

# Input file for ASCII code

(Graphical interface and  
correctness checkers )

---

*Lidia Onica*



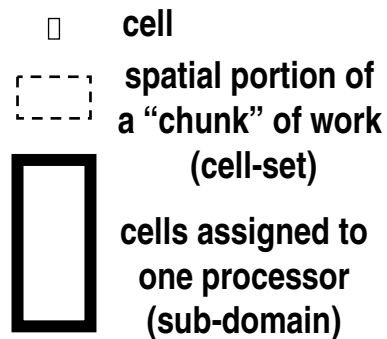
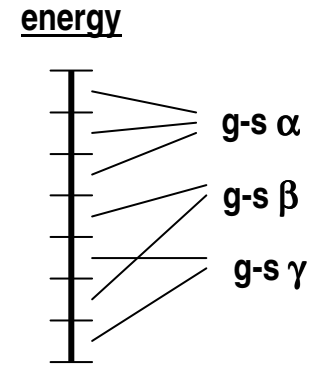
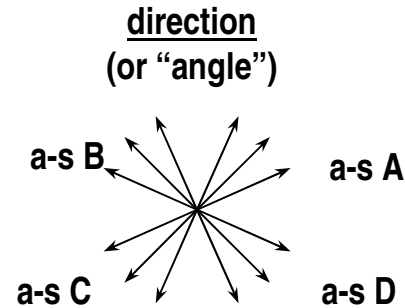
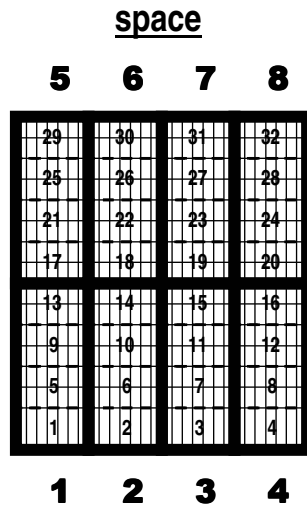
# Input data

---

- data regarding the problem : boundary conditions,  
material properties
- data regarding the way the problem is solved :  
domain discretization,  
partitioning in subdomains ,  
aggregation of cellsets, angleset, groupsets,  
quadrature type, iterative schemes.

( the work “chunk” = groupset x angleset x cellset )

# An example



**Example problem:**  
 2 spatial dimensions, 8 processors,  
 8x4 = 32 cell-sets, 8x4x14=448 cells,  
 12 angles, 4 angle-sets,  
 7 energy groups, 3 group-sets, same  
 directions and angle-sets for each  
 group-set.

# Input format

---

- The input data is stored in a XML file, and we use XPATH to specify nodes in the file.
- XML is a language to store structured data.
- XML documents may be viewed as tree with the tags forming the nodes.
- XPATH is a standard notation used to reach the nodes of XML documents .
- For example the XPATH expression `/a/b/c` points to all the nodes "c" of the tree, where the tree is rooted at "a" and has "b" as one of its children which in turn has node "c" as its child.

# The tree structure of input.xml



```
<problem>
  <groupsets>
    <energy_set>
      <anglesets> ...</anglesets>
      <quad_info> ...</quad_info>
    </energy_set>
    .....
  </groupsets>

  <dimension>... </dimension>
  <material_def>...</material_def>

</problem>
```

# anglesets aggregation for our example

---

```
<anglesets>
  <set>
    <ID>0</ID>
    <set_include>0</set_include>
    <set_include>1</set_include>
    <set_include>2</set_include>
  </set>
  <set>
    <ID>1</ID>
    <set_include>3</set_include>
    <set_include>4</set_include>
    <set_include>5</set_include>
  </set>
  .....
</anglesets>
```

# Examples of XPATH expressions

---

`/problem/groupsets/energy_sets`

each *energy\_set* tag represents a energy group ( 7 in our example)  
the XPATH expression will refer to all *energy\_set* nodes

**`/problem/groupsets/energy_sets/anglesets/ set`**

refers to all anglesets

there are 4 anglesets and 12 angles, so we get  
3 angles per set ( though they don't have to be evenly distributed )

# Example of Java library methods to manipulate the nodes in the tree



```
static Document doc = xml_tree.getDoc();
```

```
NodeList list = doc.getElementsByTagName("ngroups.int");
```

```
Node node = list.item(0);
```

```
node.appendChild( doc.createTextNode("1") );
```

```
NodeList nodelist = XPathAPI.selectNodeList (doc, xpath);
```

# GUI demo



# Correctness conditions



- A first example:

domain discretization

- in each dimension the domain can be divided to correspond to different material type
- the divisions are disjoint, but their union is connected.

a problem : divisions overlapping

# Correctness conditions, examples

---

- The energy\_group group\_id's should go from 1 to ngroups.int .
- Groupsets: every number between 0 and ngroups-1 must be in set\_include for some energy set.
- For every energy set, all numbers between 0 and  $(n*(n+ 2)) - 1$  appears in one set/set\_include,  $n$  is quadrature order .

These will fit into a “checking range” category.

# Example of correctness conditions

---

- The number of energy groups must be equal to `ngroups.int`
- Number of energy sets must be less than `ngroups.int`
- For every isotope, one `isotope_def.group` per energy group.

a “Count” category

# Rules in terms of XPATH expressions

## Generic rules (abstract rules)



Some of the checks we wanted to apply to the input file were of a common type, more specific, the same check applied to different nodes.

Thus we designed a few general rules that apply to sets of nodes( specified with XPATH expressions).

By specifying the XPATH expressions we obtain a concrete rule, which we store in a “database” file rules.txt.

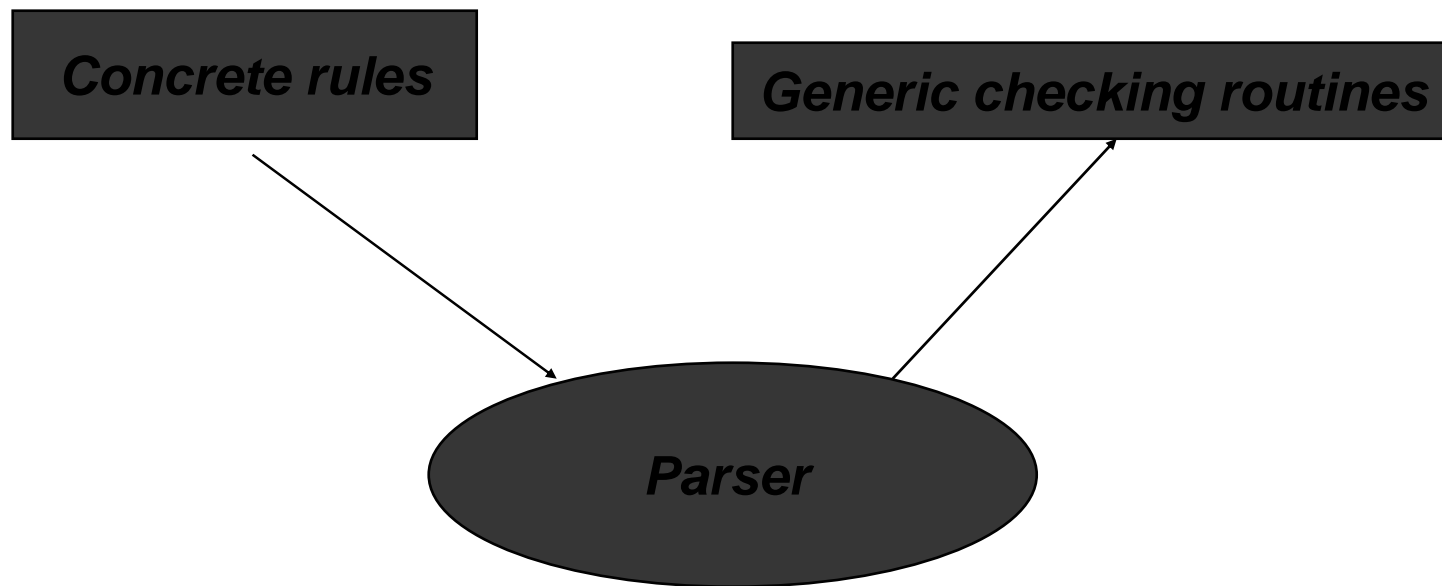
The checking process :

A rule is parsed and the generic rule it instantiates is found .

Next the checker routine corresponding to the general rule is called with arguments from the concrete rule, thus an assertion for the particular set of nodes is verified.

# the checking diagram

The parser recognizes the type of a concrete rule



and calls a method to verify if the rule is satisfied

## Generic rule: example

---

Xpath\_expr = RANGE (type, start, end)

where    type: data type of the range  
         start, end: the initial and final values of the range

Use: The set of values in LHS should be in the range specified in RHS

# Concrete rules



```
/prototype/energy_groups/energy_group/group_id  
= RANGE (int, 1, /prototype/common/ngroups.int)
```

```
/prototype/groupsets/energy_set/set_include  
= RANGE (int, 0, /prototype/common/ngroups.int - 1)
```

```
/prototype/groupsets/energy_set/set/set_include  
= RANGE (int, 0, (/prototype/groupsets/energy_sets/quad_info/quad_order.int *  
(/prototype/groupsets/energy_sets/quad_info/quad_order.int + 2))-1)
```

# More generic rules



- COUNT (Xpath\_expr) = Xpath\_expr
- COUNT (Xpath\_expr)= COUNT (Xpath\_expr)  
Use: for comparisons based on tag count
- UNIQUE(Xpath\_expr)  
Use: To check if the set of values returned by the expression is unique
- Function (Xpath\_expr)  
where Function = MAX, MIN, SUM  
Use: Returns the max, min, sum of the set of values

## Concrete rules , instances of “Count” generic rule

---

```
COUNT ($/prototype/energy_groups/energy_group$) =  
        $/prototype/common/ngroups.int$
```

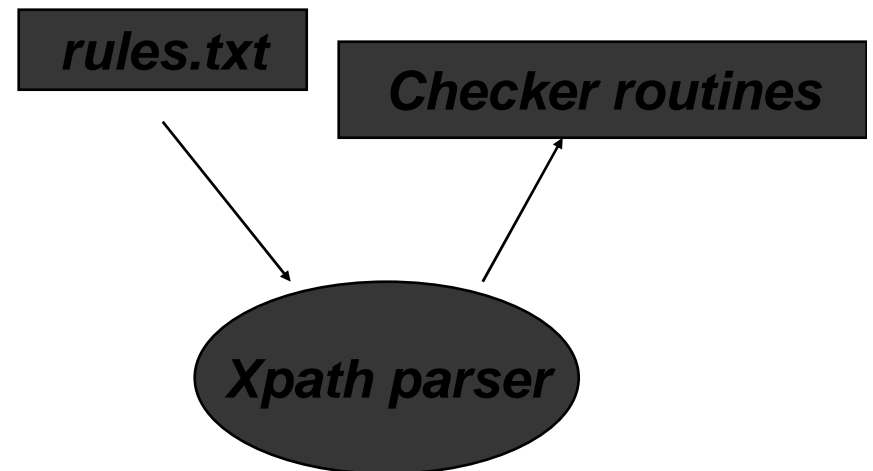
```
COUNT($/prototype/groupsets/energy_set$)<=$/prototype/common/  
        ngroups.int $
```

```
COUNT ($/prototype/isotope_def$[i]$/isotope_def.group$) =  
        COUNT ($/prototype/energy_groups/energy_group$)
```

# instantiating a parser

```
new XPathParser("rules.txt");
```

```
class XPathParser {  
.....  
  ch = new checker Routines("input.xml");  
.....  
}
```



# Questions

# Unique, Sum

---

- UNIQUE (\$/prototype/isotope\_def\$[i]\$/isotope\_def.clide.int\$)
- SUM(\$/prototype/material\_def\$[i]\$/material\_def.isotop/material\_def.isotop.intensity.fp\$) = 1.0
- UNIQUE (\$/prototype/dimension\$[i]\$/dim.division/dim.division.start.fp\$)
- UNIQUE (\$/prototype/dimension\$[i]\$/dim.division/dim.division.end.fp\$ )

# Conditional rules



- Generic rules  
IF\_DEFINED(xpath\_expr) THEN (rule )  
IF (rule) THEN ( rule)
- IF\_DEFINED(\$/prototype/common/TSA\_info/TSA\_levels\$) THEN  
IF (\$/prototype/common/TSA\_info/TSA\_levels\$ > 0) THEN  
COUNT(\$/prototype/common/TSA\_level\$)=\$/prototype/common/TSA\_info/TSA\_levels\$
- IF\_DEFINED(\$/prototype/common/TSA\_info/TSA\_max.int\$) THEN  
\$/prototype/common/TSA\_info/TSA\_min.int\$=  
RANGE (int, 1,\$/prototype/common/TSA\_info/TSA\_max.int\$ )

# Example of checker implementation

---

method to check if a tag is defined in the input file:

```
public static boolean checkDefined (String xpath)
{
    NodeList nodelist=
        org.apache.xpath.XPathAPI.selectNodeList(doc, xpath);

    if(nodelist.getLength() >0)
        return true;
    else
        return false;
}
```

# Example of checker implementation



Checker routine for the rule:  $\text{COUNT}(\text{XPath}) = \text{COUNT}(\text{XPath})$

```
public static int checkTypeCount (String xpathLHS, String xpathRHS, String op) {  
  
    NodeList nodelistLHS = org.apache.xpath.XPathAPI.selectNodeList (doc, xpathLHS);  
    NodeList nodelistRHS = org.apache.xpath.XPathAPI.selectNodeList (doc, xpathRHS);  
    int numNodesLHS = nodelistLHS.getLength();  
    int numNodesRHS = nodelistRHS.getLength();  
  
        switch (help.getOpType(op))  
        {  
        case 1: if (nodelistRHS.getLength() == nodelistLHS.getLength())  
                return true;  
                else    return false;  
        .....}  
}
```

# parser code : parsing a Count rule

---

```
f = <COUNT> { xpathVector = new Vector() ; }  
  
  ( <LEFT_BRKT> xpath_for() <RIGHT_BRKT>  
    {  
      rel_op = <REL_OPERATOR>  
      (  
        <COUNT> <LEFT_BRKT>  
          xpath_expr2=<XPATH> <RIGHT_BRKT>  
          {  
            checkVal = ch.checkCount(xpathVector,  
                                   xpath_expr2.image, rel_op.image);  
            return checkVal;  
          }  
      )  
    }  
  )
```

# problem/common node



```
<common>
  <geometry>XYZ</geometry>

  <ngroups.int>1</ngroups.int>

  <dimensions.int>3</dimensions.int>

  .....
  <alpha.fp>0</alpha.fp>
  <beta.fp>0</beta.fp>
  <gamma.fp>0</gamma.fp>

  <aggregation_type>PLANE_BASED</aggregation_type>

  <partition_type>OTHER</partition_type>

  <aggregation_factor_x>4</aggregation_factor_x>
  <aggregation_factor_y>4</aggregation_factor_y>
  <aggregation_factor_z>2</aggregation_factor_z>

  <partition_params>
    <partition_x>1</partition_x>
    <partition_y>1</partition_y>
    <partition_z>1</partition_z>
  </partition_params>
</common>
```



# groupsets

```
<groupsets>
  <energy_set>
    <ID>0</ID>
    <set_include>0</set_include>
    <anglesets>
      <set>
        <ID>0</ID>
        <set_include>0</set_include>
      </set>
      <set_include>0</set_include>
      </set>
      .....
      <set>
        <ID>7</ID>
        <set_include>14</set_include>
        <set_include>15</set_include>
      </set>
    </anglesets>
  </energy_set>
  <quad_info>
    <quad_order.int>2</quad_order.int>
    <quad_norm.fp>12.566371</quad_norm.fp>
    <quad_type.st>LevelSym</quad_type.st>
  </quad_info>
</groupsets>
```



# dimension

```
<dimension>
  <dimension.id>1</dimension.id>
  <dim.division>
    <dim.division.cells.int>4</dim.division.cells.int>
    <dim.division.start.fp>0.0</dim.division.start.fp>
    <dim.division.end.fp>3.0</dim.division.end.fp>
    <dim.division.id>1.1</dim.division.id>
  </dim.division>
</dimension>
<dimension>
  <dimension.id>2</dimension.id>
  <dim.division>
    <dim.division.cells.int>4</dim.division.cells.int>
    <dim.division.start.fp>0.0</dim.division.start.fp>
    <dim.division.end.fp>5.0</dim.division.end.fp>
    <dim.division.id>2.1</dim.division.id>
  </dim.division>
</dimension>
<dimension>
  <dimension.id>3</dimension.id>
  <dim.division>
    <dim.division.cells.int>4</dim.division.cells.int>
    <dim.division.start.fp>0.0</dim.division.start.fp>
    <dim.division.end.fp>7.0</dim.division.end.fp>
    <dim.division.id>3.1</dim.division.id>
  </dim.division>
</dimension>
```

# Materials Section

---

```
<isotope_def>
```

```
  <isotope_def.clide.int>1</isotope_def.clide.int>  
  <isotope_def.scatt_order.int>0</isotope_def.scatt_order.int>
```

```
  <isotope_def.group>  
    <isotope_def.sigtot.fp>1.0</isotope_def.sigtot.fp>  
    <isotope_def.nusigf.fp>0.0</isotope_def.nusigf.fp>  
    <isotope_def.chi.fp>0.0</isotope_def.chi.fp>  
    <isotope_def.sigs>  
      <isotope_def.sigs.fp>0.0</isotope_def.sigs.fp>  
    </isotope_def.sigs>  
  </isotope_def.group>
```

```
</isotope_def>
```

# Materials Definitions

- 

```
<material_def>  
  <material_def.name>material1</material_def.name>  
  <material_def.isotop>  
    <material_def.isotop.clide.int>1</material_def.isotop.clide.int>  
    <material_def.isotop.intensity.fp>1.0</material_def.isotop.intensity.fp>  
  </material_def.isotop>  
</material_def>
```

- 

```
<regions>  
  <regions-material_region>  
    <material_reg.material.str>material1</material_reg.material.str>  
    <material_reg.dim_bounds>  
      <material_reg.dim_bounds.dim.int>1</material_reg.dim_bounds.dim.int>  
      .....  
    </material_reg.dim_bounds>  
    <material_reg.dim_bounds.dim.int>3</material_reg.dim_bounds.dim.int>  
  </material_reg.dim_bounds>  
</regions-material_region>  
</regions>
```



# boundary\_info

```
<boundary_info>  
  <left_bound>  
    <bound_type>VACUUM</bound_type>  
  </left_bound>  
  <right_bound>  
    <bound_type>VACUUM</bound_type>  
  </right_bound>  
  <front_bound>  
    <bound_type>VACUUM</bound_type>  
  </front_bound>  
  <back_bound>  
    <bound_type>VACUUM</bound_type>  
  </back_bound>  
  <top_bound>  
    <bound_type>VACUUM</bound_type>  
  </top_bound>  
  <bottom_bound>  
    <bound_type>VACUUM</bound_type>  
  </bottom_bound>  
</boundary_info>
```

# sources



```
<named_sources>
```

```
  <source_def>
```

```
    <source_def_name>Test_Source</source_def_name>
```

```
    <intensity>1.0</intensity>
```

```
  </source_def>
```

```
</named_sources>
```

```
<source_geometry>
```

```
  <source_region>
```

```
    <source_name>Test_Source</source_name>
```

```
    <source_dim_bounds>
```

```
      <source_dim_bounds_dim>0</source_dim_bounds_dim>
```

```
      <source_dim_start>1</source_dim_start>
```

```
      <source_dim_end>1</source_dim_end>
```

```
    </source_dim_bounds>
```

```
  </source_region>
```

```
</source_geometry>
```



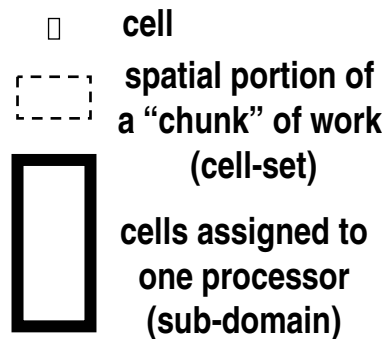
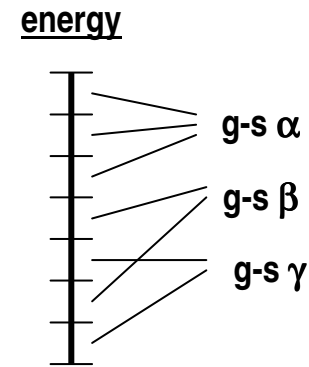
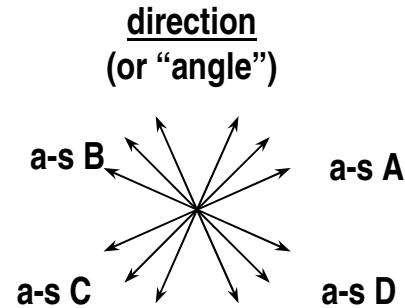
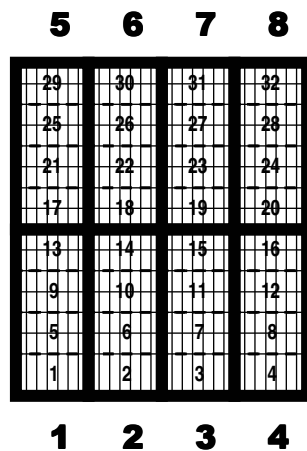
# AngleSets

<anglesets>

```
<set>
  <ID>0</ID>
  <set_include>0</set_include>
  <set_include>1</set_include>
  <set_include>2</set_include>
</set>
<set>
  <ID>1</ID>
  <set_include>3</set_include>
  <set_include>4</set_include>
  <set_include>5</set_include>
</set>
<set>
  <ID>1</ID>
  <set_include>6</set_include>
  <set_include>7</set_include>
  <set_include>8</set_include>
  <set_include>9</set_include>
  <set_include>10</set_include>
  <set_include>11</set_include>
</set>
.....
<set>
  <ID>7</ID>
  <set_include>42</set_include>
  <set_include>43</set_include>
  <set_include>44</set_include>
  <set_include>45</set_include>
  <set_include>46</set_include>
  <set_include>47</set_include>
</set>
```

</anglesets>

# An example



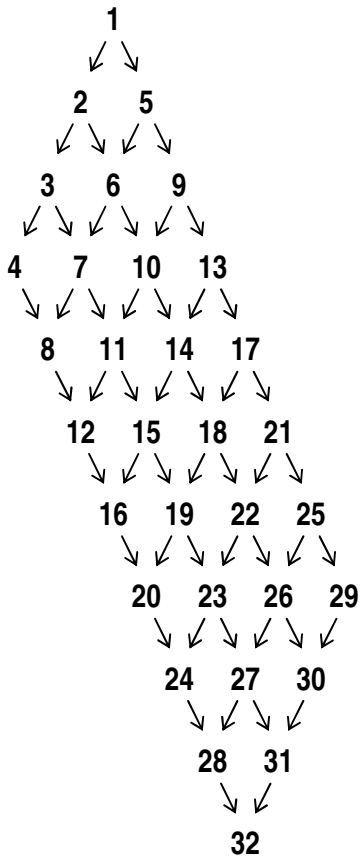
**Example problem:**

2 spatial dimensions, 8 processors,  
 $8 \times 4 = 32$  cell-sets,  $8 \times 4 \times 14 = 448$  cells,  
 12 angles, 4 angle-sets,  
 7 energy groups, 3 group-sets, same  
 directions and angle-sets for each  
 group-set.

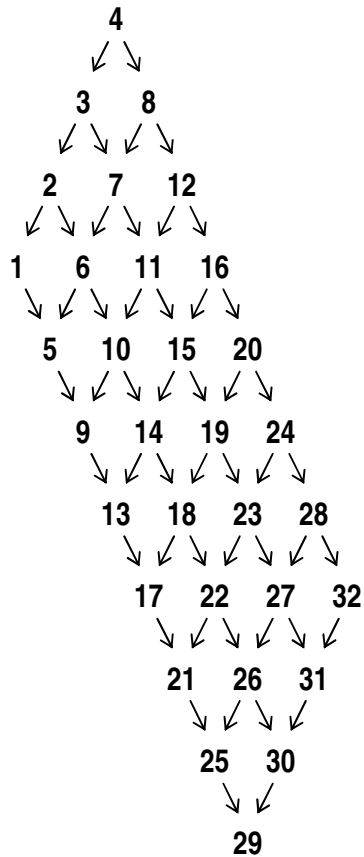
# DAG s for each groupset.

Parasol

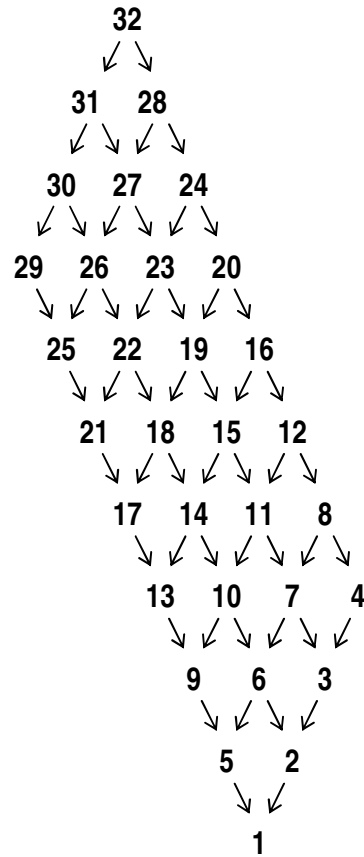
angle-set A



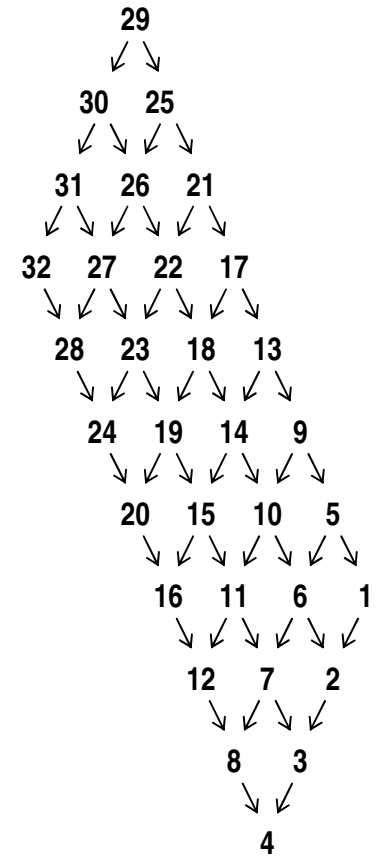
angle-set B



angle-set C



angle-set D



- Numbers are cellset indices
- Colors indicate processors