Improving Your TABLEGEN Description

Javed Absar

GRAPHCORE

WHAT IS ‘TABLEGEN’ ?

- DSL invented for LLVM
- Used extensively
- Describe records of –
  - Instructions, registers, intrinsics, attributes, scheduler model, warnings, …

THIS TALK IS **NOT…**

- Introduction to TableGen
- Complete Guide to TableGen

BUT…

Look at interesting features that you may

- Not know exist
- Know, but haven’t used
- Used, but not this way

‘FOREACH’ AND ‘CONCAT’

Listing One-by-One

```
def F9Dwarf  : DwarfMapping<28>;
def F11Dwarf : DwarfMapping<29>;
def F13Dwarf : DwarfMapping<30>;
def F15Dwarf : DwarfMapping<31>;


def F16Dwarf : DwarfMapping<68>;
def F18Dwarf : DwarfMapping<69>;
def F20Dwarf : DwarfMapping<70>;
def F22Dwarf : DwarfMapping<71>;
```

Source:

‘FOREACH’ AND ‘CONCAT’

Listing One-by-One

```
def F9Dwarf  : DwarfMapping<28>;
def F11Dwarf : DwarfMapping<29>;
def F13Dwarf : DwarfMapping<30>;
def F15Dwarf : DwarfMapping<31>;


def F16Dwarf : DwarfMapping<68>;
def F18Dwarf : DwarfMapping<69>;
def F20Dwarf : DwarfMapping<70>;
def F22Dwarf : DwarfMapping<71>;
```

Generating using ‘foreach’

```
1. def RegMap  {
2.   list<list<int>> val = [[9,28], [11,29], [13,30],
                            [15, 31],  [16, 68], [18, 69],
                            [20, 70], [22, 71]];
3. }


4. foreach N = RegMap.val  in {
5.   def F#!head(N)#Dwarf :
                    DwarfMapping<!head(!tail(N))>;
6. }
```

Source:

‘FOLD’

Listing One-by-One

```
def {
  list<int> Ascending = [1, 2, 3, 4, 5, 6, 7, 8, 9];
  list<int> Descending = [9, 8, 7, 6, 5, 4, 3, 2, 1];
}
```

Source:

‘FOLD’

Listing One-by-One

```
1. def {
2.   list<int> Ascending = [1, 2, 3, 4, 5, 6, 7, 8, 9];

3.   // list<int> Descending = [9, 8, 7, 6, 5, 4, 3, 2, 1];
4.   list<int> Descending = reverse<Ascending>.val;
5. }
```

Generating using ‘foldl’

```
1. class reverse<list<int> L> {
2.    list<int> val = !foldl([ ]<int>, L, a, b, [b] # a);
3. }
```

Source:

‘SORT’ USING TABLEGEN

Merge Sort Algorithm

```
def mergesort(L):
    if len(L) < 2:
        return L
    else:
        h = len(L) div 2
        return merge(mergesort(L[:h]), mergesort(L[h:]))
```

Source:

‘SORT’ USING TABLEGEN

Merge Sort Algorithm

```
def mergesort(L):
    if len(L) < 2:
        return L
    else:
        h = len(L) div 2
        return merge(mergesort(L[:h]), mergesort(L[h:]))
```

Using TableGen

```
1. class mergeSort<list<int> L> {
2.   list<int> val = !if(!lt(!size(L),2), L,
3.                       merge<
4.                             mergeSort< split<L>.lower >.val,
5.                             mergeSort< split<L>.upper >.val
6.                           >.val );
7. }
```

Source:

‘SORT’ USING TABLEGEN

Using TableGen

```
1. class split<list<int> L> {
2.    int sizeW = !size(L);

3.    int sizeL = !srl(!add(sizeW,1),1);
4.    int sizeU = !add(sizeW,!mul(sizeL,-1));

5.    list<int> lower = !foldl([ ]<int>, L, a, b, !if(!lt(!size(a),sizeL),
6.                                                    a # [b],
7.                                                    a));
8.   list<int> upper = !foldl(L, L, a, b, !if(!gt(!size(a), sizeU),
9.                                             !tail(a),
10.                                            a));
11.}
```

Source:

‘SORT’ USING TABLEGEN

Merge Function (Functional Programming)

```
def merge(a, b):
    if len(a) == 0: return b
    elif len(b) == 0: return a
    elif a[0] < b[0]:
        return [a[0]] + merge(a[1:], b)
    else:
        return [b[0]] + merge(a, b[1:])
```

Source:

‘SORT’ USING TABLEGEN

Merge Function (Functional Programming)

```
def merge(a, b):
    if len(a) == 0: return b
    elif len(b) == 0: return a
    elif a[0] < b[0]:
        return [a[0]] + merge(a[1:], b)
    else:
        return [b[0]] + merge(a, b[1:])
```

Using TableGen

```
1. class merge<list<int> a, list<int> b> {
2.    list<int> val = !if(!eq(!size(a), 0), b,
3.                        !if(!eq(!size(b),0), a,
4.                            !if(!lt(!head(a), !head(b)), [!head(a)] # merge<!tail(a), b>.val,
5.                                [!head(b)] # merge<a, !tail(b)>.val)));
6. }
```

Source:

STATIC POLYMORPHISM WITH TABLEGEN

Test Code

```
1. class GeoObj { string draw = "Invalid"; }

2. class Line<int N> : GeoObj { string draw = "Line:" # N; }
3. class Rectangle<int N> : GeoObj { string draw = "Rectangle:" # N; }

4. class myDraw<list<GeoObj> objs> {
5.       list<string> drawings = !foreach(obj, objs, obj.draw);
6. }

7. def main {
8.       GeoObj o1 = Line<1>;   GeoObj o2 = Line<2>;   GeoObj o3 = Rectangle<3>;
9.        list<string> D = myDraw<[o1, o2, o3]>.drawings;
10. }
```

Source:

STATIC POLYMORPHISM WITH TABLEGEN

Test Code

```
1. class GeoObj { string draw = "Invalid"; }
2. class Line<int N> : GeoObj { string draw = "Line:" # N; }
3. class Rectangle<int N> : GeoObj { string draw = "Rectangle:" # N; }
4. class myDraw<list<GeoObj> objs> {
5.       list<string> drawings = !foreach(obj, objs, obj.draw);
6. }

7. def main {
8.       GeoObj o1 = Line<1>;   GeoObj o2 = Line<2>;   GeoObj o3 = Rectangle<3>;
9.        list<string> D = myDraw<[o1, o2, o3]>.drawings;
10. }
```

Generated Definition

```
$ ./bin/llvm-tblgen static_polymorphism.td
def main {
  …
 list<string> D = ["Line:1", "Line:2", "Rectangle:3"];
}
```

Source:

‘COND’

Using Nested-‘IF’s

```
class getSubRegs<int size> {
  list<SubRegIndex> ret2 = [sub0, sub1];
  list<SubRegIndex> ret3 = [sub0, sub1, sub2];
  list<SubRegIndex> ret4 = [sub0, sub1, sub2, sub3];
  list<SubRegIndex> ret5 = [sub0, sub1, sub2, sub3, sub4];
  list<SubRegIndex> ret8 = [sub0, sub1, sub2, sub3, sub4, sub5,
sub6, sub7];
  list<SubRegIndex> ret16 = [sub0, sub1, sub2, sub3,
                     sub4, sub5, sub6, sub7,
                     sub8, sub9, sub10, sub11,
                     sub12, sub13, sub14, sub15];
  list<SubRegIndex> ret = !if(!eq(size, 2), ret2,
                                !if(!eq(size, 3), ret3,
                                   !if(!eq(size, 4), ret4,
                                      !if(!eq(size, 5), ret5,
                                         !if(!eq(size, 8), ret8,
ret16))))); }
```

Using ‘COND’

```
class getSubRegs<int size> {
  list<SubRegIndex> ret2 = [sub0, sub1];
  list<SubRegIndex> ret3 = [sub0, sub1, sub2];
  list<SubRegIndex> ret4 = [sub0, sub1, sub2, sub3];
  list<SubRegIndex> ret5 = [sub0, sub1, sub2, sub3, sub4];
  list<SubRegIndex> ret8 = [sub0, sub1, sub2, sub3, sub4, sub5,
sub6, sub7];

 list<SubRegIndex> ret_opt = !cond(!eq(size, 2):  ret2,
                                   !eq(size, 3):  ret3,
                                   !eq(size, 4):  ret4,
                                   !eq(size, 5):  ret5,
                                   !eq(size, 8):  ret8,
                                   !eq(1, 1):     ret16);
}
```

Source:

CONCLUSION

Take a look at the video and apply it to your context!