AdvancedGAP

October 18, 2016

1 Advanced GAP Programming

Note: Uses current GAP `master` branch and the current version of the GAP Jupyter kernel

If you want go to https://cloud.gap-system.org/ with username `codima2016` and password `uuShan0i`, and play along (or not along)

1.1 Type System, Bespoke Objects, Extending Functionality

- (Almost) everything in GAP is an *Object*
- Every Object has a *Type*
- In *GAP* a type is a pair $(Family, Filter)$
- Families partition objects
- Filters are hierarchical starting at `IsObject`

```
In [1]: TypeObj(true)
```

```
<Type: (BooleanFamily, [ IsBool, IsInternalRep ])>
```

```
In [2]: TypeObj(Group((1,2,3)))
```

```
<Type: (CollectionsFamily(...), [ IsComponentObjectRep, IsAttributeStoringRep, IsLi
```

```
In [3]: Display(TypeObj(Group( (1,2,3) )))
```

```
family:
    CollectionsFamily(...)
filters:
    IsComponentObjectRep
    IsAttributeStoringRep
    IsListOrCollection
    IsCollection
    IsFinite
    Tester(IsFinite)
    CanEasilyCompareElements
    Tester(CanEasilyCompareElements)
    CanEasilySortElements
    Tester(CanEasilySortElements)
    CanComputeSize
```

```
IsDuplicateFree
Tester(IsDuplicateFree)
IsExtLElement
CategoryCollections(IsExtLElement)
IsExtRElement
CategoryCollections(IsExtRElement)
CategoryCollections(IsMultiplicativeElement)
CategoryCollections(IsMultiplicativeElementWithOne)
CategoryCollections(IsMultiplicativeElementWithInverse)
IsOddAdditiveNestingDepthObject
CategoryCollections(IsAssociativeElement)
CategoryCollections(IsFiniteOrderElement)
IsGeneralizedDomain
CategoryCollections(IsPerm)
IsMagma
IsMagmaWithOne
IsMagmaWithInversesIfNonzero
IsMagmaWithInverses
Tester(GeneratorsOfMagmaWithInverses)
IsGeneratorsOfMagmaWithInverses
Tester(IsGeneratorsOfMagmaWithInverses)
IsAssociative
Tester(IsAssociative)
IsCommutative
Tester(IsCommutative)
Tester(MultiplicativeNeutralElement)
IsGeneratorsOfSemigroup
Tester(IsGeneratorsOfSemigroup)
IsSimpleSemigroup
Tester(IsSimpleSemigroup)
IsRegularSemigroup
Tester(IsRegularSemigroup)
IsInverseSemigroup
Tester(IsInverseSemigroup)
IsCompletelyRegularSemigroup
Tester(IsCompletelyRegularSemigroup)
IsCompletelySimpleSemigroup
Tester(IsCompletelySimpleSemigroup)
IsGroupAsSemigroup
Tester(IsGroupAsSemigroup)
IsMonoidAsSemigroup
Tester(IsMonoidAsSemigroup)
IsOrthodoxSemigroup
Tester(IsOrthodoxSemigroup)
IsCyclic
Tester(IsCyclic)
IsFinitelyGeneratedGroup
Tester(IsFinitelyGeneratedGroup)
```

```
    IsSubsetLocallyFiniteGroup
    Tester(IsSubsetLocallyFiniteGroup)
    CanEasilyTestMembership
    CanEasilyComputeWithIndependentGensAbelianGroup
    CanComputeSizeAnySubgroup
    KnowsHowToDecompose
    Tester(KnowsHowToDecompose)
    IsNilpotentGroup
    Tester(IsNilpotentGroup)
    IsSupersolvableGroup
    Tester(IsSupersolvableGroup)
    IsMonomialGroup
    Tester(IsMonomialGroup)
    IsSolvableGroup
    Tester(IsSolvableGroup)
    IsPolycyclicGroup
    Tester(IsPolycyclicGroup)
    CanEasilyComputePcgs
    CanComputeFittingFree
    IsNilpotentByFinite
    Tester(IsNilpotentByFinite)
```

```
In [4]: TypeObj(1)

<Type: (CyclotomicsFamily, [ IsInt, IsRat, IsCyc, ... ])>

In [5]: TypeObj(TypeObj(1))

<Type: (FamilyOfTypes, [ IsPositionalObjectRep, IsType, IsTypeDefaultRep ])>

In [6]: TypeObj(TypeObj(TypeObj(1)))

<Type: (FamilyOfTypes, [ IsPositionalObjectRep, IsType, IsTypeDefaultRep ])>
```

$\Rightarrow$ There is a *Type of Types* which contains itself, the same is true for Families

```
In [7]: FamilyObj(true)

<Family: "BooleanFamily">

In [8]: FamilyObj(1)

<Family: "CyclotomicsFamily">

In [9]: Display(FamilyObj((1,2,3)))

name:
    PermutationsFamily
required filters:
```

```
    IsPerm
implied filters:
    IsPerm
    CanEasilyCompareElements
    Tester(CanEasilyCompareElements)
    CanEasilySortElements
    Tester(CanEasilySortElements)
    IsExtLElement
    IsExtRElement
    IsMultiplicativeElement
    IsMultiplicativeElementWithOne
    IsMultiplicativeElementWithInverse
    IsAssociativeElement
    IsFiniteOrderElement
```

Play around a bit, find out what some of these things are, maybe ask GAP about an object I didn't demonstrate yet.

Think about what the type of a list should be.

2 Families of Group Elements

2.1 Elements of free groups get their own family

it does not make sense to multiply an element of $F(\{a, b\})$ with an element of $F(\{x, y\})$

```
In [10]: F := FreeGroup(2)

<group with 2 generators>

In [11]: Display(FamilyObj(F));

name:
    CollectionsFamily(...)
required filters:
    IsCollection
implied filters:
    IsListOrCollection
    IsCollection
    IsExtLElement
    CategoryCollections(IsExtLElement)
    IsExtRElement
    CategoryCollections(IsExtRElement)
    CategoryCollections(IsMultiplicativeElement)
    CategoryCollections(IsMultiplicativeElementWithOne)
    CategoryCollections(IsMultiplicativeElementWithInverse)
    IsOddAdditiveNestingDepthObject
    CategoryCollections(IsAssociativeElement)
    IsGeneratorsOfMagmaWithInverses
```

```
    Tester(IsGeneratorsOfMagmaWithInverses)
    IsGeneratorsOfSemigroup
    Tester(IsGeneratorsOfSemigroup)
    CategoryCollections(IsWord)
    CategoryCollections((IsWord and IsAssociativeElement))
    CategoryCollections(((IsWord and IsAssociativeElement) and (IsWord and IsMultip
    CategoryCollections(((IsWord and IsAssociativeElement) and (IsWord and IsMultip
```

```
In [12]: FamilyObj(F.1)

<Family: "FreeGroupElementsFamily">

In [13]: G := FreeGroup(2);

<group with 2 generators>

In [14]: FamilyObj(G.1)

<Family: "FreeGroupElementsFamily">

In [15]: FamilyObj(F.1) = FamilyObj(G.1)

false
```

2.2 Permutations all lie in the same family

```
In [16]: G := SymmetricGroup(5)

Sym( [ 1 .. 5 ] )

In [17]: H := SymmetricGroup(5)

Sym( [ 1 .. 5 ] )

In [18]: IsIdenticalObj(G,H)

false

In [19]: FamilyObj(Representative(G)) = FamilyObj(Representative(H))

true
```

3 Type of Objects can *change*

- The *Family* of an Object stays static for its lifetime
- Certain *Filters* can change and reflect knowledge GAP has about the Object
- *Categories* are filters
- the category of an object *does not change*
- mathematically: "signature of an algebraic structure":

- Magma $\{\circ\}$, MagmaWithOne $\{\circ, 1\}$, MagmaWithInverses $\{\circ, ^{-1}\}$
- Graph E
- does *not* enforce any axioms!

```
In [20]: G := Group( [ (1,2,3,4,5,6,7,8,9,10,11), (3,7,11,8)(4,10,5,6) ] );

Group([ (1,2,3,4,5,6,7,8,9,10,11), (3,7,11,8)(4,10,5,6) ])

In [21]: Display(TypeObj(G));

family:
    CollectionsFamily(...)
filters:
    IsComponentObjectRep
    IsAttributeStoringRep
    IsListOrCollection
    IsCollection
    IsFinite
    Tester(IsFinite)
    CanEasilyCompareElements
    Tester(CanEasilyCompareElements)
    CanEasilySortElements
    Tester(CanEasilySortElements)
    CanComputeSize
    IsDuplicateFree
    Tester(IsDuplicateFree)
    IsExtLElement
    CategoryCollections(IsExtLElement)
    IsExtRElement
    CategoryCollections(IsExtRElement)
    CategoryCollections(IsMultiplicativeElement)
    CategoryCollections(IsMultiplicativeElementWithOne)
    CategoryCollections(IsMultiplicativeElementWithInverse)
    IsOddAdditiveNestingDepthObject
    CategoryCollections(IsAssociativeElement)
    CategoryCollections(IsFiniteOrderElement)
    IsGeneralizedDomain
    CategoryCollections(IsPerm)
    Tester(LargestMovedPoint)
    IsMagma
    IsMagmaWithOne
    IsMagmaWithInversesIfNonzero
    IsMagmaWithInverses
    Tester(GeneratorsOfMagmaWithInverses)
    IsGeneratorsOfMagmaWithInverses
    Tester(IsGeneratorsOfMagmaWithInverses)
    IsAssociative
    Tester(IsAssociative)
    Tester(MultiplicativeNeutralElement)
```

```
    IsGeneratorsOfSemigroup
    Tester(IsGeneratorsOfSemigroup)
    IsSimpleSemigroup
    Tester(IsSimpleSemigroup)
    IsRegularSemigroup
    Tester(IsRegularSemigroup)
    IsInverseSemigroup
    Tester(IsInverseSemigroup)
    IsCompletelyRegularSemigroup
    Tester(IsCompletelyRegularSemigroup)
    IsCompletelySimpleSemigroup
    Tester(IsCompletelySimpleSemigroup)
    IsGroupAsSemigroup
    Tester(IsGroupAsSemigroup)
    IsMonoidAsSemigroup
    Tester(IsMonoidAsSemigroup)
    IsOrthodoxSemigroup
    Tester(IsOrthodoxSemigroup)
    IsFinitelyGeneratedGroup
    Tester(IsFinitelyGeneratedGroup)
    IsSubsetLocallyFiniteGroup
    Tester(IsSubsetLocallyFiniteGroup)
    CanEasilyTestMembership
    CanComputeSizeAnySubgroup
    KnowsHowToDecompose
    Tester(KnowsHowToDecompose)
    CanComputeFittingFree
    IsNilpotentByFinite
    Tester(IsNilpotentByFinite)
```

```
In [22]: JoinStringsWithSeparator(CategoriesOfObject(G), "\n");
```

```
"IsListOrCollection
IsCollection
IsExtLElement
CategoryCollections(IsExtLElement)
IsExtRElement
CategoryCollections(IsExtRElement)
CategoryCollections(IsMultiplicativeElement)
CategoryCollections(IsMultiplicativeElementWithOne)
CategoryCollections(IsMultiplicativeElementWithInverse)
CategoryCollections(IsAssociativeElement)
CategoryCollections(IsFiniteOrderElement)
IsGeneralizedDomain
CategoryCollections(IsPerm)
IsMagma
IsMagmaWithOne
```

```
IsMagmaWithInversesIfNonzero
IsMagmaWithInverses"

In [23]: KnownAttributesOfObject(G)

[ "LargestMovedPoint", "GeneratorsOfMagmaWithInverses", "MultiplicativeNeutralEleme
```

3.0.1 Note: Nothing reflects the fact that G is a group.

```
In [24]: IsGroup(G);

true

In [25]: IsGroup;

<Filter "(IsMagmaWithInverses and IsAssociative)">

In [26]: IsMagmaWithInverses;

<Category "IsMagmaWithInverses">

In [27]: IsAssociative;

<Property "IsAssociative">

In [28]: HasSize(G)

false

In [29]: Size(G)

7920

In [30]: HasSize(G)

true

In [31]: Display(TypeObj(G))

family:
    CollectionsFamily(...)
filters:
    IsComponentObjectRep
    IsAttributeStoringRep
    IsListOrCollection
    IsCollection
    Tester(IsEmpty)
    Tester(IsTrivial)
    IsNonTrivial
    Tester(IsNonTrivial)
    IsFinite
```

```
Tester(IsFinite)
Tester(Size)
CanEasilyCompareElements
Tester(CanEasilyCompareElements)
CanEasilySortElements
Tester(CanEasilySortElements)
CanComputeSize
IsDuplicateFree
Tester(IsDuplicateFree)
IsExtLElement
CategoryCollections(IsExtLElement)
IsExtRElement
CategoryCollections(IsExtRElement)
CategoryCollections(IsMultiplicativeElement)
CategoryCollections(IsMultiplicativeElementWithOne)
CategoryCollections(IsMultiplicativeElementWithInverse)
IsOddAdditiveNestingDepthObject
CategoryCollections(IsAssociativeElement)
CategoryCollections(IsFiniteOrderElement)
Tester(OneImmutable)
IsGeneralizedDomain
CategoryCollections(IsPerm)
Tester(LargestMovedPoint)
Tester(NrMovedPoints)
Tester(MovedPoints)
IsMagma
IsMagmaWithOne
IsMagmaWithInversesIfNonzero
IsMagmaWithInverses
Tester(GeneratorsOfMagmaWithInverses)
IsGeneratorsOfMagmaWithInverses
Tester(IsGeneratorsOfMagmaWithInverses)
IsAssociative
Tester(IsAssociative)
Tester(MultiplicativeNeutralElement)
IsGeneratorsOfSemigroup
Tester(IsGeneratorsOfSemigroup)
IsSimpleSemigroup
Tester(IsSimpleSemigroup)
IsRegularSemigroup
Tester(IsRegularSemigroup)
IsInverseSemigroup
Tester(IsInverseSemigroup)
IsCompletelyRegularSemigroup
Tester(IsCompletelyRegularSemigroup)
IsCompletelySimpleSemigroup
Tester(IsCompletelySimpleSemigroup)
IsGroupAsSemigroup
```

```
    Tester(IsGroupAsSemigroup)
    IsMonoidAsSemigroup
    Tester(IsMonoidAsSemigroup)
    IsOrthodoxSemigroup
    Tester(IsOrthodoxSemigroup)
    IsFinitelyGeneratedGroup
    Tester(IsFinitelyGeneratedGroup)
    IsSubsetLocallyFiniteGroup
    Tester(IsSubsetLocallyFiniteGroup)
    CanEasilyTestMembership
    CanComputeSizeAnySubgroup
    KnowsHowToDecompose
    Tester(KnowsHowToDecompose)
    Tester(IsNilpotentGroup)
    Tester(IsSolvableGroup)
    Tester(StabChainMutable)
    Tester(StabChainOptions)
    CanComputeFittingFree
    IsNilpotentByFinite
    Tester(IsNilpotentByFinite)
    Tester(IsTorsionFree)
    Tester(IsFreeAbelian)


In [32]: Size;

<Attribute "Size">

In [33]: IsNilpotent(G)

false

In [34]: IsCommutative(G)

false

In [35]: IsNilpotent;

<Operation "IsNilpotent">

In [36]: IsCommutative;

<Property "IsCommutative">

In [37]: IsSimpleGroup(G)

true

In [38]: SylowSubgroup(G, 2);

Group([ (2,6)(3,7)(4,10)(8,9), (3,7)(4,8)(5,11)(9,10), (2,3)(5,11)(6,7)(8,9), (2,10
```

3.0.2 *Warning*

```
In [39]: S := Semigroup([Transformation([2,1,3])])

<commutative transformation semigroup of degree 2 with 1 generator>

In [40]: IsGroup(S)

false
```

3.0.3 Maybe Wilf will talk about this.

4 Operation, Attribute, Property

4.1 Operation

- defined by a name O and a k-tuple ($1 \leq k \leq 6$) of *filters*
- install *methods* for O

4.2 Attribute

- defined by a name A and a filter F
- a value attached to an object in F set
- value can be cached (*WARNING*: then immutable!)
- an operation A (with 1-tuple (F) of filters)
- an property Has$A$ (if the value is known)
- an operation Set$A$

4.3 Property

- defined by a name P and a filter F
- a boolean attribute
- also a filter P

```
In [41]: IsCommutative

<Property "IsCommutative">
```

5 Worked example: Graphs

```
In [42]: GraphsFamily := NewFamily("GraphsFamily")

<Family: "GraphsFamily">

In [43]: DeclareCategory("IsGraph", IsObject);
```

(nb: DeclareX(name, ...) is essentially the same as BindGlobal(name, NewX(...))

```
In [44]: IsGraph

<Category "IsGraph">

In [45]: BindGlobal("GraphType", NewType(GraphsFamily, IsGraph))



In [46]: GraphType;

<Type: (GraphsFamily, [ IsGraph ])>

In [47]: BindGlobal("NewGraph",
         function(v,e)
             return Objectify(GraphType, rec( vertices := v, edges := e ));
         end);



In [48]: G := NewGraph([1,2,3], [[1,2],[3,4]])

<object>

In [49]: InstallMethod(ViewString
                      , "for a graph"
                      , [IsGraph],
         function(g)
             local res;
             res := "<a graph with ";
             Append(res, String(Length(g!.vertices)));
             Append(res, " vertices and ");
             Append(res, String(Length(g!.edges)));
             Append(res, " edges>");
             return res;
         end)
```

6 Exercise

- Vertices
- Edges
- IsConnectedGraph
- GraphUnion
- GraphsIsomorphism

7 OR: Make some object relevant to your research!

```
In [50]: DeclareAttribute("Vertices", IsGraph)
```

```
In [51]: DeclareProperty("IsConnectedGraph", IsGraph)
```

```
In [52]: DeclareOperation("GraphsIsomorphism", [IsGraph, IsGraph])
```

8 Representations

- A representation is just a filter
- subfilter of `IsComponentObjectRep` or `IsPositionalObjectRep`
- `IsAttributeStoringRep` causes values of attributes to be cached, only possible for `IsComponentObjectRep`

```
In [53]: DeclareRepresentation("IsGraphByAdjacencyListRep", IsGraph and IsComponent
```

```
In [54]: IsGraphByAdjacencyListRep
```

```
<Representation "IsGraphByAdjacencyListRep">
```

```
In [55]: BindGlobal("GraphByAdjacencyListType", NewType(GraphsFamily, IsGraph and I
```

```
In [56]: InstallMethod( ViewString,
                        "for a graph by adjacency list",
                        [IsGraph and IsGraphByAdjacencyListRep],
                        function(g)
                           return "<a graph in adjacency list representation>";
                        end)
```

```
In [57]: G := Objectify(GraphByAdjacencyListType, rec( vertices := [], adjacency :=
```

```
<a graph in adjacency list representation>
```

9 Possible Exercises

- Define different representations for graphs
- Can you abstract graphs enough to be able to implement generic algorithms?
- Implement IsConnectedGraph for different representations

```
In [ ]:
```