

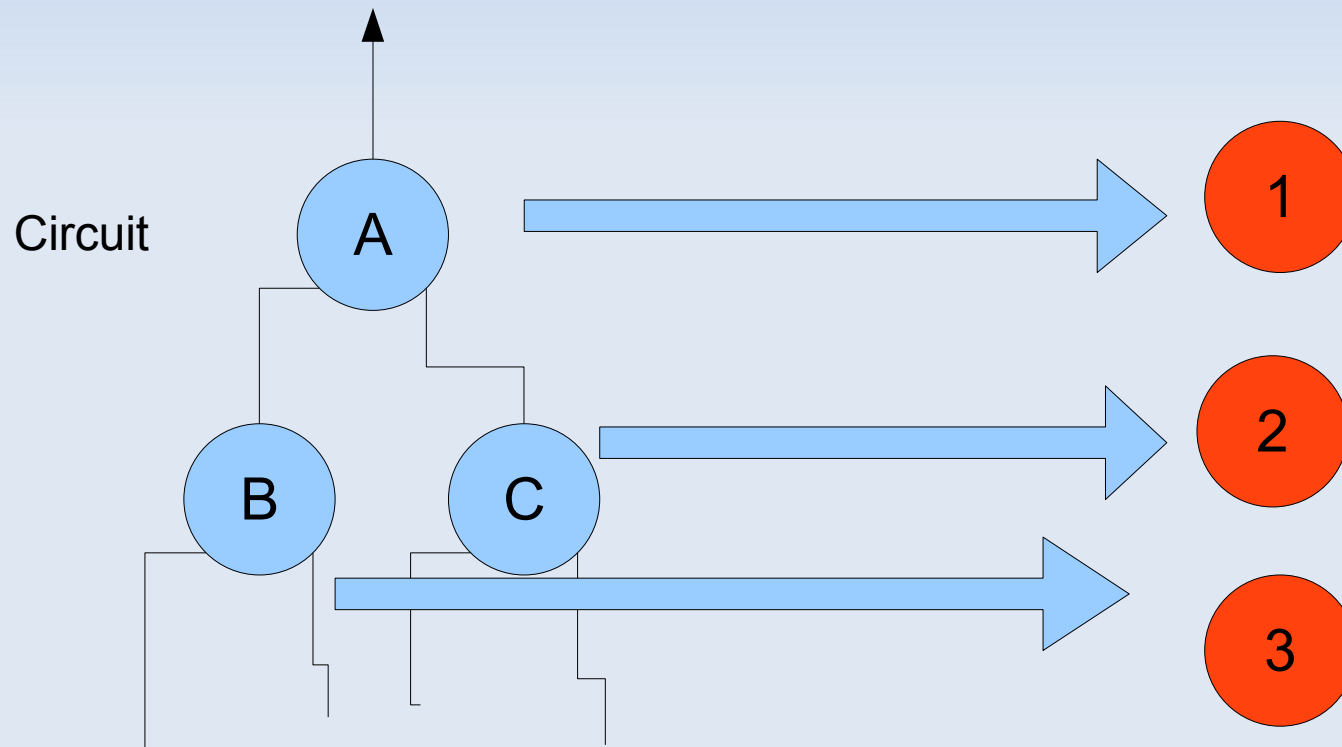
# wetColour / dryColour Fields

Supporting Large Circuits with Only  $2^{16}-1$  BDD  
Indices

Andrew Ling

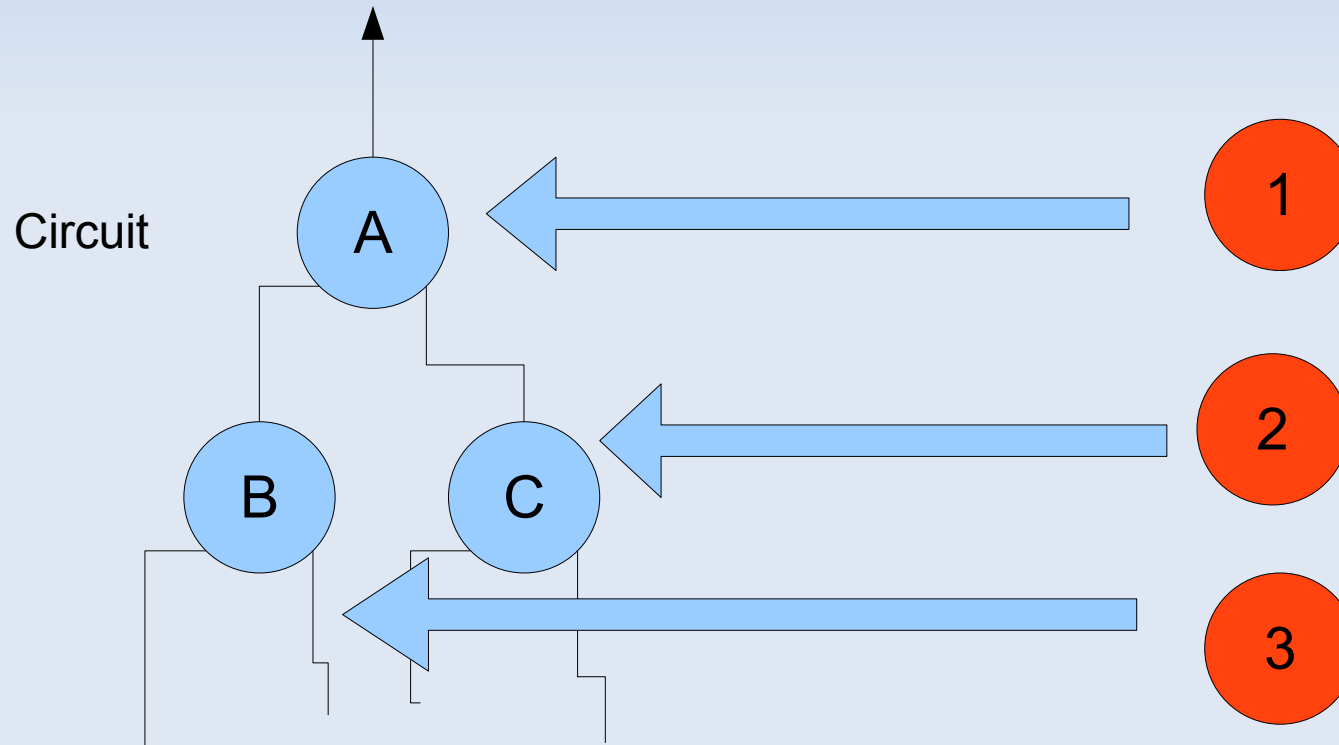
# BddCut, use of indeces

- Each graph node is assigned a BDD variable



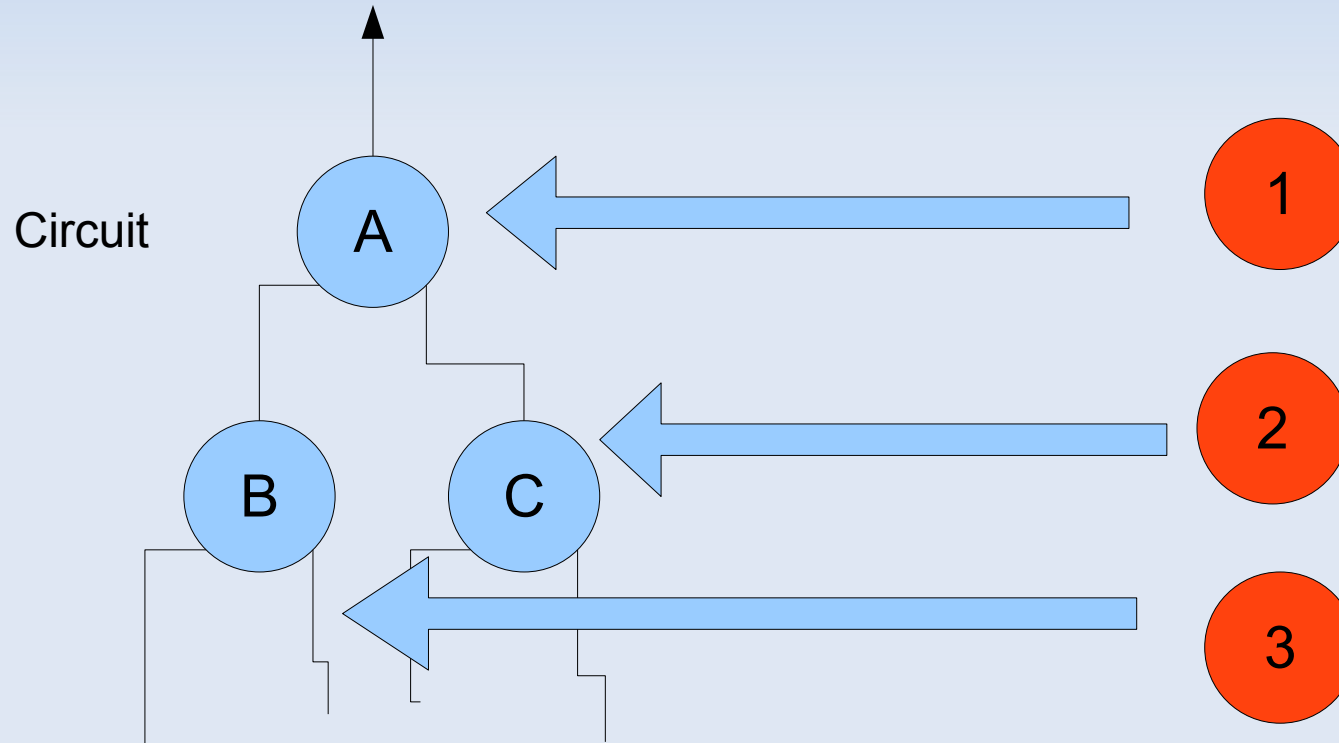
# BddCut, use of indeces

- Needs to have a one-to-one assignment



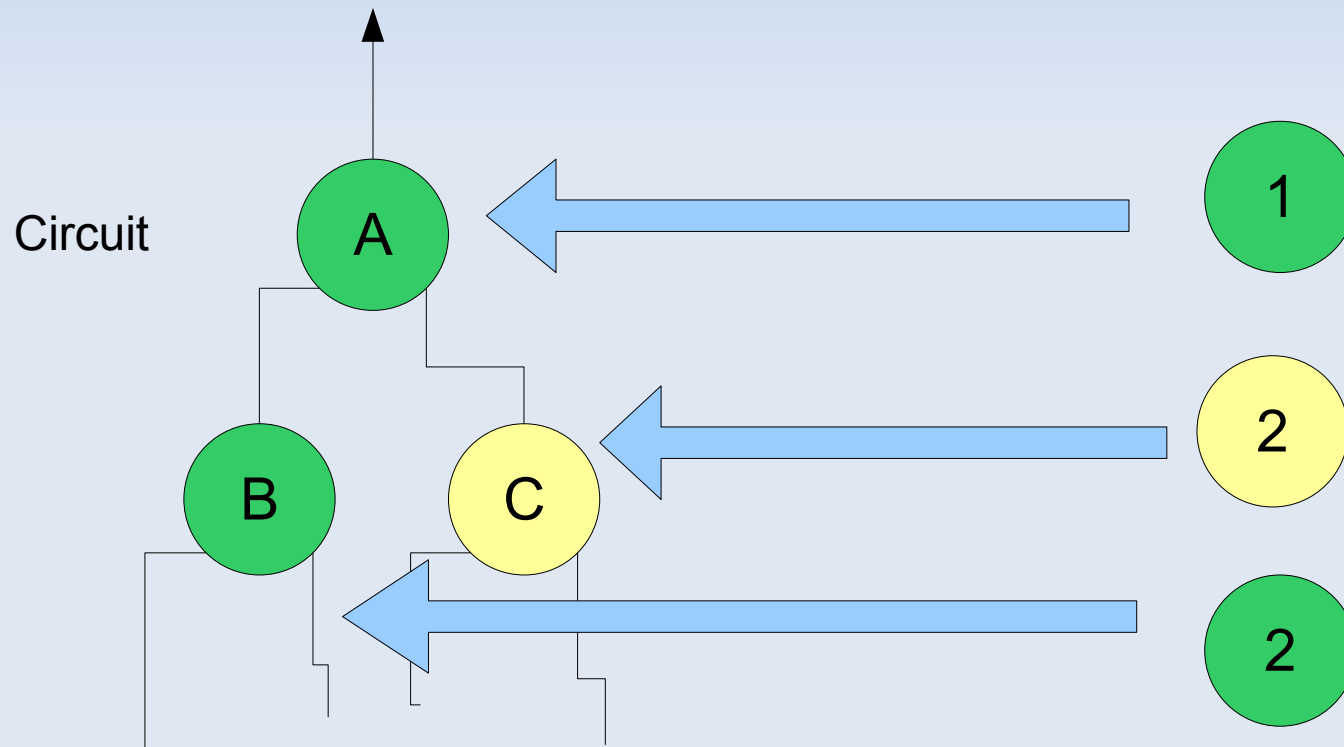
# BddCut, use of indeces

- Needs to have a one-to-one assignment
  - There are only  $2^{16}-1$  BDD indeces, can potentially have millions of circuit nodes



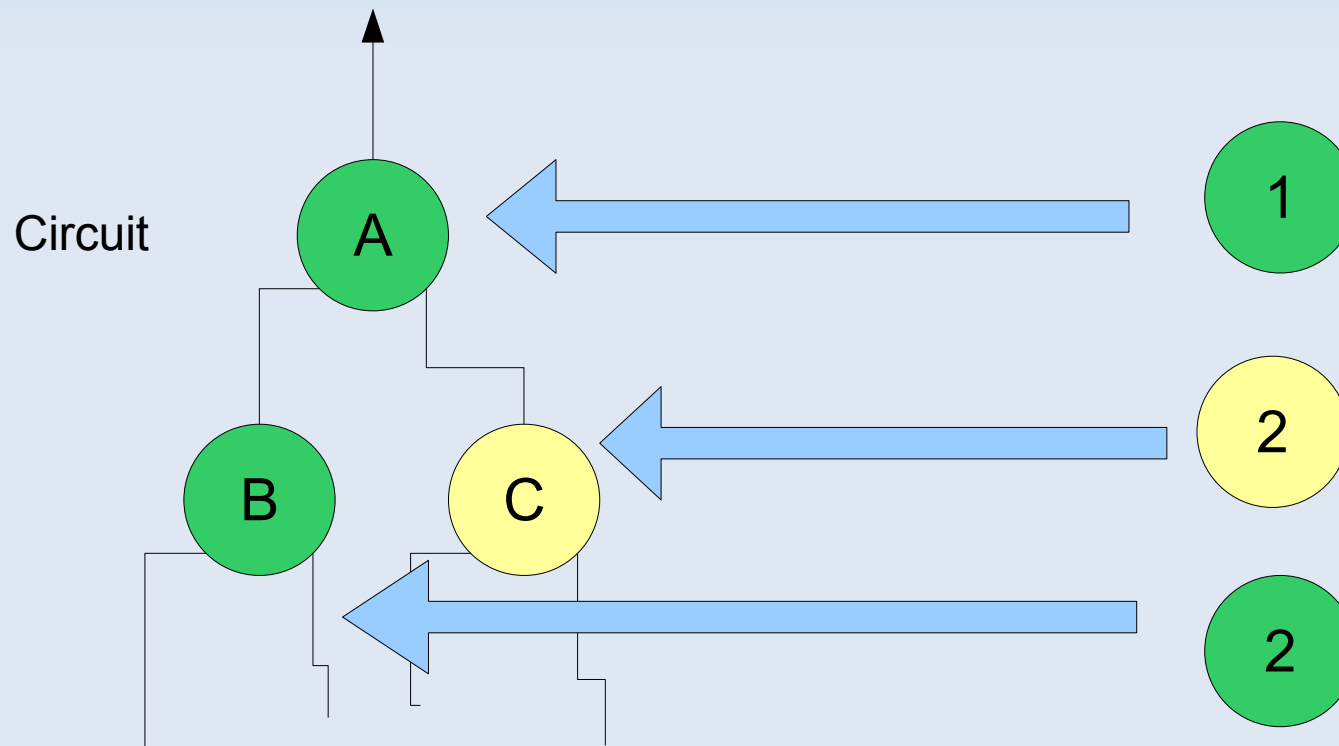
# Bdd Index Domains

- Use a colour index to represent a BDD index domain



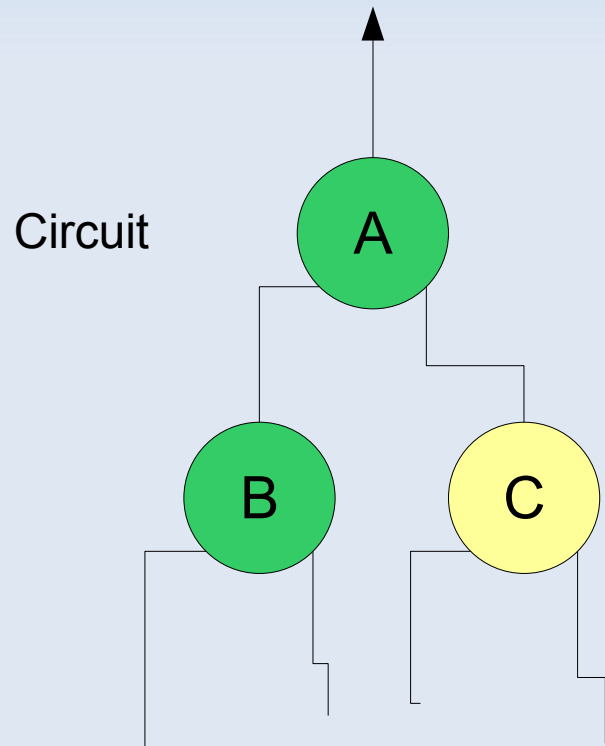
# Generating Cuts Across Domains

- BDD manager is not aware of domains, how do I avoid aliasing issues??
  - e.g. In this example, index 2 represents 2 different nodes



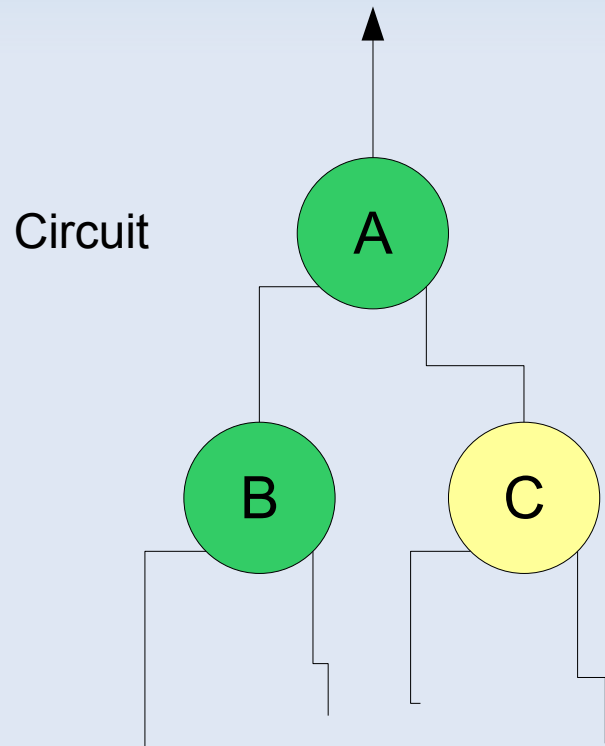
# Generating Cuts Across Domains

- I create a copy of the bdd cut set of the cross domain fanin;
  - **HOWEVER**, all the bdd indeces have been shifted to a set of unused variables in the current domain



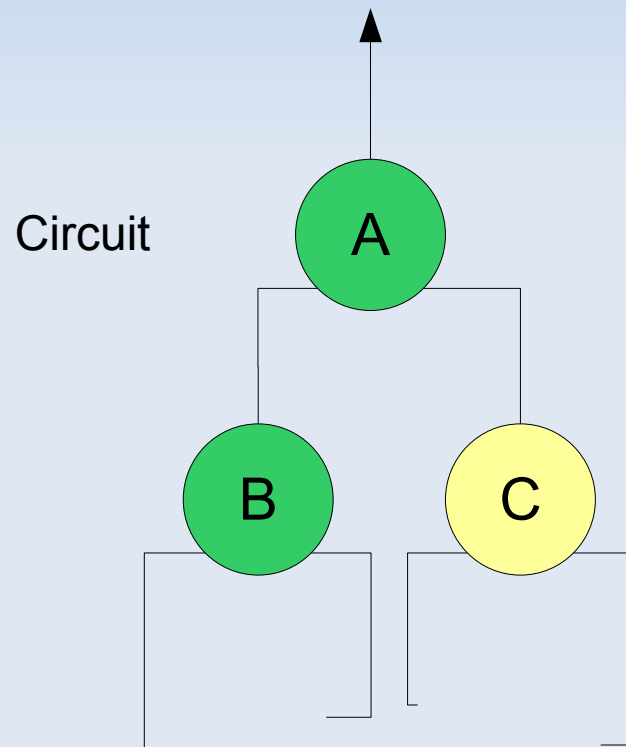
# Generating Cuts Across Domains

- After I am done with the shifted copy, I recycle it (free up the memory)
  - However, I still need to keep a mapping from the shifted indices to the cross domain circuit nodes



# Example: Generating Cuts across domain

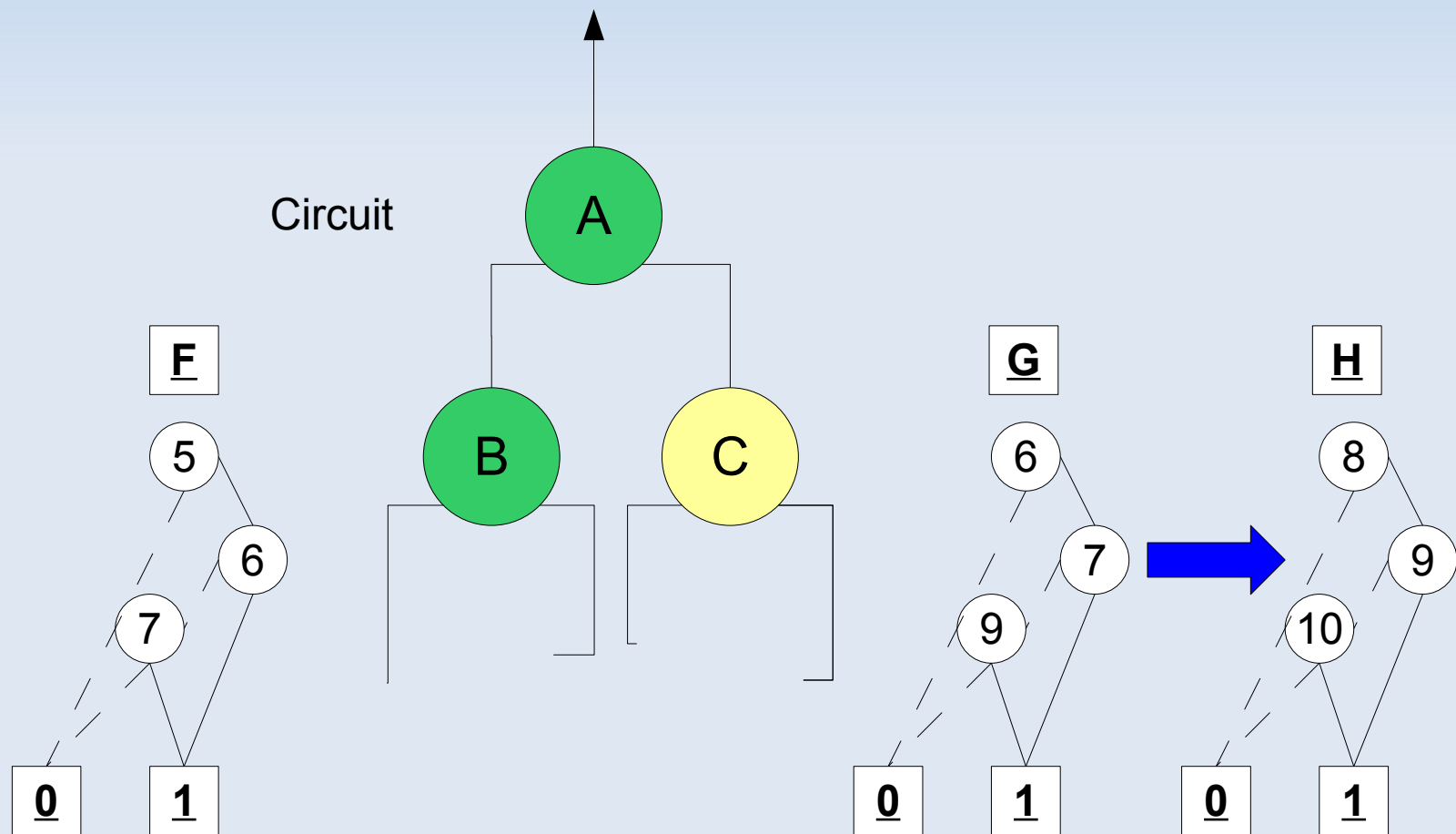
- Generate cuts for node **A**





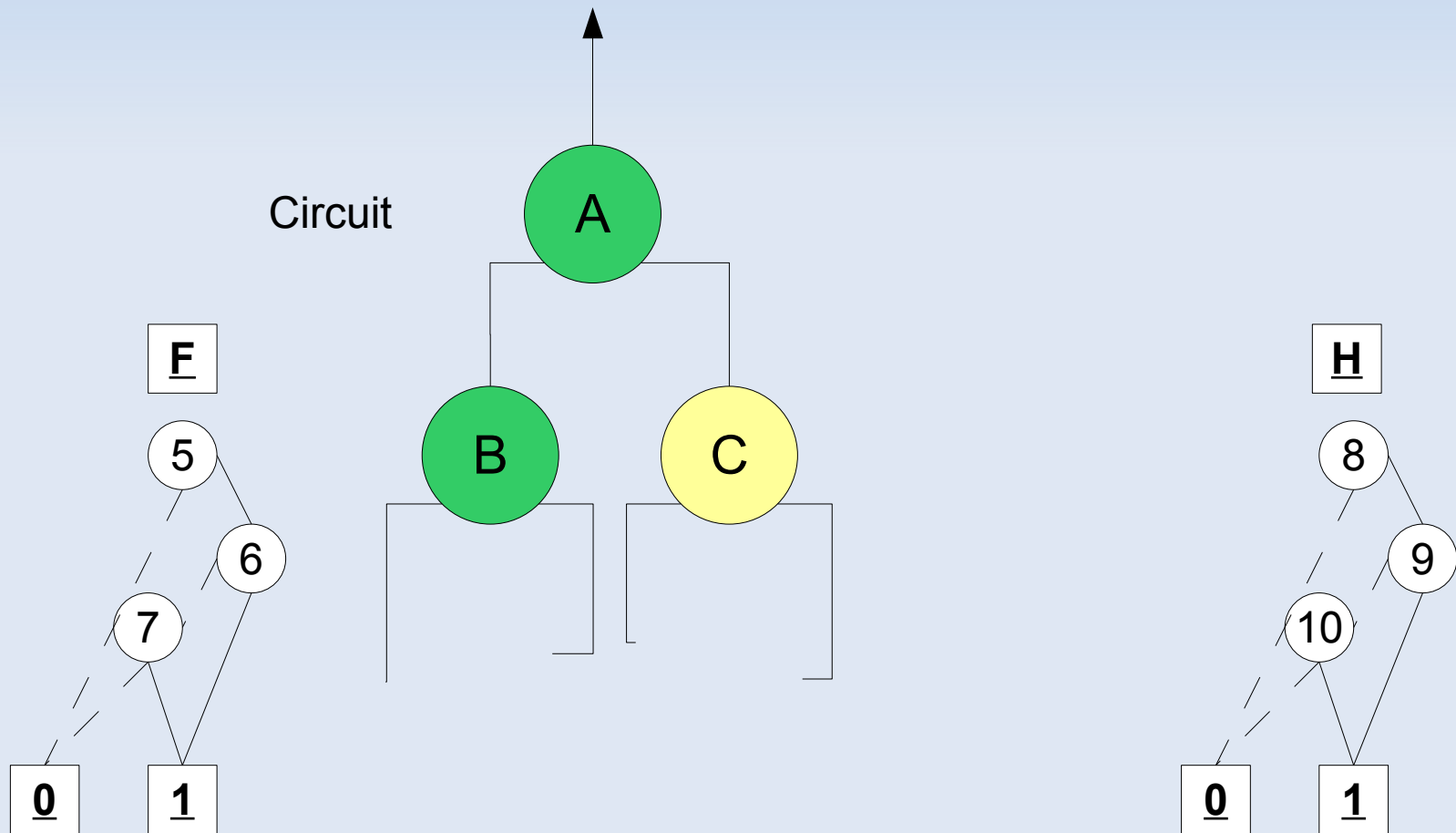
# Example: Generating Cuts across domain

- Generate cuts for node **A**



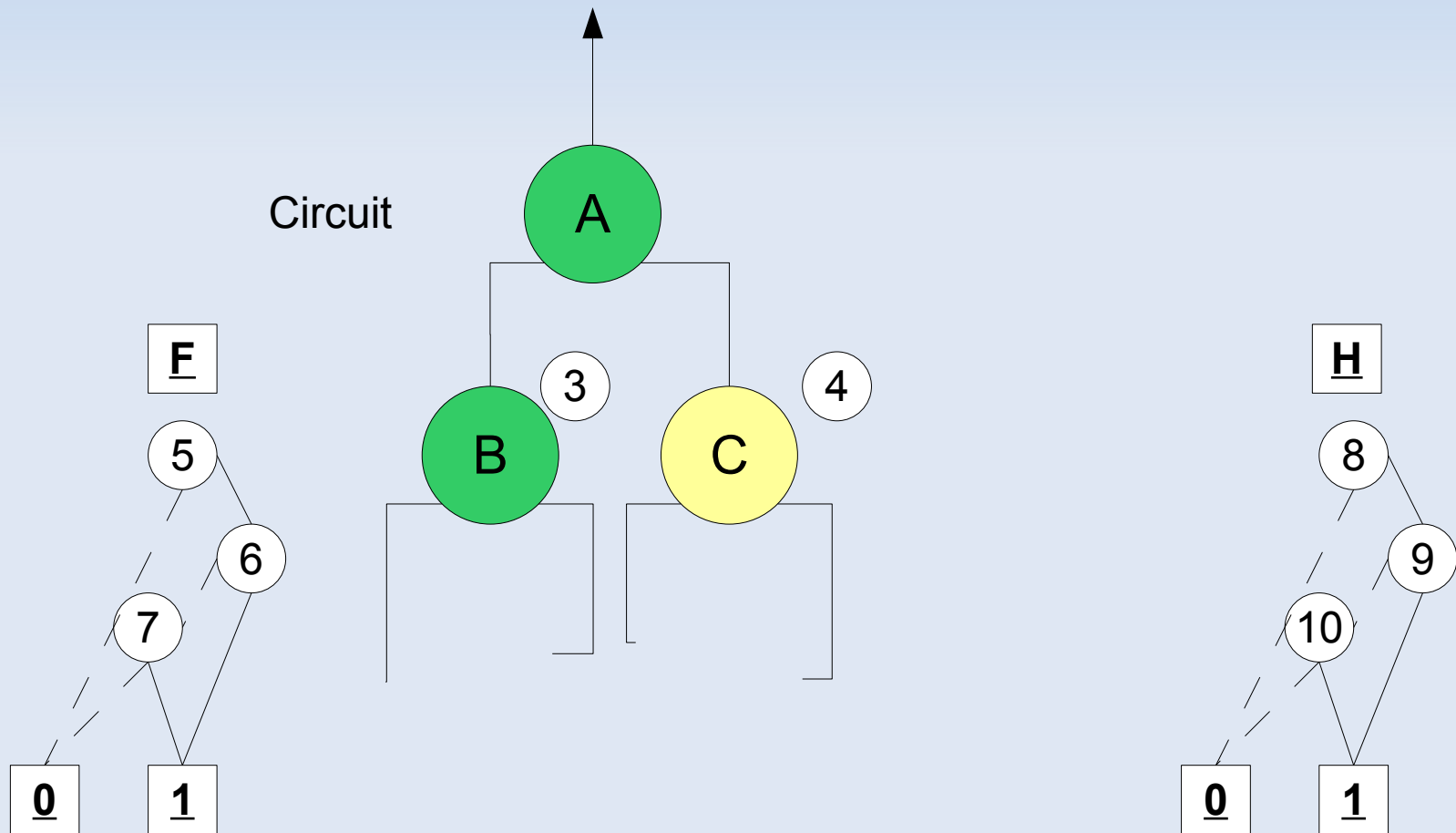
# Example: Generating Cuts across domain

- Generate cuts for node **A**



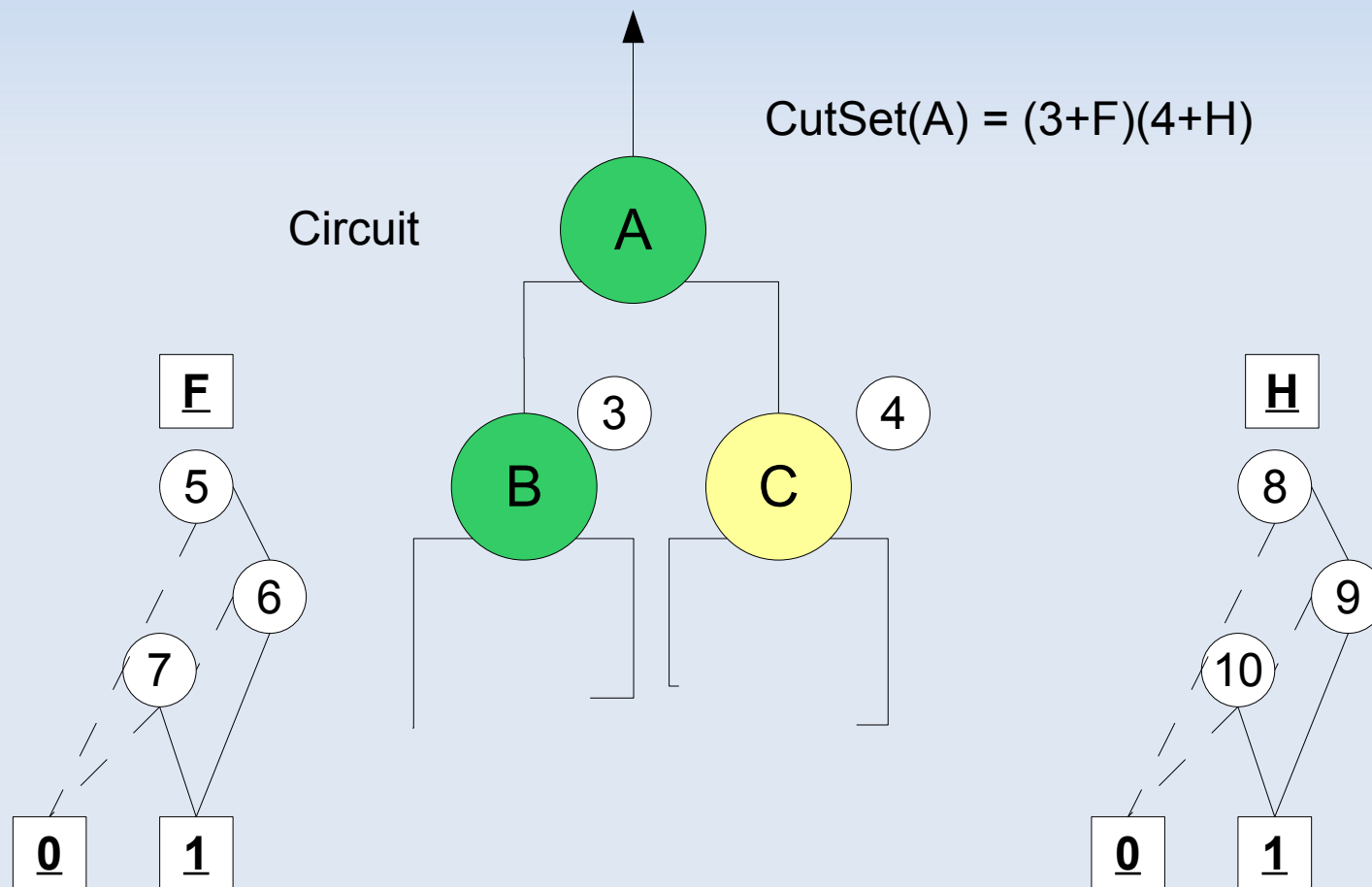
# Example: Generating Cuts across domain

- Generate cuts for node **A**



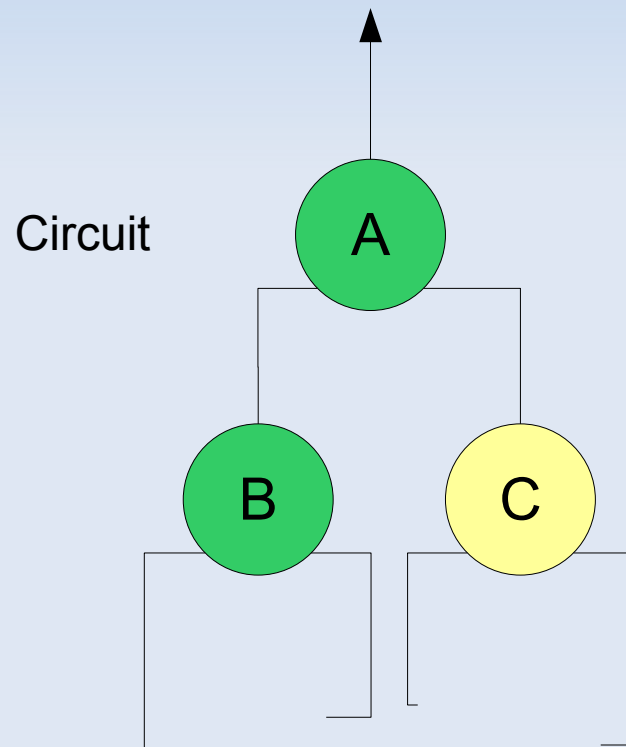
# Example: Generating Cuts across domain

- Generate cuts for node **A**

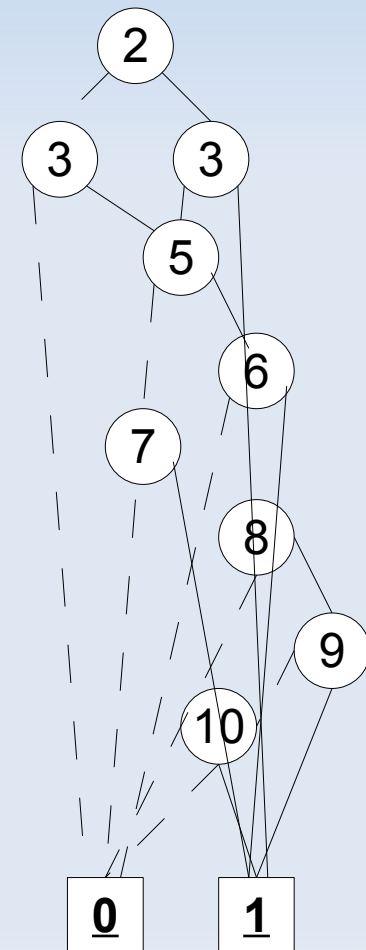


# Example: Generating Cuts across domain

- Generate cuts for node **A**



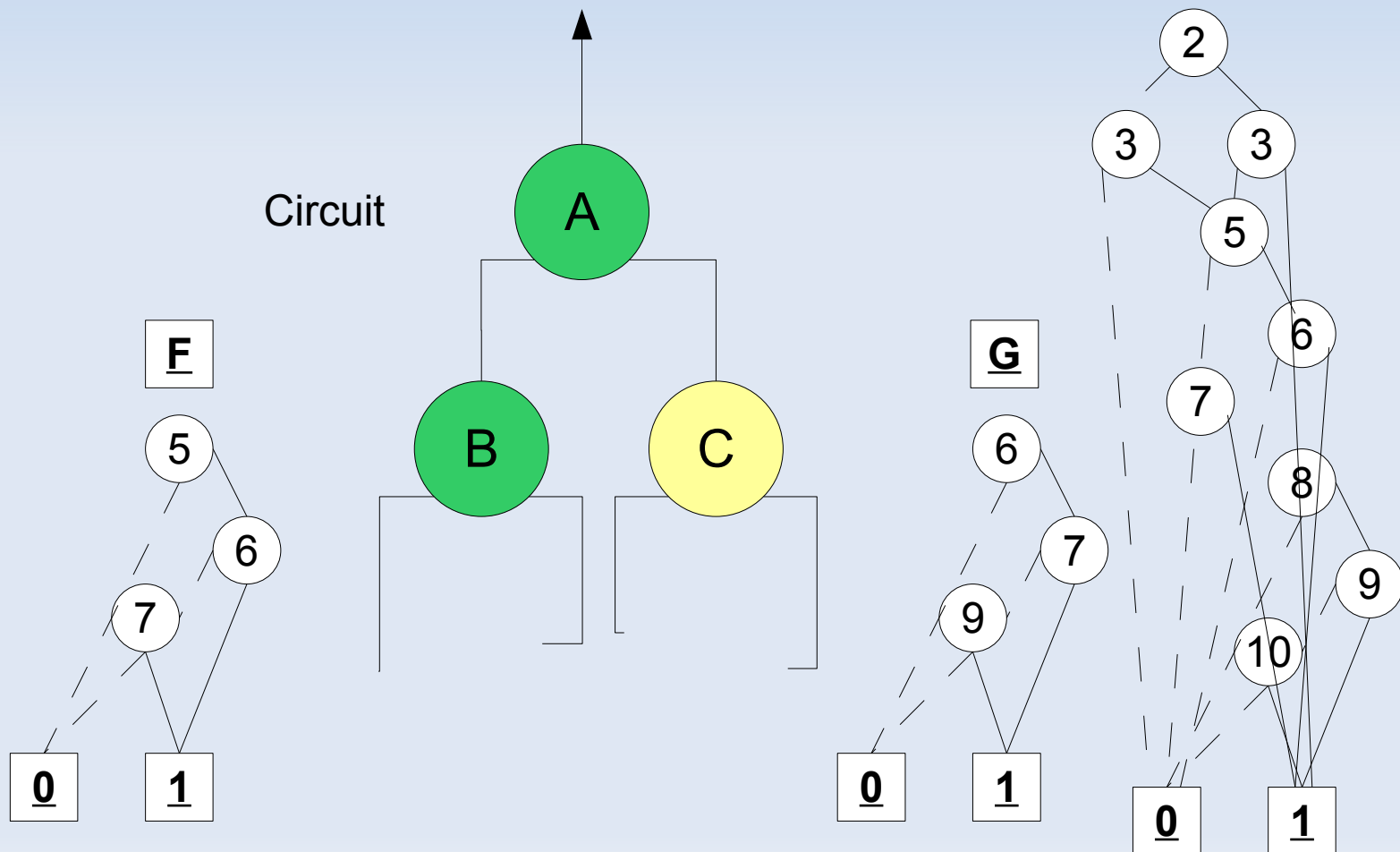
$$\text{CutSet}(A) = (3+F)(4+H)$$



# Example: Generating Cuts across domain

- Generate cuts for node **A**

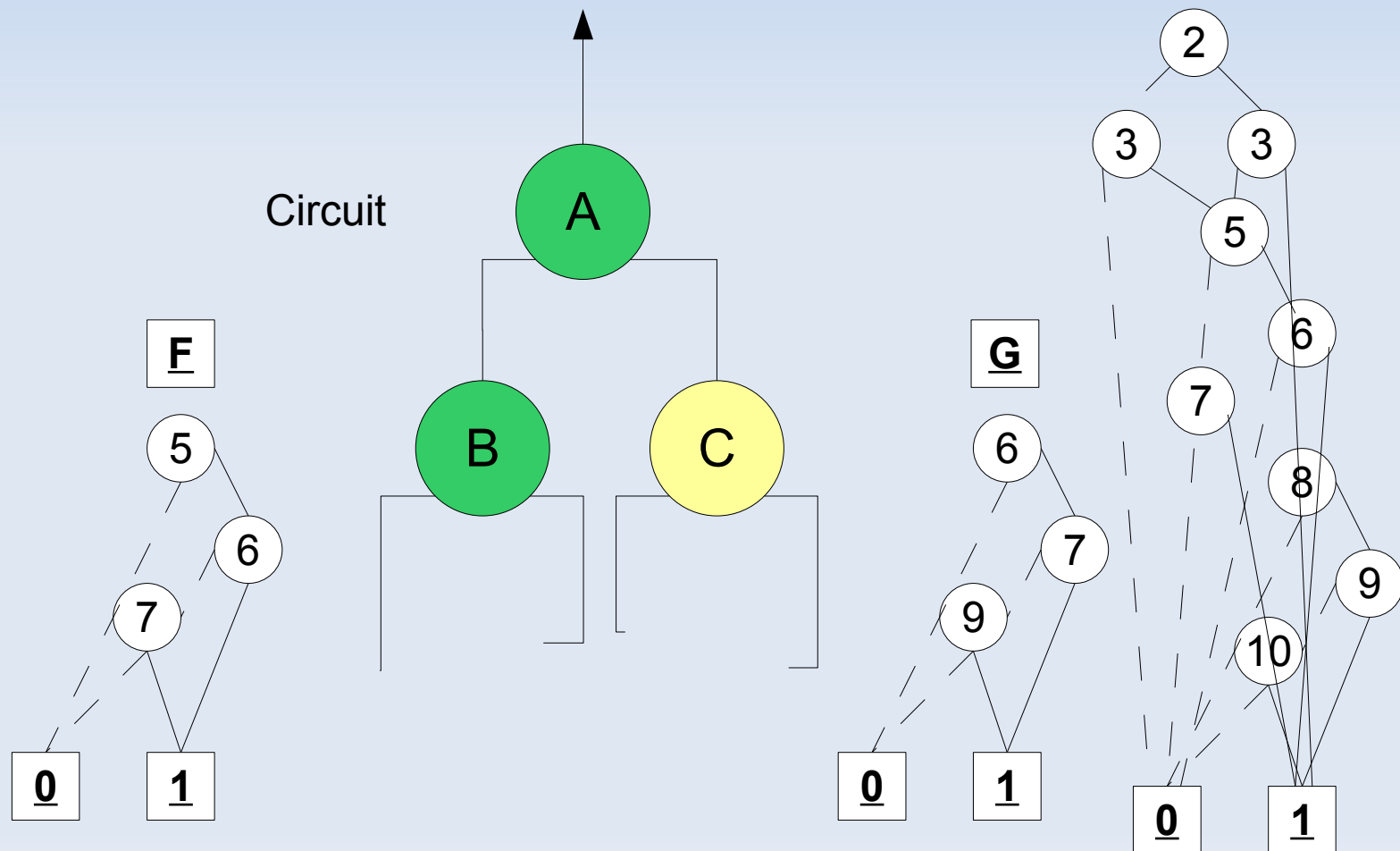
$$\text{CutSet}(A) = (3+F)(4+H)$$



# Example: Generating Cuts across domain

- Each domain has a hash table, of their BDD indices to the circuit node

$$\text{CutSet}(A) = (3+F)(4+H)$$



# Pseudo Code

```
GenerateBddCuts (cct)
```

```
    DomainSize=min (size (cct) , MaxBddIndex)
```

```
    while (needCuts)
```

```
        domain=GenerateAndMarkDomain (DomainSize)
```

```
        GenerateCuts (domain)
```

# Pseudo Code

```
GenerateDomain (node, domainSize)
  if (size(domain) >= domainSize)
    return domain
  foreachFanin (node)
    if (domain(fanin) != domain)
      addCutSetNodesSubDomain (fanin, domain)
    else
      domain=GenerateDomain (fanin, domainSize)
  AddNode (node, domain)
  return domain
```

# Pseudo Code

```
addCutSetNodesSubDomain (node, subDomain)
  bddCutSet = cutSet (node)
  domain = getDomain (node)
  support = GetSupport (bddCutSet)
  foreachSupportVariable (support)
    node = getCctNode (bddIndex, domain)
    subDomain (node) = subDomain
```

# Pseudo Code

```
GenerateCuts (node)
```

```
  CutSet = 1;
```

```
  forEachFanin (node)
```

```
    if (domain (fanin) != domain (node) )
```

```
      shiftBddCutSet (fanin, domain)
```

```
    else
```

```
      CutSet &= cutSet (fanin) + bddIndex (fanin)
```

```
  cutSet (node) = cutSet
```

```
  AssignBddIndex (node)
```

```
  AddNodeMap (bddIndex (node) , domain) = node
```

# Pseudo Code

```
shiftBddCutSet (node, subDomain)  
  updateBddIndex (node, subDomain)  
  bddCutSet = shiftCutSet (cutSet (node), domain (node))
```

# Code

Domain:

```
exNewJushortField(  
    ec, "dryColour", 0, &cutter->ecNode->dryColour  
);
```

SubDomain:

```
exNewJushortField(  
    ec, "wetColour", 0, &cutter->ecNode->wetColour  
);
```

# Create Domain and Generate Cuts

```
// Generate Domain
```

```
coloured = setColour(  
    cutter,node,curColour, &count  
);
```

```
// Generate Cuts in Domain
```

```
if ( !coloured ) {  
    Jint i;  
    for (i=0;i<vecGetSize(colourPart);i++) {  
        GridNodeRef colourNode = comCastP2I(  
            vecGetElement(colourPart,i)  
        );  
        createBddCuts(  
            db, cutter, cutter->part, colourNode, curColour, &bddIndex  
        );  
    }  
}
```

# Generate Cuts

```
createBddCuts(  
    GridDB db, GridAlgoCut cutter, GridPartRef part,  
    GridNodeRef node, Jushort curColour, Jushort *curBddIndex  
){  
    DdManager *dd = cutter->bddManager;  
    IGeneric key = 0;  
    DdNode *bddCuts=NULL,*bddVar=NULL;  
    GridPartKind kind;  
  
    assert( gridNode(node,flag).pin != KIND_PIN_INOUT );  
    if (gridNode(node,flag).pin == KIND_PIN_OUTPUT) {  
        return;  
    }  
    kind = gridPart( part, kind );  
    if ( !nodesCI(cutter,node) ) {  
        bddCuts = getBddCut (  
            db, cutter, dd, node, curColour, curBddIndex  
        );  
        assert(bddCuts)  
        gridCutNode(node,bddCuts) = bddCuts;  
    }  
}
```

# Generate Bdd Cuts

```
static DdNode* getBddCut( GridDB db, GridAlgoCut cutter, DdManager*dd, GridNodeRef node, Jushort curColour, Jushort
    *bddIndex ) {
    Jint i;
    DdNode* bddCuts, *orTmp;

    bddCuts = Cudd_ReadOne(dd);
    Cudd_Ref(bddCuts);
    for (i=0;i < gridNode(node,ndins);i++) {
        DdNode* bddCutSet, *bddVar, *newCutSet;
        GridEdgeRef edge = gridNode(node,dins)+i;
        GridNodeRef fanin = gridGetParentNode(db, gridEdge(edge,src));

        if ( gridCutNode(fanin,dryColour) != curColour ) {
            // copy over and shift...
            updateBddCuts(
                cutter, dd, fanin, curColour, bddIndex
            );
        }
        bddVar = Cudd_bddIthVar( dd, gridCutNode(fanin,bddVarIndex) );
        bddCutSet = gridCutNode(fanin, bddCuts);

        if ( nodelsCI( cutter, fanin ) ) {
            orTmp = bddVar;
        }
        else {
            orTmp = Cudd_bddOr( dd, bddVar, bddCutSet );
        }
        newCutSet = Cudd_bddAndPrune(
            dd, orTmp, bddCuts, cutter->nMaxCutSiz
        )
        bddCuts = newCutSet;
    }

    return bddCuts;
}
```