Verified Transformation of Continuation-Passing Style into Static Single Assignment Form

Siyu Liu   Yuting Wang

John Hopcroft Center for Computer Science,
School of Electronic Information and Electrical Engineering,
Shanghai Jiao Tong University

TASE 2023,
Bristol, UK

Table of Contents

1. Background & Motivation
2. Implementation of CPS $\Rightarrow$ SSA Transformation
3. Verification of CPS $\Rightarrow$ SSA Transformation
4. Evaluation & Conclusion

Table of Contents

1. Background & Motivation
2. Implementation of CPS $\Rightarrow$ SSA Transformation
3. Verification of CPS $\Rightarrow$ SSA Transformation
4. Evaluation & Conclusion

Intermediate Representations for Compilers

CPS (Continuation-Passing Style)

- Compilers for functional programming languages.
- Every control point is explicitly named via continuations.
- Facilitate control-flow analysis.

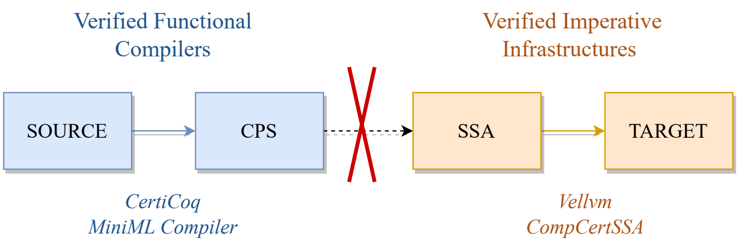
SSA (Static Single Assignment)

- Mainstream compiler infrastructures like LLVM and GCC.
- Every variable could be assigned only once.
- Facilitate data-flow analysis.

SSA programs can be represented by functional languages.

Motivation

There is no verified transformation from CPS to SSA.



Can verified compilers for functional programming languages exploit the benefits of SSA?

Challenge & Approach

Most common techniques of compiler verification:

- Verification via logical relations
  Not compatible with the existing SSA infrastructures
- Verification via simulation

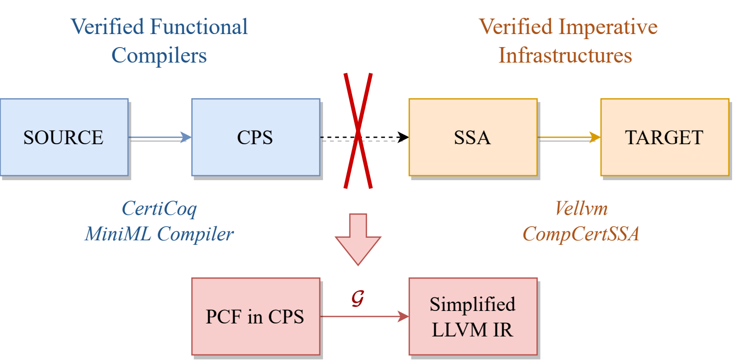
So we use verification via simulation.

Challenge: different components of program states:

- CPS: term, continuation information...
- SSA: program counter, variable information, stack...

Main Contributions 1

Design and verification of a transformation from PCF in CPS to Simplified LLVM IR.



Main Contributions 2

Apply it to a compiler from PCF to LLVM IR.

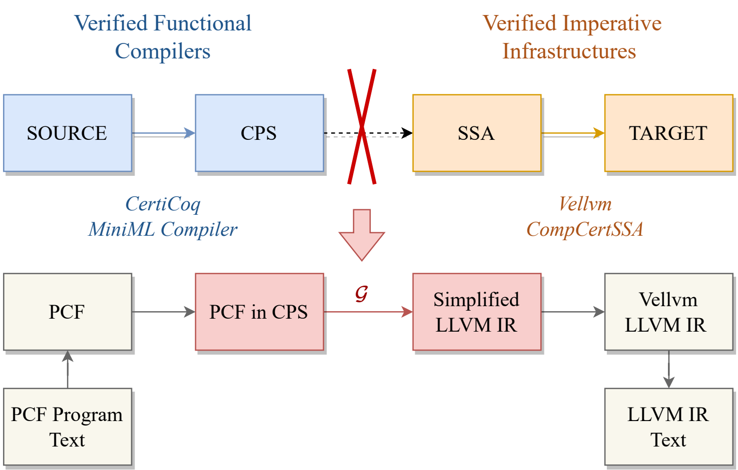


Table of Contents

1. Background & Motivation
2. Implementation of CPS $\Rightarrow$ SSA Transformation
3. Verification of CPS $\Rightarrow$ SSA Transformation
4. Evaluation & Conclusion

Source Language: PCF in CPS

$$\text{PCF in direct style} \xrightarrow{\textit{CPS Transformation}} \text{PCF in CPS}$$

Syntax of PCF

$op := + \mid - \mid \times \mid \div$

$t := i \mid x \mid t_1\ t_2 \mid \textbf{ifz}\ t_1\ t_2\ t_3$
$\mid op\ t_1\ t_2 \mid \textbf{let}\ x = t_1\ \textbf{in}\ t_2$
$\mid \textbf{fix}\ f\ x\ t$

Example of PCF

$(\textbf{fix}\ f\ x\ =$
$\textbf{ifz}\ x\ 0\ 1)$
2

Syntax of CPS

$v := i \mid x$

$t := \textbf{letval}\ x = v\ \textbf{in}\ t$
$\mid k\ v \mid f\ k\ v \mid \textbf{ifz}\ v\ t_1\ t_2$
$\mid \textbf{letop}\ x = op\ x_1\ x_2\ \textbf{in}\ t$
$\mid \textbf{letcont}\ k\ x = t_1\ \textbf{in}\ t_2$
$\mid \textbf{letfun}\ f\ k\ x = t_1\ \textbf{in}\ t_2$

Example of CPS

$\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x$
$(\textbf{letval}\ x_1 = 0\ \textbf{in}$
$k\ x_1)$
$(\textbf{letval}\ x_2 = 1\ \textbf{in}$
$k\ x_2))\ \textbf{in}$
$(\textbf{letval}\ x_3 = 2\ \textbf{in}$
$(\textbf{letcont}\ k_2\ y = k_{init}\ y\ \textbf{in}$
$(f\ k_2\ x_3)))$

Source Language: PCF in CPS

$$\text{PCF in direct style} \xrightarrow{\textit{CPS Transformation}} \text{PCF in CPS}$$

Syntax of PCF

$op \; := \; + \mid - \mid \times \mid \div$

$t \; := \; i \mid x \mid t_1 \; t_2 \mid \textbf{ifz} \; t_1 \; t_2 \; t_3$
$\quad \mid op \; t_1 \; t_2 \mid \textbf{let} \; x = t_1 \; \textbf{in} \; t_2$
$\quad \mid \textbf{fix} \; f \; x \; t$

Syntax of CPS

$v \; := \; i \mid x$

$t \; := \; \textbf{letval} \; x = v \; \textbf{in} \; t$
$\quad \mid k \; v \mid f \; k \; v \mid \textbf{ifz} \; v \; t_1 \; t_2$
$\quad \mid \textbf{letop} \; x = op \; x_1 \; x_2 \; \textbf{in} \; t$
$\quad \mid \textbf{letcont} \; k \; x = t_1 \; \textbf{in} \; t_2$
$\quad \mid \textbf{letfun} \; f \; k \; x = t_1 \; \textbf{in} \; t_2$

Example of PCF

$(\textbf{fix} \; f \; x \; =$
$\quad \textbf{ifz} \; x \; 0 \; 1)$
2

Example of CPS

$\textbf{letfun} \; f \; k \; x = (\textbf{ifz} \; x$
$\quad (\textbf{letval} \; x_1 = 0 \; \textbf{in}$
$\qquad k \; x_1)$
$\quad (\textbf{letval} \; x_2 = 1 \; \textbf{in}$
$\qquad k \; x_2)) \; \textbf{in}$
$(\textbf{letval} \; x_3 = 2 \; \textbf{in}$
$\quad (\textbf{letcont} \; k_2 \; y = k_{init} \; y \; \textbf{in}$
$\qquad (f \; k_2 \; x_3)))$

Target SSA Language

Syntax

$$t := \overline{f}$$
$$f := \textbf{define}\ l_1(l_2)\ \overline{b}$$
$$b := l : \overline{\phi}\ \overline{a}\ r$$
$$a := x = c;$$
$$c := v \mid op\ v_1\ v_2 \mid \textbf{icmp}\ v_1\ v_2 \mid \textbf{call}\ x\ v$$
$$phi := x = \phi\ \overline{(l,\ v)};$$
$$r := \textbf{ret}\ v \mid \textbf{br}_{\textbf{uc}}\ l \mid \textbf{br}_{\textbf{c}}\ v\ l_1\ l_2$$
$$l := string \quad v := i \mid x$$

Example of SSA

$\textbf{define}\ f\ (x)$

$b_1 :\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_{\textbf{c}}\ b_0\ t_0\ f_0;$

$t_0 :\ x_1 = 0;\ \textbf{br}_{\textbf{uc}}\ if_0;$

$f_0 :\ x_2 = 1;\ \textbf{br}_{\textbf{uc}}\ if_0;$

$if_0 :\ r_x = \phi\ [(t_0,\ x_1),\ (f_0,\ x_2)];\ \textbf{ret}\ r_x;$

$\textbf{define}\ main\ ()$

$b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2;$

$k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};$

CPS $\Rightarrow$ SSA Transformation

Key ideas:

CPS		SSA
Variable binding	$\Rightarrow$	Assignment to a fresh variable
Continuation	$\Rightarrow$	A new basic block
Multiple application of the same continuation	$\Rightarrow$	A Φ-node
...		

CPS $\Rightarrow$ SSA Transformation

$\mathcal{G}$: a recursive function

1. START: $\mathcal{G}$ takes the CPS term and an empty SSA program with main function.
2. $\mathcal{G}$ recursively translates the CPS term:
   - Puts new components (basic blocks, instructions...) into the SSA program.
   - Updates the parameters.
3. Translation is finished: Return the current SSA program.

Example of CPS $\Rightarrow$ SSA Transformation

CPS Program

$$
\begin{array}{l}
\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x \\
\quad (\textbf{letval}\ x_1 = 0\ \textbf{in} \\
\quad\quad k\ x_1) \\
\quad (\textbf{letval}\ x_2 = 1\ \textbf{in} \\
\quad\quad k\ x_2)) \\
(\textbf{letval}\ x_3 = 2\ \textbf{in} \\
\quad (\textbf{letcont}\ k_2\ y = \\
\quad\quad\quad k_{init}\ y\ \textbf{in} \\
\quad\quad (f\ k_2\ x_3)))
\end{array}
$$

SSA Program

$$
\begin{array}{l}
\textbf{define}\ f\ (x) \\
\quad b_1:\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_{\textbf{c}}\ b_0\ t_0\ f_0; \\
\quad t_0:\ x_1 = 0;\ \textbf{br}_{\textbf{uc}}\ if_0; \\
\quad f_0:\ x_2 = 1;\ \textbf{br}_{\textbf{uc}}\ if_0; \\
\quad if_0:\ r_x = \phi\ [(t_0, x_1), (f_0, x_2)];\ \textbf{ret}\ r_x; \\
\textbf{define}\ main\ () \\
\quad b_1:\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2; \\
\quad k_2:\ r_{k2} = y;\ \textbf{ret}\ r_{k2};
\end{array}
$$

Example of CPS ⇒ SSA Transformation

CPS Program

letfun $f\ k\ x =$ (**ifz** x
  (**letval** $x_1 = 0$ **in**
    $k\ x_1$)
  (**letval** $x_2 = 1$ **in**
    $k\ x_2$))
(**letval** $x_3 = 2$ **in**
  (**letcont** $k_2\ y =$
      $k_{init}\ y$ **in**
    $(f\ k_2\ x_3)))$

SSA Program

define $f\ (x)$
  b_1 : $b_0 =$ **icmp** $x\ 0$; $\mathbf{br_c}\ b_0\ t_0\ f_0$;
  t_0 : $x_1 = 0$; $\mathbf{br_{uc}}\ if_0$;
  f_0 : $x_2 = 1$; $\mathbf{br_{uc}}\ if_0$;
  if_0 : $r_x = \phi\ [(t_0, x_1), (f_0, x_2)]$; **ret** r_x;

define $main\ ()$
  b_1 : $x_3 = 2$; $y =$ **call** $f\ x_3$; $\mathbf{br_{uc}}\ k_2$;
  k_2 : $r_{k2} = y$; **ret** r_{k2};

Example of CPS $\Rightarrow$ SSA Transformation

CPS Program

$\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x$
$\quad (\textbf{letval}\ x_1 = 0\ \textbf{in}$
$\qquad k\ x_1)$
$\quad (\textbf{letval}\ x_2 = 1\ \textbf{in}$
$\qquad k\ x_2))$
$(\textbf{letval}\ x_3 = 2\ \textbf{in}$
$\quad (\textbf{letcont}\ k_2\ y =$
$\qquad k_{init}\ y\ \textbf{in}$
$\qquad (f\ k_2\ x_3)))$

SSA Program

$\textbf{define}\ f\ (x)$
$\quad b_1 :\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_\textbf{c}\ b_0\ t_0\ f_0;$
$\quad t_0 :\ x_1 = 0;\ \textbf{br}_\textbf{uc}\ if_0;$
$\quad f_0 :\ x_2 = 1;\ \textbf{br}_\textbf{uc}\ if_0;$
$\quad if_0 :\ r_x = \phi\ [(t_0, x_1), (f_0, x_2)];\ \textbf{ret}\ r_x;$

$\textbf{define}\ main\ ()$
$\quad b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_\textbf{uc}\ k_2;$
$\quad k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};$

Example of CPS $\Rightarrow$ SSA Transformation

CPS Program

$$
\begin{array}{l}
\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x \\
\quad (\textbf{letval}\ x_1 = 0\ \textbf{in} \\
\quad\quad k\ x_1) \\
\quad (\textbf{letval}\ x_2 = 1\ \textbf{in} \\
\quad\quad k\ x_2)) \\
(\textbf{letval}\ x_3 = 2\ \textbf{in} \\
\quad (\textbf{letcont}\ k_2\ y = \\
\quad\quad\quad k_{init}\ y\ \textbf{in} \\
\quad\quad (f\ k_2\ x_3)))
\end{array}
$$

SSA Program

$$
\begin{array}{l}
\textbf{define}\ f\ (x) \\
\quad b_1 :\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_{\textbf{c}}\ b_0\ t_0\ f_0; \\
\quad t_0 :\ x_1 = 0;\ \textbf{br}_{\textbf{uc}}\ if_0; \\
\quad f_0 :\ x_2 = 1;\ \textbf{br}_{\textbf{uc}}\ if_0; \\
\quad if_0 :\ r_x = \phi\ [(t_0, x_1), (f_0, x_2)];\ \textbf{ret}\ r_x; \\
\textbf{define}\ main\ () \\
\quad b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2; \\
\quad k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};
\end{array}
$$

Example of CPS $\Rightarrow$ SSA Transformation

CPS Program

$$
\begin{array}{l}
\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x \\
\quad (\textbf{letval}\ x_1 = 0\ \textbf{in} \\
\qquad k\ x_1) \\
\quad (\textbf{letval}\ x_2 = 1\ \textbf{in} \\
\qquad k\ x_2)) \\
(\textbf{letval}\ x_3 = 2\ \textbf{in} \\
\quad (\textbf{letcont}\ k_2\ y = \\
\qquad\quad k_{init}\ y\ \textbf{in} \\
\qquad (f\ k_2\ x_3)))
\end{array}
$$

SSA Program

$$
\begin{array}{l}
\textbf{define}\ f\ (x) \\
\quad b_1 :\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_\textbf{c}\ b_0\ t_0\ f_0; \\
\quad t_0 :\ x_1 = 0;\ \textbf{br}_\textbf{uc}\ if_0; \\
\quad f_0 :\ x_2 = 1;\ \textbf{br}_\textbf{uc}\ if_0; \\
\quad if_0 :\ r_x = \phi\ [(t_0, x_1), (f_0, x_2)];\ \textbf{ret}\ r_x; \\
\textbf{define}\ main\ () \\
\quad b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_\textbf{uc}\ k_2; \\
\quad k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};
\end{array}
$$

Example of CPS $\Rightarrow$ SSA Transformation

CPS Program

$$
\begin{array}{l}
\textbf{letfun}\ f\ k\ x = (\textbf{ifz}\ x \\
\quad (\textbf{letval}\ x_1 = 0\ \textbf{in} \\
\qquad k\ x_1) \\
\quad (\textbf{letval}\ x_2 = 1\ \textbf{in} \\
\qquad k\ x_2)) \\
(\textbf{letval}\ x_3 = 2\ \textbf{in} \\
\quad (\textbf{letcont}\ k_2\ y = \\
\qquad\quad k_{init}\ y\ \textbf{in} \\
\qquad (f\ k_2\ x_3)))
\end{array}
$$

SSA Program

$$
\begin{array}{l}
\textbf{define}\ f\ (x) \\
\quad b_1 :\ b_0 = \textbf{icmp}\ x\ 0;\ \textbf{br}_\textbf{c}\ b_0\ t_0\ f_0; \\
\quad t_0 :\ x_1 = 0;\ \textbf{br}_\textbf{uc}\ if_0; \\
\quad f_0 :\ x_2 = 1;\ \textbf{br}_\textbf{uc}\ if_0; \\
\quad if_0 :\ r_x = [(t_0, x_1), (f_0, x_2)];\ \textbf{ret}\ r_x; \\
\textbf{define}\ main\ () \\
\quad b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_\textbf{uc}\ k_2; \\
\quad k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};
\end{array}
$$

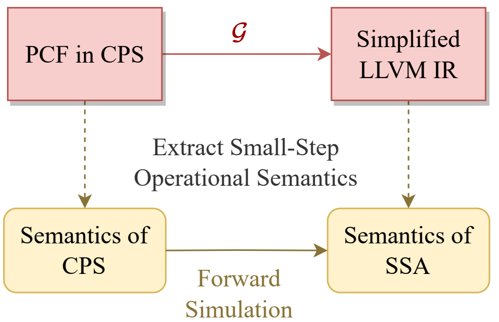
Observations:

- Continuations imply the ϕ-nodes we need to insert.
- Fresh variables make sure every variable is assigned once.

Table of Contents

1. Background & Motivation
2. Implementation of CPS $\Rightarrow$ SSA Transformation
3. Verification of CPS $\Rightarrow$ SSA Transformation
4. Evaluation & Conclusion

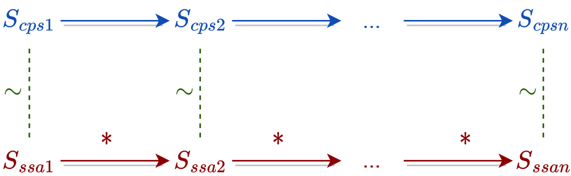
Verification via Simulation



What we need to verify via simulation:

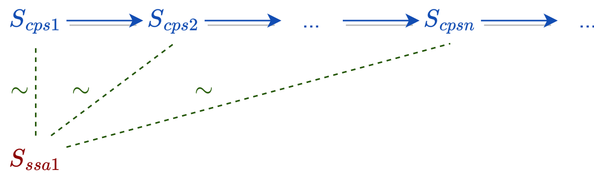
Target programs preserve the behavior of source programs.

Forward Simulation From CompCert



1. Invariant between the program states of CPS and SSA: $S_{cps} \sim S_{ssa}$
2. The invariant holds at the beginning: $\texttt{initial}\ (t_{cps}) \sim \texttt{initial}\ (t_{ssa})$.
3. Simulation for internal executions (Star Simulation).
4. Then we can derive: the invariant holds at the end.

Prevent Infinite Stuttering



- Define a measure function M for source states.
- M is strictly decreasing: $S_{cps1} \to S_{cps2},\ M(S_{cps1}) > M(S_{cps2})$.
- Stuttering could happen when a **letcont** term is evaluated.
- We use the number of **letcont** structures as M.

Forward Simulation

Take the CPS and SSA programs introduced before as an example:

CPS Program

$(\textbf{letcont}\ k_2\ y =$
$\quad k_{init}\ y\ \textbf{in}$
$\quad (f\ k_2\ x_3)))$

SSA Program

$\textbf{define}\ main\ ()$
$\quad b_1:\ x_3 = 2;\ \ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2;$
$\quad k_2:\ r_{k2} = y;\ \textbf{ret}\ r_{k2};$

$S_{cps1}:\ ((f\ k_2\ x_3),\ loc)$
$S_{ssa1}:\ (t_{ssa},\ (main, b_1, 1),\ (main, empty, 0),\ loc_{ssa},\ s_{empty})$

S_{cps1}
$\sim$
S_{ssa1}

Forward Simulation

Take the CPS and SSA programs introduced before as an example:

CPS Program

$$(\textbf{letcont}\ k_2\ y = k_{init}\ y\ \textbf{in}\ (f\ k_2\ x_3)))$$

SSA Program

$$\textbf{define}\ main\ ()$$
$$b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2;$$
$$k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};$$

S_{cps1} : $((f\ k_2\ x_3),\ loc)$
S_{ssa1} : $(t_{ssa},\ (main, b_1, 1),\ (main, empty, 0),\ loc_{ssa},\ s_{empty})$

S_{cps2} : $((k_{init}\ y),\ loc\ [k_2 \mapsto t_{cps}])$
S_{ssa2} : $(t_{ssa},\ (main, k_2, 0),\ (main, b_1, 1),\ loc_{ssa},\ s_{empty})$

$$\begin{array}{ccc} S_{cps1} & \Longrightarrow & S_{cps2} \\ \sim & & \sim \\ S_{ssa1} & \overset{*}{\Longrightarrow} & S_{ssa2} \end{array}$$

Forward Simulation

Take the CPS and SSA programs introduced before as an example:

CPS Program

$$(\textbf{letcont}\ k_2\ y = k_{init}\ y\ \textbf{in}\ (f\ k_2\ x_3)))$$

SSA Program

$$\textbf{define}\ main\ ()$$
$$b_1 :\ x_3 = 2;\ y = \textbf{call}\ f\ x_3;\ \textbf{br}_{\textbf{uc}}\ k_2;$$
$$k_2 :\ r_{k2} = y;\ \textbf{ret}\ r_{k2};$$

$S_{cps1} :\ ((f\ k_2\ x_3),\ loc)$

$S_{ssa1} :\ (t_{ssa},\ (main, b_1, 1),\ (main, empty, 0),\ loc_{ssa},\ s_{empty})$

$S_{cps2} :\ ((k_{init}\ y),\ loc\ [k_2 \mapsto t_{cps}])$

$S_{ssa2} :\ (t_{ssa},\ (main, k_2, 0),\ (main, b_1, 1),\ loc_{ssa},\ s_{empty})$

$$\begin{array}{ccccccc}
S_{cps1} & \Longrightarrow & S_{cps2} & \Longrightarrow & \dots & \Longrightarrow & S_{cpsn} \\
\sim & & \sim & & & & \sim \\
S_{ssa1} & \overset{*}{\Longrightarrow} & S_{ssa2} & \overset{*}{\Longrightarrow} & \dots & \overset{*}{\Longrightarrow} & S_{ssan}
\end{array}$$

Semantics

Small-step Operational Semantics of CPS

Judgement: $(t_{cps}, loc_{cps}) \to (t'_{cps}, loc'_{cps})$

$$\text{Rule: } \frac{loc_{cps}\ k = (\textbf{letcont}\ k\ x = t_1\ \textbf{in}\ t_2)}{(k\ v,\ loc_{cps}) \to (t_1[v/x],\ loc_{cps})}$$

...

Small-step Operational Semantics of SSA

Judgement: $(pc, ppc, loc_{ssa}, s_{ssa}) \to (pc', ppc', loc'_{ssa}, s'_{ssa})$

Rule:

$$\frac{\textbf{code}_{at}\ pc\ =\ (y = \textbf{call}\ f\ v_0)\quad \textbf{arg}\ f = x}{(pc,\ ppc,\ loc_{ssa},\ s_{ssa}) \to ((f, b_1, 0),\ pc,\ loc_{ssa}\ [x \mapsto v_0],\ \textbf{push}\ s_{ssa}\ pc)}$$

...

Forward Simulation

Define the invariant $\sim$:

Predicate between CPS program states and SSA program states

$$\frac{loc_{cps}\ k = \textbf{letcont}\ k\ x_1 = t\ \textbf{in}\ u \qquad \textbf{code}_{at}\ pc\ =\ x_1 = x \quad \textbf{code}_{at}\ (pc+1)\ =\ \textbf{br}_{\textbf{uc}}\ k}{(k\ x, loc_{cps}) \sim (t_{ssa}, pc, ppc, loc_{ssa}\ x_1 \mapsto x, s_{ssa})}$$

$$\frac{t_{cps} = \textbf{letcont}\ k\ x_1 = t\ \textbf{in}\ u \quad (u, loc_{cps}) \sim (t_{ssa}, pc, ppc, loc_{ssa}, s_{ssa}) \qquad (t, loc_{cps}\ k \mapsto t_{cps}) \sim (t_{ssa}, (pc.l_f, k, 0), pc, loc_{ssa}, s_{ssa})}{(t_{cps}, loc_{cps}\ k \mapsto t_{cps}) \sim (t_{ssa}, pc, ppc, loc_{ssa}, s_{ssa})}$$

$\ldots$

Table of Contents

1. Background & Motivation

2. Implementation of CPS $\Rightarrow$ SSA Transformation

3. Verification of CPS $\Rightarrow$ SSA Transformation

4. Evaluation & Conclusion

Development & Evaluation

Development is carried out in Coq
(PCF parser is implemented in OCaml).
The complete artifact can be accessed in Zenodo:
https://zenodo.org/record/7882331

May 1, 2023 Software Open Access

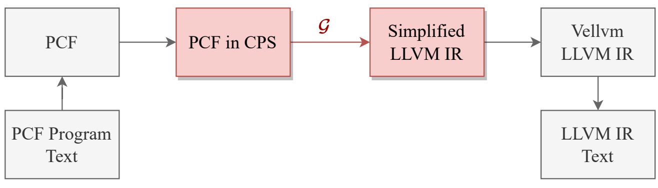
Verified transformation from CPS to SSA

Liu Siyu; Wang Yuting

Artifact for TASE23: Verified Transformation of Continuation-Passing Style into Static Single Assignment Form.

Categories	Contents	LOC	Proportion(%)
Language Definitions	PCF, CPS, SSA	702	23.9
Transformations	PCF→CPS, CPS→SSA, SSA→Vellvm LLVM IR	717	24.5
Verification	PCF→CPS Forward Simulation, CPS→SSA Forward Simulation, Combination of Forward Simulation, Backward Simulation	1513	51.6

Conclusion



- Provide a verified transformation algorithm from CPS to SSA.
- Build a prototype compiler for PCF that targets LLVM IR.
- Provides a foundation for developing verified functional compilers that may exploit the benefits of SSA compilation infrastructures.

In the future

Link verified functional compilers to verified SSA infrastructures.

Q & A

Thank You For Listening!