Construction & Extraction of EqMLton

Jordan Quinn

July 24, 2025

1 Construction

Given our MLton SSA, specified in [1], a construction for the CFG skeleton, S, follows. As our EqSat implementation operates on each function, we look at this structure in our SSA signature. First, we label each block with its index in the function's vector of blocks.

Then, we start at the first block and begin constructing its block in the CFG skeleton. To do this, we construct an e-graph from the statements in this block, mainly identical to the expected construction of a PEG-like e-graph. The most significant difference is that we will create a region for each node that represents the value of an assignment. When a variable already referred to by a region is used to calculate another variable, we create a new region that uses all the parameter nodes for the sub-regions, and we create a new root node that calculates the new variable using the root nodes of the sub-regions as children. This reuses the existing e-graph nodes and limits the parameters to those passed to this block.

Then, when we get to the transfer, we forget all the regions that were not used.

After the initial construction step, it would be useful to contract sequential blocks until we are left with only merge and join points.

1. Create an empty E-Graph, G, and an empty CFG Skeleton, K
2. Create an empty map from SSA variables to region in our E-Graph, m
3. For each global in our SSA, represent them in our E-Graph and add the variable label and root E-class to m as parameterless regions
4. (Only looking at one function, f, for now) Create a $\hat{param}$ node for each function parameter to f. Add these to m as regions that take one parameter
5. Let B be a DFS traversal of all blocks in f starting from its starting block
6. For each b in B:

(a) Create a new block in the CFG skeleton with the same label and block arguments as b

(b) Create a new empty map from SSA variable label to E-Graph region

```
$G : \textsc{FPeg} \leftarrow \textsc{InitFPeg}()$
$todo : \textsc{Set}[\textsc{SsaBlock}] = \{\textsc{FuncEntryBlock}\}$
$B : \textsc{Map}[\textsc{BlockLabel}, \textsc{SkBlock}]$
$m : \textsc{Map}[\textsc{SsaVariable}, \textsc{Region}] \leftarrow \textsc{InitMap}()$
for $\langle l : \tau = e \rangle$ in $\textsc{SsaGlobals}$ do
    $G, r \leftarrow \textsc{MakeRegion}(G, e)$
    $m \leftarrow m[l \mapsto r]$
end for
for $p = \langle l : \tau \rangle$ in $\textsc{FuncParams}$ do
    $G, r \leftarrow \textsc{MakeParam}(G, p)$
    $m \leftarrow m[l \mapsto r]$
end for
for $b$ in $todo$ do
    for $x = \langle l : \tau \rangle$ in $b_{args}$ do
        $G, r \leftarrow \textsc{MakeParam}(G, x)$
        $m \leftarrow m[l \mapsto r]$
    end for
    for $\langle l : \tau = e \rangle$ in $b_{assignments}$ do
        $G, r \leftarrow \textsc{MakeRegion}(G, e)$
        $m \leftarrow m[l \mapsto r]$
    end for
    for $l$ in $\textsc{Labels}(b_t) - B_{keys}$ do
        $todo \leftarrow todo \cup \{\textsc{GetSsaBlock}(l)\}$
    end for
    $t \leftarrow \textsc{MakeTransition}(m, b_t)$
    $\langle X : T \rangle \leftarrow b_{args}$
    $B \leftarrow B[b_{label} \mapsto \textsc{SkBlock}(b_{label}, X, T, t)]$
end for
$K_{root} \leftarrow \textsc{FuncEntryBlock}_{label}$
$K \leftarrow \textsc{Skeleton}(K_{root}, B, G)$
yield $K$
```

2 Extraction

```
function FpegToExpr($G$: Feg, $m$: Map[Region, SsaVariable], $r$: Region) : Expression
    if $r$ in $m_{keys}$ then return $m$.Get($r$)
    else
        $n \leftarrow$ ENodeSelect($G, r$)
        $es \leftarrow$ map(FpegToExpr($G, m$), $n$.Children)
        return $[\![(n_L($UnpackTuple($es$)$)]\!]$
    end if
end function
```

```
function SkTransToSsa($m$: Map[Region, SsaVariable], $t$: SkTrans) : SsaTrans
    if $\langle$SkTrans::Goto$_b(R_1, \ldots, R_n)\rangle = t$ then
        $R \leftarrow \{R_1, \ldots, R_n\}$
        if $R \cap$ MapKeys$(m) \neq R$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Goto$_b(x_1, \ldots, x_n)$
    else if $\langle$SkTrans::Call$_f(R_1, \ldots, R_n) \Rightarrow b_{ok}, b_{err}\rangle = t$ then
        $R \leftarrow \{R_1, \ldots, R_n\}$
        if $R \cap$ MapKeys$(m) \neq R$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Call$_f(x_1, \ldots, x_n; b_{ok}, b_{err})$
    else if $\langle$SkTrans::Raise$(R_1, \ldots, R_n)\rangle = t$ then
        $R \leftarrow \{R_1, \ldots, R_n\}$
        if $R \cap$ MapKeys$(m) \neq R$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Raise$(x_1, \ldots, x_n; b_{ok}, b_{err})$
    else if $\langle$SkTrans::Return$(R_1, \ldots, R_n)\rangle = t$ then
        $R \leftarrow \{R_1, \ldots, R_n\}$
        if $R \cap$ MapKeys$(m) \neq R$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Return$(x_1, \ldots, x_n; b_{ok}, b_{err})$
    else if $\langle$SkTrans::Match$(R_p; c_1 \Rightarrow b_1, \ldots, c_n \Rightarrow b_n)\rangle = t$ then
        if $R_p \notin$ MapKeys$(m)$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Match$(x_p; c_1 => b_1, \ldots, c_n \Rightarrow b_n)$
    else if $\langle$SkTrans::Match$(R_p; c_1 \Rightarrow b_1, \ldots, c_n \Rightarrow b_n; \Rightarrow b_d)\rangle = t$ then
        if $R_p \notin$ MapKeys$(m)$ then
            throw "To SSA called before regions were translated"
        end if
        Let $x_i =$ MapGet$(m, R_i)$
        return SsaTrans::Match$(x_p; c_1 => b_1, \ldots, c_n \Rightarrow b_n; b_d)$
    end if
end function
```

```
1: function SkBlockToSsa($Sk$: Skeleton, $m$: Map[Region, SsaVariable], $block$: SkBlock) : Set[SsaBlock]
2:     for $x_i$ in $block_X$ do
3:         $r_i \leftarrow$ GetParamRegionByVar($Sk_G, x$)
4:         $m \leftarrow$ MapInsert($m, r_i, x_i$)
5:     end for
6:     $assigns$ : List[SsaAssign] $\leftarrow$ [ ]
7:     Let $R$ be the set of regions referenced in $bl_t$
8:     Let $C$ be the closure of Children over all e-classes in $R$
9:     Let $C'$ be $C$ sorted ascending by region height
10:    for $r$ in $C'$ − MapKeys($m$) do
11:        $x \leftarrow$ FreshVar()
12:        $e \leftarrow$ FpegToExpr($Sk_G, m, r$)
13:        $m \leftarrow$ MapInsert($m, r, x$)
14:        $assigns \leftarrow$ ListAppend($assigns$, SsaAssign($x$, Type($r$), $e$))
15:    end for
16:    $B \leftarrow \emptyset$
17:    for $block'$ in DominatorTreeImmediateChildren($Sk, block$) do
18:        $B \leftarrow B \cup$ SkBlockToSsa($Sk, m, block'$)
19:    end for
20:    $t \leftarrow$ SkTransToSsa($m, block_t$)
21:    $block' \leftarrow$ SsaBlock($block_b, block_X, block_T, assigns, t$)
22:    return $B \cup \{block'\}$
23: end function
```

For the following, note that m would only contain the globals at this point in computation.

```
1: function SkFuncToSsa($Sk$: Skeleton, $f$: SkFunc, $m$: Map[Region, SsaVariable]) : SsaFunc
2:     for $x_i$ in $f_X$ do
3:         $r_i \leftarrow$ GetParamRegionByVar($Sk_G, x$)
4:         $m \leftarrow$ MapInsert($m, r_i, x_i$)
5:     end for
6:     $B \leftarrow$ SkBlockToSsa($Sk, m, f_{entry}$)
7:     return SsaFunc($f_{label}, f_X, f_T, f_{ret}, B$)
8: end function
```

References

[1] Matthew Fluet, Henry Cejtin, Suresh Jagannathan, and Stephen Weeks. Mlton ssa representation source code. `https://github.com/MLton/mlton/blob/master/mlton/ssa/ssa-tree.fun`, 2024.