The Essence of Compiling with Continuations

Presentation of an article by C.Flamagan, A.Sabry, B.F.Duba and M.Felleisen

Bastien Thomas

Table of Contents

Table of Contents

Syntax of Core Scheme (CS)

$$
\begin{aligned}
M ::= {} & V \\
\mid {} & (\text{let } (x\ M_1)\ M_2) \\
\mid {} & (\text{if0 } M_1\ M_2\ M_3) \\
\mid {} & (M\ M_1 \ldots M_n) \\
\mid {} & (O\ M_1 \ldots M_n)
\end{aligned}
$$

$$V ::= c \mid x \mid (\lambda x_1 \ldots x_n.M)$$

- M Commands
- V Values
- c Constants
- x Variables
- O Primitive Operations ($+$, $\times$ $\ldots$)

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle$$
$$\to \langle\lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle\bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$
\begin{gathered}
\langle (\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop} \rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0) \rangle, \emptyset, \mathrm{stop} \rangle \rangle \\
\to^2 \langle (\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle \langle \mathrm{cl}\ x, x, \emptyset \rangle, \bullet \rangle, \emptyset, \mathrm{stop} \rangle \rangle
\end{gathered}
$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$\begin{gathered}
\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle \\
\rightarrow \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\rightarrow^2 \langle (\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle \langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\rightarrow \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet, 0\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle\rangle
\end{gathered}$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$
\begin{gathered}
\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle(\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet, 0\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle\rangle \\
\to^2 \langle 0, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle
\end{gathered}
$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$
\begin{gathered}
\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle(\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet, 0\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle\rangle \\
\to^2 \langle 0, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle x, [x := 0], \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle
\end{gathered}
$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$\begin{gathered}
\langle(\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle(\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet, 0\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle\rangle \\
\to^2 \langle 0, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle x, [x := 0], \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle x, [x := 0], \mathrm{stop}\rangle
\end{gathered}$$

Semantic of Core Scheme

Defined using the CEK-Machine (see Figure 2). Example :

$$
\begin{gathered}
\langle (\lambda x.x\ (\lambda x.x\ 0)), \emptyset, \mathrm{stop}\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet\ (\lambda x.x\ 0)\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle (\lambda x.x\ 0), \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to \langle \lambda x.x, \emptyset, \langle \mathrm{ap}\ \langle \bullet, 0\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle\rangle \\
\to^2 \langle 0, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle x, [x := 0], \langle \mathrm{ap}\ \langle\langle \mathrm{cl}\ x, x, \emptyset\rangle, \bullet\rangle, \emptyset, \mathrm{stop}\rangle\rangle \\
\to^2 \langle x, [x := 0], \mathrm{stop}\rangle \\
\to \langle \mathrm{stop}, 0\rangle
\end{gathered}
$$

Limitations of the CEK-Machine

- Every single function call will have to go through the call stack.
- Some optimisations like *tail-call* seem hard to implement.
- Problematic for functional languages with lots of function calls.

Table of Contents

Motivation

We want to effectively remove `return`-like statements from the program.
We do this by transforming the input program into a Continuation Passing Style (CPS) one.
For example the function $+$ would normally work like this:

$$(+\ a\ b) := \text{return } a + b$$

We will change it into something like:

$$(+'\ f\ a\ b) := (f\ (a + b))$$

The CPS Transformation

We can define a syntactic transformation $\mathcal{F}$ of a regular programm into a CPS one (See Figure 3).

The CPS Transformation

We can define a syntactic transformation $\mathcal{F}$ of a regular programm into a CPS one (See Figure 3). For example, $\mathcal{F}[\![(+\ a\ b)]\!]$ is :

$$\overline{\lambda}k.(\mathcal{F}[\![a]\!]\ \overline{\lambda}t_a.(\mathcal{F}[\![b]\!]\ \overline{\lambda}t_b.(+'\ k\ t_a\ t_b)))$$

$$\overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.((\overline{\lambda}k_b.(k_b\ b))\ \overline{\lambda}t_b.(+'\ k\ t_a\ t_b)))$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$\mathcal{F}[\![+ \; a \; b]\!] =$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$\mathcal{F}[\![+ \ a \ b]\!] = \overline{\lambda} k.((\overline{\lambda} k_a.(k_a \ a)) \ \overline{\lambda} t_a.((\overline{\lambda} k_b.(k_b \ b)) \ \overline{\lambda} t_b.(+' \ k \ t_a \ t_b)))$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$\begin{aligned}\mathcal{F}[\![+\ a\ b]\!] &= \overline{\lambda} k.((\overline{\lambda} k_a.(k_a\ a))\ \overline{\lambda} t_a.((\overline{\lambda} k_b.(k_b\ b))\ \overline{\lambda} t_b.(+'\ k\ t_a\ t_b))) \\ &\rightarrow_{\overline{\beta}} \overline{\lambda} k.((\overline{\lambda} k_a.(k_a\ a))\ \overline{\lambda} t_a.(\overline{\lambda} t_b.(+'\ k\ t_a\ t_b)\ b))\end{aligned}$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$
\begin{aligned}
\mathcal{F}[\![+\ a\ b]\!] &= \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.((\overline{\lambda}k_b.(k_b\ b))\ \overline{\lambda}t_b.(+'\ k\ t_a\ t_b))) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.(\overline{\lambda}t_b.(+'\ k\ t_a\ t_b)\ b)) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.(+'\ k\ t_a\ b))
\end{aligned}
$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$
\begin{aligned}
\mathcal{F}[\![+\ a\ b]\!] &= \overline{\lambda} k.((\overline{\lambda} k_a.(k_a\ a))\ \overline{\lambda} t_a.((\overline{\lambda} k_b.(k_b\ b))\ \overline{\lambda} t_b.(+'\ k\ t_a\ t_b))) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda} k.((\overline{\lambda} k_a.(k_a\ a))\ \overline{\lambda} t_a.(\overline{\lambda} t_b.(+'\ k\ t_a\ t_b)\ b)) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda} k.((\overline{\lambda} k_a.(k_a\ a))\ \overline{\lambda} t_a.(+'\ k\ t_a\ b)) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda} k.(\overline{\lambda} t_a.(+'\ k\ t_a\ b)\ a)
\end{aligned}
$$

Optimisations on CPS transformation

The CPS transformation adds quite a few $\overline{\lambda}$-abstractions to the program.
We can apply β reduction to simplify the terms.
For example :

$$
\begin{aligned}
\mathcal{F}[\![+\ a\ b]\!] &= \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.((\overline{\lambda}k_b.(k_b\ b))\ \overline{\lambda}t_b.(+'\ k\ t_a\ t_b))) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.(\overline{\lambda}t_b.(+'\ k\ t_a\ t_b)\ b)) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.((\overline{\lambda}k_a.(k_a\ a))\ \overline{\lambda}t_a.(+'\ k\ t_a\ b)) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.(\overline{\lambda}t_a.(+'\ k\ t_a\ b)\ a) \\
&\rightarrow_{\overline{\beta}} \overline{\lambda}k.(+'\ k\ a\ b)
\end{aligned}
$$

Structure of the CPS transformation of a program

Syntax of CS:

$$
\begin{aligned}
M ::= {} & V \\
& \mid (\text{let } (x\ M_1)\ M_2) \\
& \mid (\text{if0 } M_1\ M_2\ M_3) \\
& \mid (M\ M_1 \ldots M_n) \\
& \mid (O\ M_1 \ldots M_n)
\end{aligned}
$$

$$V ::= c \mid x \mid (\lambda x_1 \ldots x_n.M)$$

Syntax of the output of the CPS transformation

$$
\begin{aligned}
P ::= {} & (k\ W) \\
& \mid (\text{let } (x\ W)\ P) \\
& \mid (\text{if0 } W\ P_1\ P_2) \\
& \mid (W\ k\ W_1 \ldots W_n) \\
& \mid (W\ (\lambda x.P)\ W_1 \ldots W_n) \\
& \mid (O'\ k\ W_1 \ldots W_n) \\
& \mid (O'\ (\lambda x.P)\ W_1 \ldots W_n)
\end{aligned}
$$

$$W ::= c \mid x \mid (\lambda k x_1 \ldots x_n.P)$$

A specialized machine for CPS programs

We can define a Machine specifically for CPS programs (Figure 4). But in practice, we would use the machine (Figure 5). It differs mainly in two ways:

A specialized machine for CPS programs

We can define a Machine specifically for CPS programs (Figure 4). But in practice, we would use the machine (Figure 5). It differs mainly in two ways:

1. A keyword ar is added in order to distinguish between normal closures and continuation-induced ones.

A specialized machine for CPS programs

We can define a Machine specifically for CPS programs (Figure 4). But in practice, we would use the machine (Figure 5). It differs mainly in two ways:

1. A keyword ar is added in order to distinguish between normal closures and continuation-induced ones.
2. We devide the environment between E^- who give the valuation of the 'true' variables and E^k who contain the information on the continuations.

Table of Contents

Redundancy in the machine for CPS programs

The value k is nether used in the rule:

$$\langle (k, W), E^-, \langle \mathrm{ar}\ x, P', E_1^-, E_1^k \rangle \rangle \to_c^{(1)} \langle P', E_1^-[x := \mu(W, E^-)], E_1^k \rangle$$

The same thing happen in rules 4 and 5.

Redundancy in the machine for CPS programs

The value k is nether used in the rule:

$$\langle (k, W), E^-, \langle \mathrm{ar}\ x, P', E_1^-, E_1^k \rangle \rangle \rightarrow_c^{(1)} \langle P', E_1^-[x := \mu(W, E^-)], E_1^k \rangle$$

The same thing happen in rules 4 and 5. We can remove these redundancies with a transformation A(CS).
This optimisation is defined on Figure 6.

A simpler language

Syntax of CS:

$$
\begin{aligned}
M ::= {} & V \\
\mid {} & (\text{let } (x\ M_1)\ M_2) \\
\mid {} & (\text{if0 } M_1\ M_2\ M_3) \\
\mid {} & (M\ M_1 \ldots M_n) \\
\mid {} & (O\ M_1 \ldots M_n)
\end{aligned}
$$

$$V ::= c \mid x \mid (\lambda x_1 \ldots x_n.M)$$

Syntax of A(CS):

$$
\begin{aligned}
M ::= {} & V \\
\mid {} & (\text{let } (x\ V)\ M) \\
\mid {} & (\text{if0 } V\ M_1\ M_2) \\
\mid {} & (V\ V_1 \ldots V_n) \\
\mid {} & (\text{let } (x\ (V\ V_1 \ldots V_n))\ M) \\
\mid {} & (O\ V\ V_1 \ldots V_n) \\
\mid {} & (\text{let } (x\ (O\ V\ V_1 \ldots V_n))\ M)
\end{aligned}
$$

$$V ::= c \mid x \mid (\lambda x_1 \ldots x_n.M)$$

A Machine for A(CS)

Figure 8 defines a machine for A(CS).

A Machine for A(CS)

Figure 8 defines a machine for A(CS). Equivalence results have been shown. In particular, we can describe an equivalence relation between the realstic machine on CPS(CS) and the one on A(CS).

A Machine for A(CS)

Figure 8 defines a machine for A(CS). Equivalence results have been shown. In particular, we can describe an equivalence relation between the realstic machine on CPS(CS) and the one on A(CS). This means that the source code produced by both methods will be essentially the same.

Global view

So far, we have performed 3 major steps on the initial program.

Global view

So far, we have performed 3 major steps on the initial program.

1. We introduced continuations by using the CPS conversion.

Global view

So far, we have performed 3 major steps on the initial program.

1. We introduced continuations by using the CPS conversion.
2. We simplified the CPS program using β-reduction.

Global view

So far, we have performed 3 major steps on the initial program.

1. We introduced continuations by using the CPS conversion.
2. We simplified the CPS program using β-reduction.
3. We removed the continuations by reintroducing some contexts, resulting in the A conversion.

Global view

So far, we have performed 3 major steps on the initial program.

1. We introduced continuations by using the CPS conversion.
2. We simplified the CPS program using β-reduction.
3. We removed the continuations by reintroducing some contexts, resulting in the A conversion.

Step 3 can be seen as the inverse of step 1.

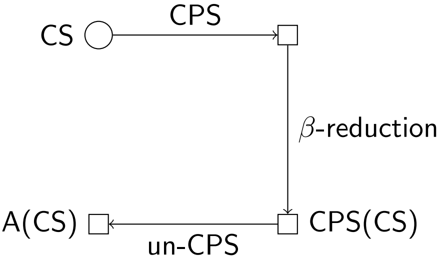
Global view

So far, we have performed 3 major steps on the initial program.

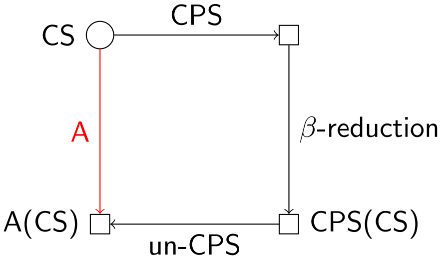
1. We introduced continuations by using the CPS conversion.
2. We simplified the CPS program using β-reduction.
3. We removed the continuations by reintroducing some contexts, resulting in the A conversion.

Step 3 can be seen as the inverse of step 1. The transformation A can be computed directly from CS in linear time.

Global View as a Drawing



Global View as a Drawing



Conclusion

This paper show that several compilation techniques can be condensed in a single transformation.

Conclusion

This paper show that several compilation techniques can be condensed in a single transformation.
This transformation is thus thought to be a ‘good’ intermediary procedure for optimizing compilers.

Conclusion

This paper show that several compilation techniques can be condensed in a single transformation.

This transformation is thus thought to be a ‘good’ intermediary procedure for optimizing compilers.

On unoptimized ML, speedup of 50% to 100%.

Conclusion

This paper show that several compilation techniques can be condensed in a single transformation.
This transformation is thus thought to be a ‘good’ intermediary procedure for optimizing compilers.
On unoptimized ML, speedup of 50% to 100%.
Some classical optimisations can be seen as β reductions on A(CS).