Supplementary Material for RAVEN: A Dataset for Relational and Analogical Visual rEasoNing

Chi Zhang*,1,2 Feng Gao*,1,2 Baoxiong Jia[1] Yixin Zhu[1,2] Song-Chun Zhu[1,2]
[1] UCLA Center for Vision, Cognition, Learning and Autonomy
[2] International Center for AI and Robot Autonomy (CARA)
{chi.zhang,f.gao,baoxiongjia,yixin.zhu}@ucla.edu, sczhu@stat.ucla.edu

1. Grammar Productions

We use an Attributed Stochastic Image Grammar (A-SIG) to represent the structured image space. The grammar is decomposed into 5 levels, each of which could have possibly multiple symbols. All production rules are summarized in Table 1.

Table 1. Grammar production rules used in RAVEN. Note that the production rules in each level produce symbols in the next level. Here, $\cdot$ denotes the `and` relation in the grammar, $\emptyset$ denotes the null symbol, and "Entity" is an instantiated object on the canvas. We use * to denote the shared production rules for every symbol in a given level.

Level	Production Rules

`Scene`	`Scene` $\rightarrow$ `Singleton`
	`Scene` $\rightarrow$ `Left-Right`
	`Scene` $\rightarrow$ `Up-Down`
	`Scene` $\rightarrow$ `Out-In`

`Structure`	`Singleton` $\rightarrow$ `Grid`
	`Left-Right` $\rightarrow$ `Left` $\cdot$ `Right`
	`Up-Down` $\rightarrow$ `Up` $\cdot$ `Down`
	`Out-In` $\rightarrow$ `Out` $\cdot$ `In`

`Component`	`Grid` $\rightarrow$ `Center`
	`Grid` $\rightarrow$ `2×2Grid`
	`Grid` $\rightarrow$ `3×3Grid`
	`Left` $\rightarrow$ `Center`
	`Right` $\rightarrow$ `Center`
	`Up` $\rightarrow$ `Center`
	`Down` $\rightarrow$ `Center`
	`Out` $\rightarrow$ `Center`
	`In` $\rightarrow$ `Center`
	`In` $\rightarrow$ `2×2Grid`

`Layout`	`Layout`* $\rightarrow$ `Entity` $\cdot$ `Layout`* `Layout`* $\rightarrow \emptyset$
`Entity`	`Entity` $\rightarrow$ Entity

Note that excluding `Entity`, we could derive 7 different sentences, *i.e.*, figure configurations, from the grammar. Semantics of symbols are explained below.

- `Scene`
  - `Scene`: This is the starting symbol of the grammar and represents the entire scene. This symbol shares the same name as the level.
- `Structure`
  - `Singleton`: This symbol treats the scene as a whole and does not divide it into independent components.
  - `Left-Right`: The structure divides the scene evenly into a left part and a right part. Two parts are separated by a vertical line.
  - `Up-Down`: The scene is decomposed evenly into an upper part and a lower part. Two parts are separated by a horizontal line.
  - `Out-In`: An outer part and an inner part compose the scene. Two parts do not have any overlaps.
- `Component`
  - `Grid`: The only component in `Singleton`. In this component, all objects are subject to the same set of rules.
  - `Left`: The left part of the `Left-Right`. Objects in this component lie in the left area.
  - `Right`: The right part of the `Left-Right`. Objects in this component lie in the right area.
  - `Up`: The upper part of `Up-Down`. Objects are grouped in the upper area of the figure.
  - `Down`: The lower part of `Up-Down`. Objects are grouped in the lower area of the figure.

* indicates equal contribution.

- `Out`: The outer part of `Out-In`. Objects are clustered in the outer area of the figure.
- `In`: The inner part of `Out-In`. Objects are clustered in the inner area of the figure.

- `Layout`
  - `Center`: There is a single object centered in the area occupied by the component.
  - `2×2Grid`: A maximum number of 4 objects distributed in a 2×2 grid.
  - `3×3Grid`: A maximum number of 9 objects distributed in a 3×3 grid.

- `Entity`
  - `Entity`: A grammatical symbol that would instantiate an object on the canvas.

Another important construct in A-SIG is the attribute. We only have attributes in `Layout` and `Entity`, summarized in Table 2. Note that all the symbols in the same level have the same set of attributes.

Table 2. Attributes in different levels of the grammar.

Level	Attributes
`Layout`	`Number`, `Position`, `Uniformity`
`Entity`	`Type`, `Size`, `Color`, `Orientation`

In Table 2, `Uniformity` and `Orientation` are noise attributes and are not governed by rules. We now explain semantics of each attribute and the values they could take.

- `Layout`
  - `Number`: The number of entities in a given layout. It could take integer values from $[1, 9]$.
  - `Position`: Possible slots for each object in the layout. Each `Entity` could occupy one slot. Different layouts have different slots.
  - `Uniformity`: This attribute could be `True` or `False`. When `False`, objects are not constrained to look alike.
- `Entity`
  - `Type`: Entity types could be triangle, square, pentagon, hexagon, and circle.
  - `Size`: 6 scaling factors uniformly distributed in $[0.4, 0.9]$.
  - `Color`: 10 grey-scale colors for entities, uniformly distributed in $[0, 255]$.
  - `Orientation`: 8 self-rotation angles uniformly distributed in $[0, 2\pi]$.

2. Rules in RAVEN

In total, we have 4 rules that operate on 5 rule-governing attributes: `Constant`, `Progression`, `Arithmetic`, and `Distribute Three`. By introducing internal parameters, we design 8 different rule instantiations.

- `Constant`: Attributes governed by this rule would not change in the row. If it is applied on `Number` or `Position`, attribute values would not change across the three panels. If it is applied on `Entity` level attributes, then we leave "as is" the attribute in each object across the three panels. This design would render every object the same if `Uniformity` is set to `True`; otherwise, it will introduce noise in a problem instance.
- `Progression`: Attribute values monotonically increase or decrease in a row. The increment or decrement could be either 1 or 2, resulting in 4 instances in this rule.
- `Arithmetic`: There are 2 instantiations in this rule, resulting in either a rule of summation or one of subtraction. `Arithmetic` derives the value of the attribute in the third panel from the first 2 panels. For `Position`, this rule is implemented as set arithmetics.
- `Distribute Three`: This rule first samples 3 values of an attribute in a problem instance and permutes the values in different rows.

Among all attributes, we realize that `Number` and `Position` are strongly coupled, hence we do not al-

Table 3. Network architecture used in CNN(+DRT).

Operator	Params
Convolution	3-2-32
BatchNorm	32
ReLU	
Convolution	3-2-32
BatchNorm	32
ReLU	
Convolution	3-2-32
BatchNorm	32
ReLU	
Convolution	3-2-32
BatchNorm	32
ReLU	
(DRT)	512-300
Linear	512
ReLU	
Dropout	0.5
Linear	8

Table 4. Network architecture used in LSTM(+DRT).

Operator	Params
Convolution	3-2-16
BatchNorm	16
ReLU	
Convolution	3-2-16
BatchNorm	16
ReLU	
Convolution	3-2-16
BatchNorm	16
ReLU	
Convolution	3-2-16
BatchNorm	16
ReLU	
(DRT)	256-300
LSTM	96
Linear	8

Table 5. Network architecture used in WReN(+DRT).

Operator	Params
Convolution	3-2-64
BatchNorm	64
ReLU	
Convolution	3-2-64
BatchNorm	64
ReLU	
Convolution	3-2-64
BatchNorm	64
ReLU	
Convolution	3-2-64
BatchNorm	64
ReLU	
Relational Module	512-512-512-256
(DRT)	256-300
Linear	256
ReLU	
Linear	256
ReLU	
Dropout	0.5
Linear	1

low non-`Constant` rules to co-occur on both of the attributes. With 4 rules and 5 attributes, we could have had 20 rule-attribute combinations. However, we exclude `Arithmetic` on `Type`, as it is counterintuitive, resulting in 19 combinations in total.

3. Model Details

In this section, we provide details for all models we use to benchmark RAVEN. We use X-Y-Z to denote the parameters of convolutional layers, where X refers to the kernel size, Y the stride, and Z the number of channels. The proposed Dynamic Residual Tree (DRT) module's parameters are denoted A-B, where A refers to the size of image features and B the length of word embeddings. The relational module is denoted using sizes of the 4 linear layers in it (see the original paper for details). The model architectures are shown in Table 3, 4, 5, and 6.

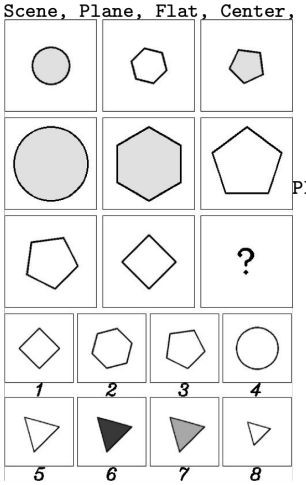
Table 6. Network architecture used in ResNet(+DRT).

Operator	Params
ResNet-18 Backbone	
(DRT)	512-300
Linear	512
ReLU	
Dropout	0.5
Linear	8

4. Examples

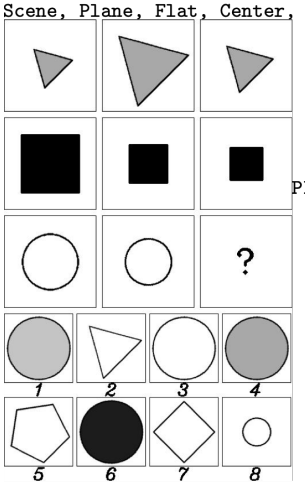
In the following, we provide more examples in RAVEN. Each example contains a problem matrix, an answer set, a serialized tree structure (excluding `Entity`) for the problem, and a rule set. Note that the tree is serialized using a pre-order traversal with / denoting the end-of-branch. As mentioned in Section 2, `Number` and `Position` are tightly coupled. Therefore, when a non-`Constant` rule applies on one attribute and modifies another, we use `NA` to denote the rule of another attribute. Note that image sizes in the answer set are identical to the ones in the problem matrix for model training, validation, and testing.

Scene, Plane, Flat, Center, /, /, /, /



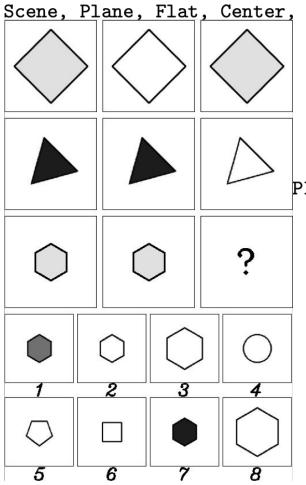
Plane
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Plane, Flat, Center, /, /, /, /



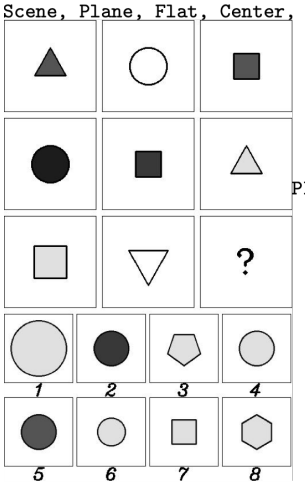
Plane
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Constant:Color]

Scene, Plane, Flat, Center, /, /, /, /



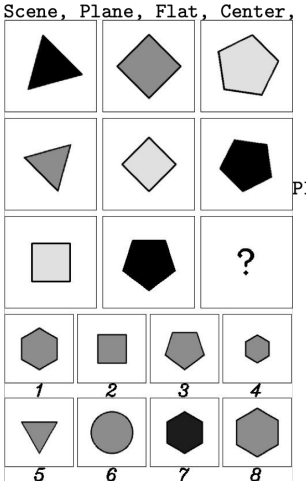
Plane
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Plane, Flat, Center, /, /, /, /



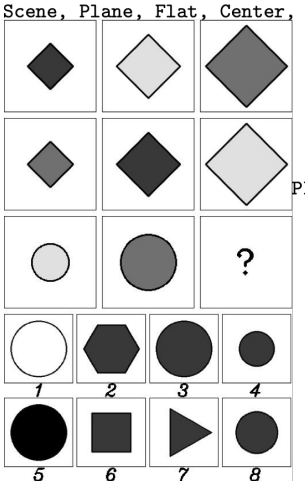
Plane
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Plane, Flat, Center, /, /, /, /



Plane
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Distribute Three:Color]

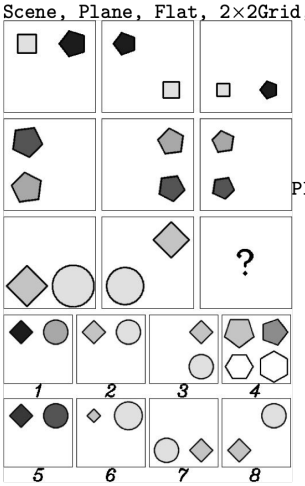
Scene, Plane, Flat, Center, /, /, /, /



Plane
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Distribute Three:Color]

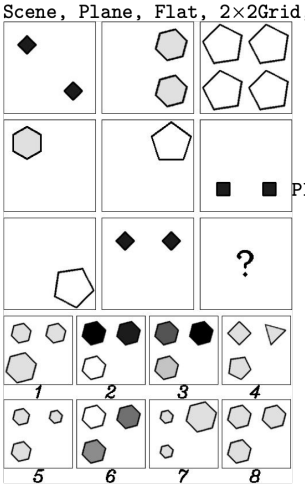
Solution (from left to right, up to down): 5, 3, 2, 4, 1, 3.

Scene, Plane, Flat, 2×2Grid, /, /, /, /



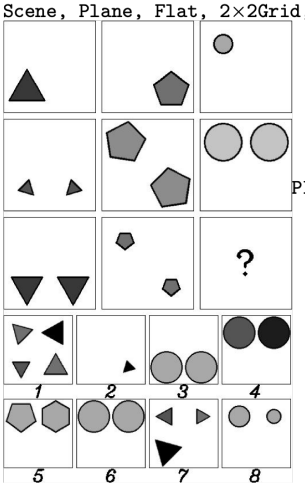
Plane
[Constant:Number]
[Progression:Position]
[Constant:Type]
[Progression:Size]
[Constant:Color]

Scene, Plane, Flat, 2×2Grid, /, /, /, /



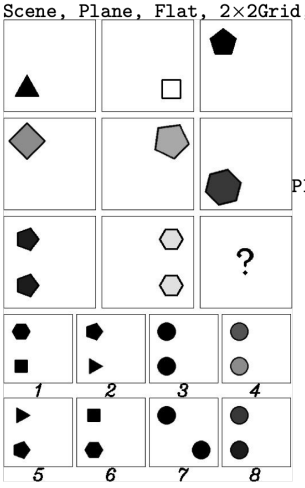
Plane
[Arithmetic:Number]
[NA:Position]
[Distribute Three:Type]
[Distribute Three:Size]
[Distribute Three:Color]

Scene, Plane, Flat, 2×2Grid, /, /, /, /



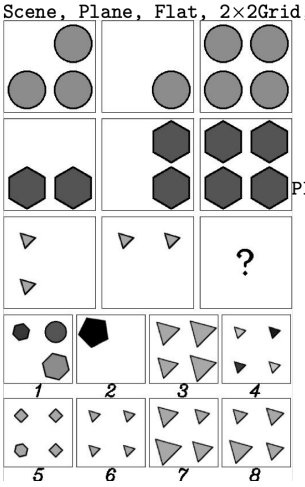
Plane
[Constant:Number]
[Progression:Position]
[Progression:Type]
[Distribute Three:Size]
[Progression:Color]

Scene, Plane, Flat, 2×2Grid, /, /, /, /



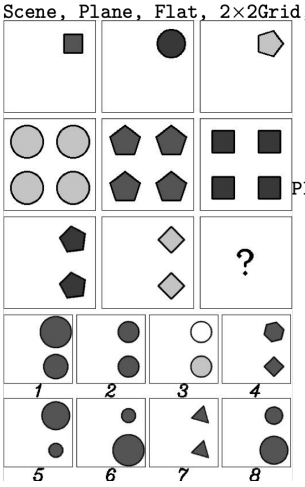
Plane
[Constant:Number]
[Progression:Position]
[Progression:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Plane, Flat, 2×2Grid, /, /, /, /



Plane
[Arithmetic:Number]
[NA:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]

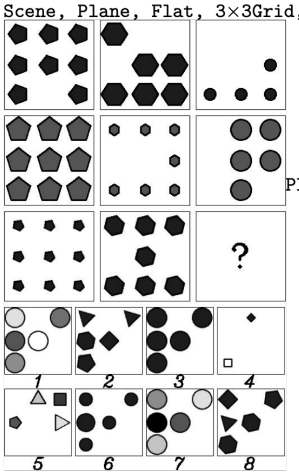
Scene, Plane, Flat, 2×2Grid, /, /, /, /



Plane
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Distribute Three:Color]

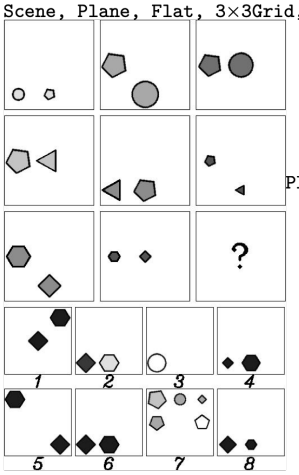
Solution (from left to right, up to down): 2, 8, 6, 3, 6, 2.

Scene, Plane, Flat, 3×3Grid, /, /, /, /



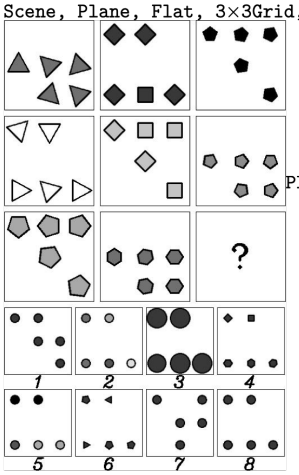
Plane
[Progression:Number]
[NA:Position]
[Progression:Type]
[Distribute Three:Size]
[Constant:Color]

Scene, Plane, Flat, 3×3Grid, /, /, /, /



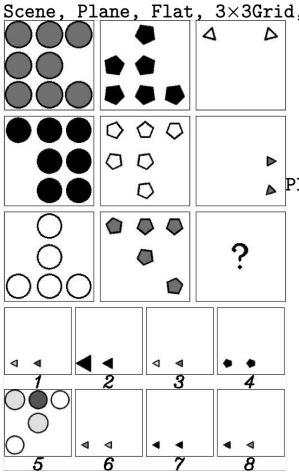
Plane
[Constant:Number]
[Distribute Three:Position]
[Constant:Type]
[Distribute Three:Size]
[Progression:Color]

Scene, Plane, Flat, 3×3Grid, /, /, /, /



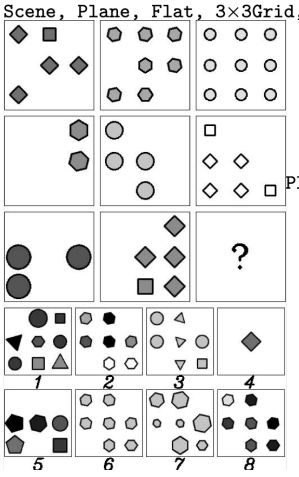
Plane
[Constant:Number]
[Distribute Three:Position]
[Progression:Type]
[Progression:Size]
[Progression:Color]

Scene, Plane, Flat, 3×3Grid, /, /, /, /



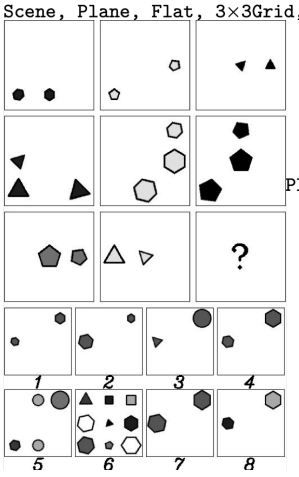
Plane
[NA:Number]
[Arithmetic:Position]
[Progression:Type]
[Progression:Size]
[Distribute Three:Color]

Scene, Plane, Flat, 3×3Grid, /, /, /, /



Plane
[Progression:Number]
[NA:Position]
[Distribute Three:Type]
[Progression:Size]
[Progression:Color]

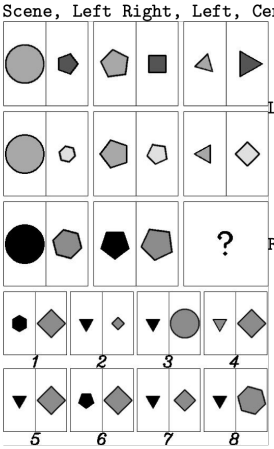
Scene, Plane, Flat, 3×3Grid, /, /, /, /



Plane
[Constant:Number]
[Progression:Position]
[Distribute Three:Type]
[Constant:Size]
[Arithmetic:Color]

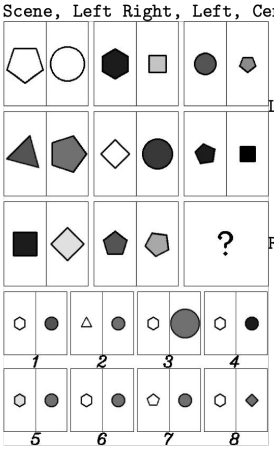
Solution (from left to right, up to down): 3, 6, 8, 7, 6, 4.

Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



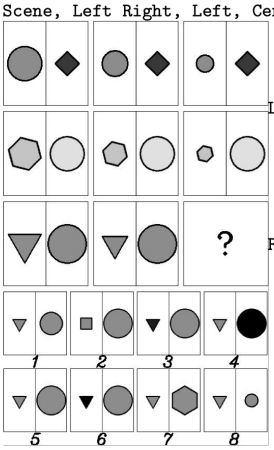
Left
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Constant:Color]
Right
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Constant:Color]

Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



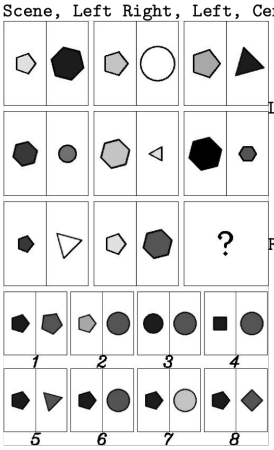
Left
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Distribute Three:Color]
Right
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Progression:Size]
[Progression:Color]

Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



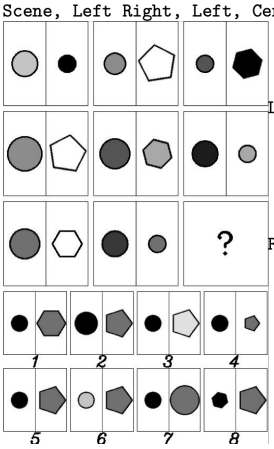
Left
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Constant:Color]
Right
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]

Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



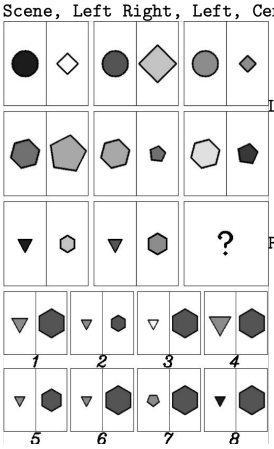
Left
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Arithmetic:Color]
Right
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



Left
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Progression:Color]
Right
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Distribute Three:Size]
[Arithmetic:Color]

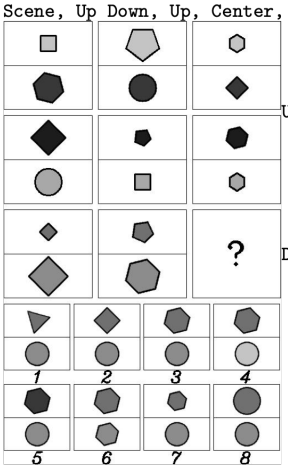
Scene, Left Right, Left, Center, /, /, Right, Center, /, /, /, /



Left
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Progression:Color]
Right
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Progression:Color]

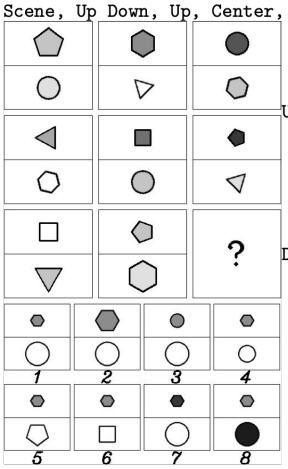
Solution (from left to right, up to down): 5, 6, 5, 6, 5, 6.

Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



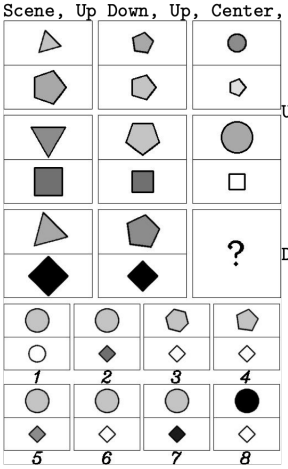
Up
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Distribute Three:Size]
[Constant:Color]
Down
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Progression:Size]
[Constant:Color]

Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



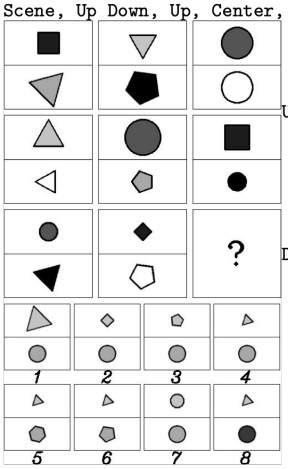
Up
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Progression:Color]
Down
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Distribute Three:Color]

Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



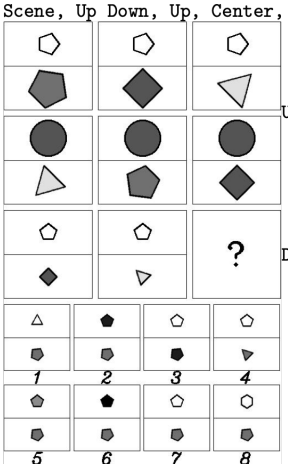
Up
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Distribute Three:Color]
Down
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Arithmetic:Color]

Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



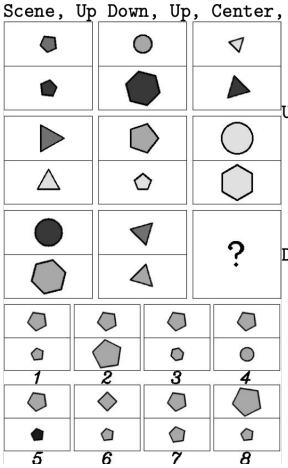
Up
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Distribute Three:Color]
Down
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Progression:Size]
[Distribute Three:Color]

Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



Up
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]
Down
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Distribute Three:Color]

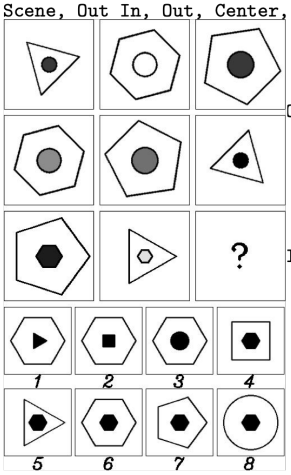
Scene, Up Down, Up, Center, /, /, Down, Center, /, /, /, /



Up
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Progression:Color]
Down
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Distribute Three:Size]
[Constant:Color]

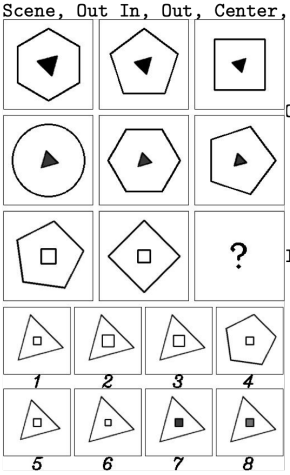
Solution (from left to right, up to down): 3, 1, 6, 4, 7, 1.

Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



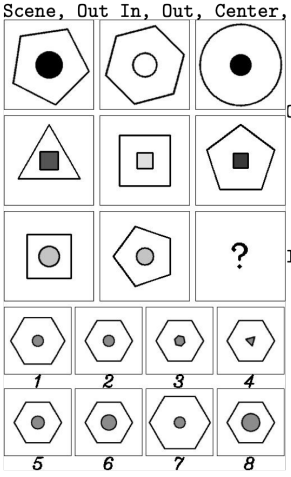
Out
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Distribute Three:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Arithmetic:Color]

Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



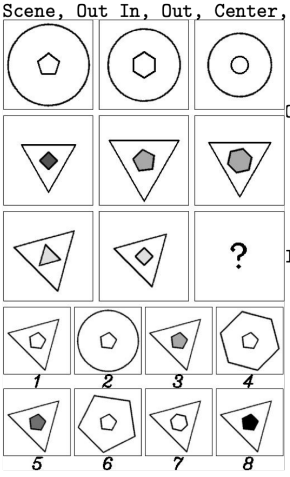
Out
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Constant:Color]

Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



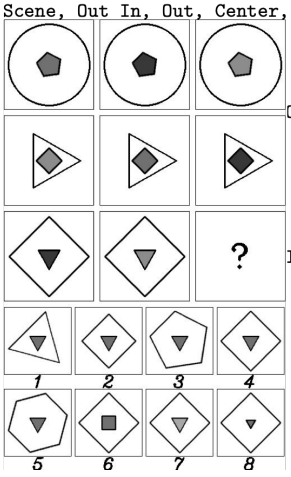
Out
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Progression:Size]
[Arithmetic:Color]

Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



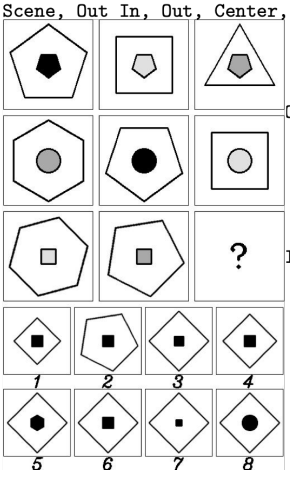
Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Distribute Three:Size]
[Arithmetic:Color]

Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Distribute Three:Color]

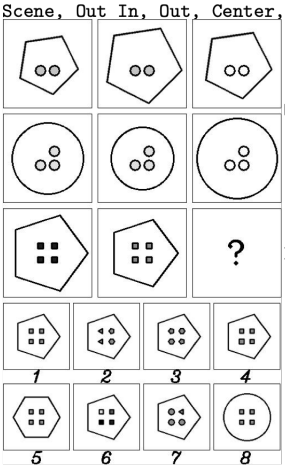
Scene, Out In, Out, Center, /, /, In, Center, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Constant:Color]
In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Distribute Three:Color]

Solution (from left to right, up to down): 6, 1, 2, 1, 4, 6.

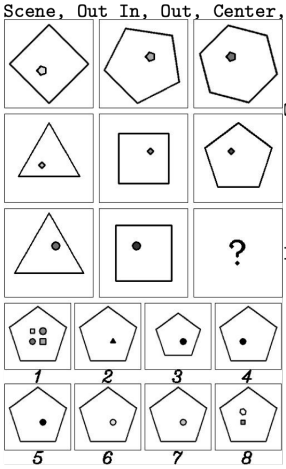
Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Constant:Color]

In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Arithmetic:Color]

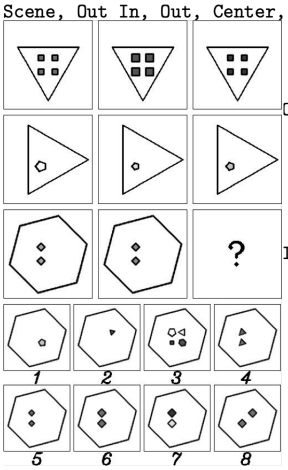
Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Constant:Size]
[Constant:Color]

In
[Constant:Number]
[Progression:Position]
[Constant:Type]
[Constant:Size]
[Progression:Color]

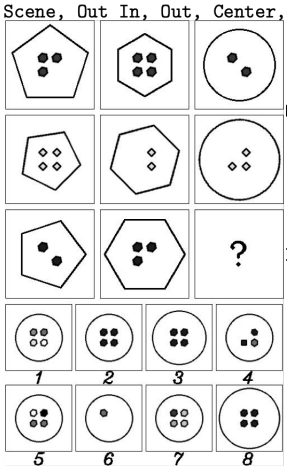
Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]

In
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Progression:Color]

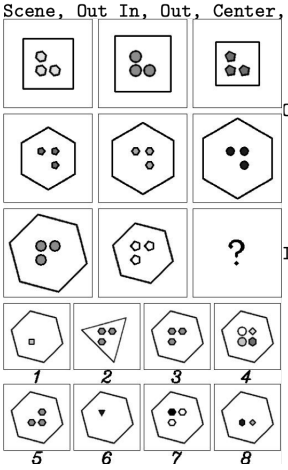
Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Progression:Type]
[Distribute Three:Size]
[Constant:Color]

In
[Distribute Three:Number]
[NA:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]

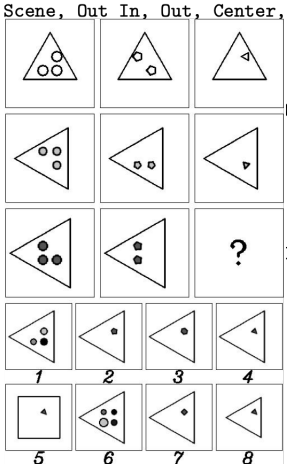
Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Distribute Three:Size]
[Constant:Color]

In
[Constant:Number]
[Constant:Position]
[Distribute Three:Type]
[Constant:Size]
[Arithmetic:Color]

Scene, Out In, Out, Center, /, /, In, 2×2Grid, /, /, /, /



Out
[Constant:Number]
[Constant:Position]
[Constant:Type]
[Constant:Size]
[Constant:Color]

In
[Arithmetic:Number]
[NA:Position]
[Progression:Type]
[Constant:Size]
[Constant:Color]

Solution (from left to right, up to down): 1, 5, 6, 2, 3, 4.