語のプログラム (と証明) を対話的に記述できるプログラミング環境を開発した
  - スタック指向言語に限らず、より広い範囲に対しても同様の手法が適用できると考えている
- 自然数やブール値などの定義と、それに関するいくつかの計算を実装した
  - 加減乗除、GCD など
- その上で記述したプログラムを PostScript に変換して実行できることを確認した