Embellished Constantpropagation

Code:

```
begin
   proc square(val z, res v) is
      v := z * z;
   end
   proc doublesquare(val z, res v) is
      call square(z, v);
      call square(v,v);
   end
   call doublesquare(7,y);
end
```

Generate information from the program:

run (ecp 2) “doublesquare”

```
OUTPUT:
Block:
Label 1: proc square(val z,, res v) is
Label 2: v := z * z
Label 3: end
Label 4: proc doublesquare(val z,, res v) is
Label 5: Call square(z, v) Entry
Label 6: Call square(z, v) Exit
Label 7: Call square(v, v) Entry
Label 8: Call square(v, v) Exit
Label 9: end
Label 10: Call doublesquare(7, y) Entry
Label 11: Call doublesquare(7, y) Exit

Flow:      {(1,2),(2,3),(4,5),(5,1),(3,6),(6,7),(7,1),(3,8),(8,9),(10,4),(9,11)}

InterFlow: {(5,1,3,6),(7,1,3,8),(10,4,9,11)}
Labels:    {1,2,3,4,5,6,7,8,9,10,11}
Init:      {10}
Finals:    {11}
Variables: {"v","y","z"}
```

```
Analysis:
1
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => T,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => T,y => T,z => 49}
}
 =>
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => T,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => T,y => T,z => 49}
}

2
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => T,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => T,y => T,z => 49}
}
 =>
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => 49,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => 2401,y => T,z => 49}
}

3
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => 49,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => 2401,y => T,z => 49}
}
 =>
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => 49,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => 2401,y => T,z => 49}
}
```

```
4
{\[]    -> {}
,\[10] -> {υ => T,y => T,z => 7}
}
 =>
{\[]    -> {}
,\[10] -> {υ => T,y => T,z => 7}
}

5
{\[]    -> {}
,\[10] -> {υ => T,y => T,z => 7}
}
 =>
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => T,y => T,z => 7}
}

6
{\[]     -> {}
,\[5]    -> {}
,\[5,10] -> {υ => 49,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => 2401,y => T,z => 49}
}
 =>
{\[]    -> {}
,\[10] -> {υ => 49,y => T,z => 7}
}

7
{\[]    -> {}
,\[10] -> {υ => 49,y => T,z => 7}
}
 =>
{\[]     -> {}
,\[7]    -> {}
,\[7,10] -> {υ => T,y => T,z => 49}
}
```

```
8
{\[]      -> {}
,\[5]    -> {}
,\[5,10] -> {υ => 49,y => T,z => 7}
,\[7]    -> {}
,\[7,10] -> {υ => 2401,y => T,z => 49}
}
 =>
{\[]    -> {}
,\[10] -> {υ => 2401,y => T,z => 7}
}

9
{\[]    -> {}
,\[10] -> {υ => 2401,y => T,z => 7}
}
 =>
{\[]    -> {}
,\[10] -> {υ => 2401,y => T,z => 7}
}

10
{\[] -> {υ => T,y => T,z => T}
}
 =>
{\[]    -> {}
,\[10] -> {υ => T,y => T,z => 7}
}

11
{\[]    -> {}
,\[10] -> {υ => 2401,y => T,z => 7}
}
 =>
{\[] -> {υ => T,y => 2401,z => T}
}
```

Analysis:

To start with, we have specifically chosen for a recursive procedure program for two reasons: First of all, the analysis is way too big, which makes it difficult for us to explain everything. A smaller example such as this one will do a lot of the same and will have a smaller analysis result. Secondly this example has a nice outcome that shows that our analysis is correct. Fib for example will always just result in seeing Top only, which is then also more difficult to prove that it is correct.

For this analysis we will do the walk through in the following order: 10 → 4 → 5 → 1 → 2 → 3 → 6 → 7 → 8 → 9 → 11

10: This is the start of the program and for constant propagation our extreme value is the lattice where every known variable is assigned Top. Furthermore for embellished constant propagation we start with a context lattice. Meaning a map of contexts paired with lattices. We will start with the empty context that is paired with the extreme value. At label 10 we have a Call to doublesquare given a value 7 for an argument and the result will be stored in y. What we do here is we add the label 10 to all existing contexts. In this case only the empty context. After that we add the empty context with no lattice i.e. bottom. Next we have to change the value of the parameter of the procedure to the argument passed in the Call. Z is the parameter and should be given the value of 7 , which has happened.

4: This is the procedure entry. We have defined this as the identity function. Thus it doesn't do anything noteworthy. Notice that there is only one non-zero context: 10. Meaning that there could have been at most 1 call to this procedure (unless k = 1), which is of course correct.

5: We now get a Call to the procedure square. Remember that we only pass this Call once for the same reason as with 4 and thus starts with the same contexts. We again add the call label to all existing contexts. Furthermore we have to pass values to the parameters of square, but because these are the same as doublesquare, we do not see a change.

1: We can see here that there have been made multiple calls to this procedure. Namely from 5 and 7. They also show that a call from 10 before calling 5 and 7 was made. Thus correctly passing variables to the parameters. From the call from 5 we can see the value 7 passed to parameter z, which is still the value of the argument in the call to doublesquare. From the call from label 7 we can see the value 49 passed to parameter z. In that call we must have already called from label 5 and returned from label 5 for the value to be then passed to label 7. The value 7 was square by the call from label 5, with the result being 49. We can now clearly see that this value was then on passed to the call from label to the procedure to the parameter z.

2: Here we do our squaring. If you look at the contexts [5,10] and [7,10] you can see that 7 and 49 are square to 49 and 2401 respectively and of course, afterward being stored in the variable v. The transfer is applied to all contexts.

3: Also for procedure exits do we do identity functions.

6: Now we have to do the return from a call. When we return we have to take care of unshadowing and also make sure the result of the procedure is stored in the right variable. What we do is we look up which call label this return belongs to. For this return that is label 5. Then we remove any contexts that do not start with label 5. Then the label 5 is removed from all remaining contexts. Finally we'll try to merge the context of the call and the context of the return. By merging we only merge contexts that are in both the call and the return context, any others are discarded. By merging we mean unshadowing and stored the result in the right variable. But because the parameters of the doublesquare and square are the same we see no difference.

7: Now we make the second call to square, now adding the label 7 to the contexts and passing the value 49 to the parameter and storing the value 2401.

8: Similarly to 6, but then for contexts starting with label 7.

9: Again just a identity function.

11: Now we can do our final return. There is only one context left: [10] that we can use. It of course works the same as the previous returns, but now we can also see that the result of the procedure is correctly stored in the variable y.