One drawback of Maple is the lack of support of multi-table. In this short draft, one possible approach, generating multi-tables from compressed TTs, is presented. The technique can be used to translate TT to FTs of low-end switches which implement multi-tables according to OpenFlow specification 1.3 naively. Besides, it can be part of compilers for FTs of high-end switches which are increasingly adopting SAI. We use the following AP as example:

```
1  Map macTable(Key: macAddress, Value: sw)
2    onPacket(p):
3      sw = macTable[p.srcMac]
4      dw = macTable[p.dstMac]
5      if sw == null || dw == null
6        Drop
7      else
8        return shortestPath(sw, dw)
```

Figure 1 demonstrates the redundancies in TT and the explosion of FT. In worst case, V node *srcMac* has 2^{48} children, each of which has 2^{48} children, resulting a tree with a total number of 2^{96} leaves. As shown in Figure 1, many siblings are redundant, in the sense that their parent can reference to the same child, and cut the others. Eliminating redundancies in TT can avoid explosion of FT by reducing the number of leaves.

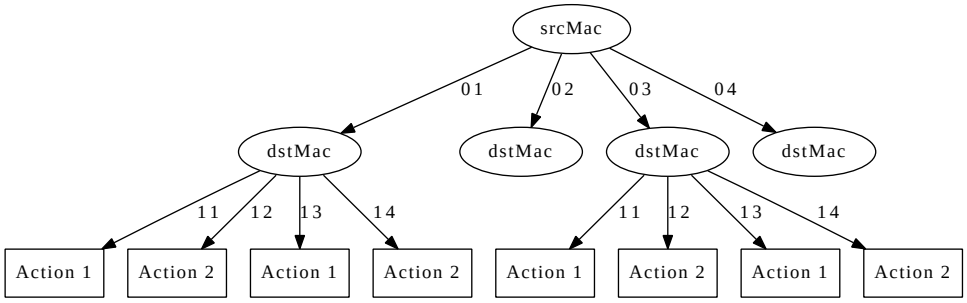

Figure 1: Fan-out in TT

1 Compressing TT

Algorithm ConvergeTT can be used to recursively remove redundancies in TT. This algorithm starts from the root of the TT, performs a depth-first tree traversal and removes redundancies from the bottom up. For each V node, if any pair of two children are the same (can be derived from a Merkel Hash Tree), one of the children is cut and both egress edges point to the remaining one, turning a tree into a DAG. If any pair of children is similar (can be derived from a Merkel Hash Tree), they can be merged (TODO job).

Figure 3 shows the TT after applying Converge() on the leftmost subtree

```
1  Algorithm ConvergeTT(t)
2      for node in t
3          node.converged = false;
4      Converge(t);
5      return;
6
7  Procedure Converge(t)
8      if type_t = L then
9          t.converged = true;
10
11     else if type_t = T then
12         Converge(t_-);
13         Converge(t_+);
14         t.converged = true;
15
16     else if type_t = V then
17         for v_1 in keys(subtrees_t) do
18             if (v_1.converged == false)
19                 Converge(v_1);
20             for v_2 in keys(subtrees_t - v_1) && v_2.converged do
21                 if( v_1 Equal v_2) then // Hask Tree
22                     v_2 points to the same child of v_1;
23                     delete subtrees_v2;
24                 else if (v_1 ⊂ v_2 || v_2 ⊂ v_1)
25                     merge(v_1, v_2) // Not well defined yet.
26         t.converged = true;
27     endif
```

Figure 2: *Algorithm ConvergeTT. For every node in the TT, field* converged *is initialized as false. The algorithm performs a depth-first tree traversal on tree t.*

(srcMac == 01), and Figure 4 shows the TT after applying Converge() on the whole tree.

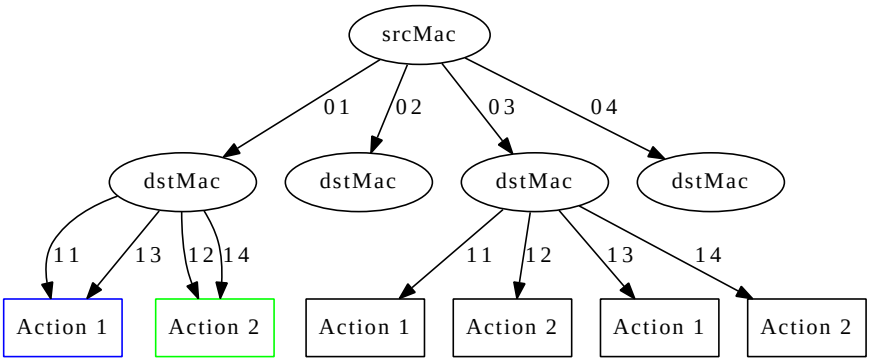

Figure 3: Converge the leftmost subtree of TT. Converge happens at nodes in color.

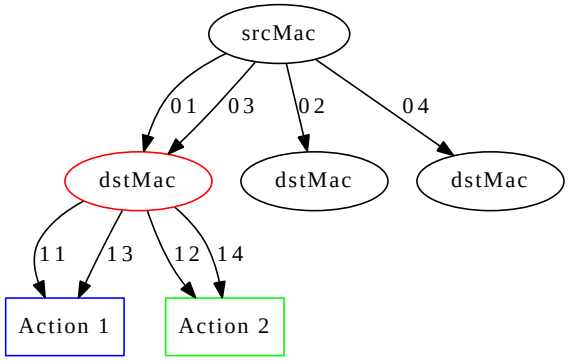

Figure 4: Converge the whole TT. The compressed TT is a DAG. Converge happens at nodes in color.

2 Generating multi-tables from compressed TT — GenerateFTS version 0.1

In GeneratFTS version 0.1, we make the following assumptions:

- The target of the FTs generated by GenerateFTS() is a virtual OpenFlow switch which provides unlimited number of common flow tables. As a result, each V node can be mapped to a dedicated flow table, and these tables will be organized as a DAG, which enables a combination of pipleline parallelism and task parallelism on switch.
- We use *metadata* defined in OpenFlow 1.3 to pass information from one table to its subsequent tables. We assume the *metadata* has unlimited bit-width.

2.1 Generating FTs from TT without T nodes

We start from TT without T nodes (shown in Table 5) to highlight the basic idea of the algorithm. Generated FTs are shown in Table 5. Each V node will be mapped to a dedicated table. Each entry of the table corresponds to an egress edge from this V node.

All of the edges point to the same child are refereed to as an EdgeSet. Each EdgeSet has a unique EdgeID, which is an integer increasing from 0. For example, edges srcMac=01 and srcMac=03 are in the EdgeSet 1. The action of each entry is to log the SetID of the egress edge.

We call the last table in the pipeline Table-ACTION, where the match field is the cumulated maps from egress edges of V nodes to their EdgeID numbers.

Figure 5: FT of TT shown in Figure 4

	Pri	Match	Action
Table0	0	srcMac=01	reg1=1; goto 1;
	0	srcMac=03	reg1=1; goto 1;
	0	srcMac=02	reg1=2; goto 2;
	0	srcMac=04	reg1=3; goto 3;
	0	otherwise	reg1=NIL; goto ACTION;
	Pri	Match	Action
Table1	0	dstMac=11	reg2=1; goto ACTION;
	0	dstMac=13	reg2=1; goto ACTION;
	0	dstMac=12	reg2=2; goto ACTION;
	0	dstMac=14	reg2=2; goto ACTION;
	Pri	Match	Action
Table2	0	otherwise	reg3=NIL; goto ACTION;
	Pri	Match	Action
Table3	0	otherwise	reg4=NIL; goto ACTION;
	Pri	Match	Action
Table-ACTION	0	reg1=1; reg2=1;	Action1
	0	reg1=1; reg2=2;	Action2
	0	reg1=NIL	drop
	0	reg1=2; reg3=NIL;	drop
	0	reg1=3; reg4=NIL;	drop

2.2 Generating FTs from TT with T nodes

GenerateFTS() generates FTs from TT. It performs a depth-first tree traversal. To generate FTs from TT with T nodes, each V node in conjunction with its egress edges and subsequent T nodes (without any other V nodes in between the V node and T nodes) are mapped to a dedicated table. For example, in Figure 6, V node srcMac, T node inport=1, and their egress edges are mapped

to table0. Other aspects are similar to previous subsection. The FTs of Figure 6 is shown in Figure 7.

```
1  Struct parent_info {
2      ID;
3      priority;
4      match;
5      action;
6  }
7
8  // emitRule(tableID, match, priority, action);
9  // GenerateSingleFT(node, metadata, parent_info);
10
11 Algorithm GenerateFTS(t)
12     tableID = 0;
13     GenerateSingleFT(t, ∅, ∅);
14
15 Procedure GenerateSingleFT(t, md, p)
16     if type_t = L node then
17         // insert rule in parent table
18         action = p.action + "goto table-ACTION";
19         emitRule(p.ID, p.match, p.priority, action);
20
21         emitRule(table-ACTION, md, p.priority, value_t);
22     endif
23
24     if type_t = T node then
25         GenerateSingleFT(t_−, md, p);
26         p.match = p.match ∧ (attr_t : value_t);
27         emitRule(p.ID, p.match , p.priority, punt);
28         p.priority = p.priority + 1;
29         GenerateSingleFT(t_+, md, p);
30     endif
31
32     if type_t = V then
33         tableID_t = tableID ++;
34         // insert rule in parent table
35         p.action = p.action + "goto tableID";
36         emitRule(p.ID, p.match, p.priority, p.action);
37
38         for (EdgeSet, setID) ∈ EdgeSets(t) do
39             md = md ∧ (Reg[tableID] : setID);
40             parent_info p_t(tableID, p.priority, EdgeSet, "Reg[tableID] = setID");
41             GenerateSingleFT(subtree[EdgeSet], md, p_t);
42         endfor
43     endif
```

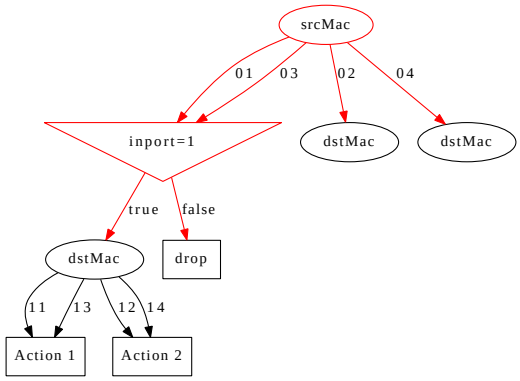

Figure 6: Converge the whole TT. The compressed TT is a DAG. Nodes and edges in red are mapped to a single table.

Figure 7: FTs generated for multi-tables

	Pri	Match	Action
Table0	0	srcMac=01	m.reg1=1; goto ACTION;
	0	srcMac=01, inport=1	punt
	1	srcMac=01, inport=1	m.reg1=1; goto 1;
	0	srcMac=03	m.reg1=1; goto ACTION;
	0	srcMac=03, inport=1	punt
	1	srcMac=03, inport=1	m.reg1=1; goto 1;
	0	srcMac=02	reg1=2; goto 2;
	0	srcMac=04	reg1=3; goto 3;
	0	otherwise	reg1=NIL; goto ACTION;
	Pri	Match	Action
Table1	1	dstMac=11	reg2=1; goto ACTION;
	1	dstMac=13	reg2=1; goto ACTION;
	1	dstMac=12	reg2=2; goto ACTION;
	1	dstMac=14	reg2=2; goto ACTION;
	Pri	Match	Action
Table2	0	otherwise	reg3=NIL; goto ACTION;
	Pri	Match	Action
Table3	0	otherwise	reg4=NIL; goto ACTION;
	Pri	Match	Action
Table-ACTION	0	reg1=1	drop
	1	reg1=1; reg2=1;	Action1
	1	reg1=1; reg2=2;	Action2
	0	reg1=NIL	drop
	0	reg1=2; reg3=NIL;	drop
	0	reg1=3; reg4=NIL;	drop