Writing A GAP Package For Local Action Diagrams

Marcus Chijoff

Supervised by Stephan Tornier

The University of Newcastle

March 3, 2023

Contents

1 Abstract

Local action diagrams are in a one-to-one correspondence with (P)-closed groups acting on trees. When the action of a (P)-closed group on a tree has finitely many orbits and the tree is locally finite the corresponding local action diagram is also finite. We provide an implementation of finite local action diagrams in GAP. It's able to determine if two local action diagrams are isomorphic, if they contain strongly confluent partial orientations, and enumerate, up to isomorphism, all local action diagrams corresponding to (P)-closed subgroups of $\mathrm{Aut}(T_d)$ with two vertex oribts.

2 Introduction

Groups are an algebraic structure that represent the symmetry of objects. The mappings on a graph that preserve edge connectivity are called automorphisms and together with the operation of function composition they form a group. The '(P)-closure' of a group of automorphisms consists of all the automorphisms that 'act the same locally' as an element of the group at each vertex of the graph. This means that these groups can be described by what's happening locally at each vertex.

Local action diagrams were introduced by C. Reid and S. Smith. [1] They are in a one-to-one correspondence with (P)-closed groups which means that every (P)-closed group has a corresponding local action diagram and every local action diagram corresponds with a (P)-closed group. A local action diagram consists of an underlying digraph, a group labelling each vertex, a set labelling each arc that is compatible with the group labelling each vertex, and a reversal mapping.

The orbits of a group action are the partition of the domain into sets containing all elements that can 'reach each other' by the group action. When a (P)-closed group is acting on a tree, the corresponding local action diagram has a vertex for each orbit of the action. This means that when there is a finite number of orbits there is a finite number of vertices in the local action diagram. Furthermore, when the tree is locally finite — meaning it has a finite number of edges originating from each vertex — then the local action diagram has a finite number of edges. This means that we are able to describe infinite groups acting on infinite trees with a finite amount of information.

GAP is a computer algebra system designed with a focus on group theory. [2] There are a number of community made packages that extend its functionality. We developed a GAP package which implements local action diagrams. The package makes use of the digraphs package to implement a category in GAP for local action diagrams. It also provides functions to check for local action diagram isomorphisms, find strongly confluent partial orientations in local action diagrams, and enumerate all local action diagrams corresponding to (P)-closed groups acting on a regular tree of degree d and either 1 or 2 vertex orbits.

3 Statement of Authorship

The code implementing the local action diagram category in GAP, finding local action diagram isomorphisms, and enumerating local action diagrams with two vertices was written by M. Chijoff. The digraphs package for GAP was used in the implementation. M. Chijoff wrote the code to load this as a package in GAP. S. Tornier wrote the code to draw local action diagrams with one and two vertices in TikZ and wrote the original code to enumerate local action diagrams with one vertex. M. Chijoff adapted the original code to enumerate local action diagrams with one vertex to work with the local action diagram category.

4 Background

In the following section, we use the same notation and definitions as [1] and this notation will be used throughout the report.

4.1 Groups

A **group** is a set, G, with a binary operation, $\cdot$, satisfying the following axioms:

- **Identity Element:** There as an $e \in G$ such that for all $g \in G$, $g \cdot e = e \cdot g = g$.
- **Inverses:** For all $g \in G$, there exists an $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = e$.
- **Associativity:** For all $x, y, z \in G$, $x \cdot (y \cdot z) = (x \cdot y) \cdot z$.
- **Closure:** For all $x, y \in G$, $x \cdot y \in G$.

When the operation is clear from context we omit writing an explicit symbol for it.

For a group, G, we define a **subgroup**, H, of G to be a subset of G which forms a group with the same group operation. We write $H \leq G$ to say that H is a subgroup of G.

For a group, G, and a set, X, a **group action** on X is a function $\alpha : G \times X \to X$ satisfying:

- **Identity:** For all $x \in X$, $\alpha(e, x) = x$.
- **Compatibility:** For all $g, h \in G$ and $x \in X$, $\alpha(g, \alpha(h, x)) = \alpha(gh, x)$.

When it is clear from context, we write gx to mean $\alpha(g, x)$ for $g \in G$ and $x \in X$.

For each element, x, of X we define the **orbit** of x — denoted by $G \cdot x$ — to be $G \cdot x \coloneqq \{g \cdot x : g \in G\}$, i.e. the set of elements in X which can be reached by the action of G. The set of orbits of a group action form a partition of X and we denote this set by $G \backslash X$.

4.2 Graphs

A **directed graph** $\Gamma = (V, A, o, r)$ consists of a vertex set, V, a set of arcs, A, a map $o : A \to V$ which assigns each arc to an **origin vertex**, and a bijection $r : A \to A$ called an **edge reversal** and such that $r^2 = \mathrm{id}$. We denote the reversal mapping by $a \mapsto \overline{a}$ and define the **terminal vertex** of an arc to be $t(a) \coloneqq o(\overline{a})$. For each $v \in V$, the set $o^{-1}(v)$ is the set of all arcs which originate at v. For a graph, Γ, we denote the set of vertices by $V\Gamma$ and the set of edges by $A\Gamma$ when not clear from context. A **subgraph** $\Gamma' = (V', A', o', r')$ is a graph such that $V' \subseteq V$, $A' \subseteq A$, and o' and r' equal to the restriction of o and r to A'.

An **isomorphism** between two graphs, Γ and Γ', is a pair of bijections, $\theta_V : V\Gamma \to V\Gamma'$ and $\theta_A : A\Gamma \to A\Gamma'$, which respect the origin vertices and edge reversal. This means that $\theta_V(o(v)) = o(\theta_V(v))$ and $\overline{\theta_A(a)} = \theta_A(\overline{a})$ for all $v \in V\Gamma$ and $a \in A\Gamma$. An isomorphism from a graph to itself is called an **automorphism**. The set of all automorphisms of a graph together with function composition forms a group and we denote this group by $\mathrm{Aut}(\Gamma)$.

For a group, G, acting on a graph, Γ, we define the **quotient graph** $G\backslash\Gamma$ by:

- The vertex set, V_G, is the set of vertex orbits of G acting on Γ.
- The arc set is, A_G, the set of arc orbits of G acting on Γ.
- The origin map $\tilde{o} : A_G \to A_G$ is defined by $\tilde{o}(Ga) = Go(a)$.
- The reversal map $\tilde{r} : A_G \to A_G$ is defined by $Ga \mapsto G\overline{a}$.

A graph is **simple** if for each $u, v \in V$ there is at most one arc of the form (u, v). For an index set $I \subseteq \mathbb{Z}$ we define $\hat{I} := \{i \in I : i + 1 \in I\}$. Then a **path** indexed by I is a sequence of vertices $(v_i)_{i\in I}$ and edges $(\{a_i, \overline{a_i}\})_{i\in\hat{I}}$ such that $\{a_i, \overline{a_i}\}$ is an edge in Γ between v_i and v_{i+1} for all $i \in \hat{I}$. A path is simple if all vertices in it are distinct. A **directed path** only includes arcs a_i where $o(a_i) = v_i$ and $t(a_i) = v_{i+1}$ for all $i \in \hat{I}$. If the index set is finite with maximum $n > 0$, $v_0 = v_n$, and all vertices in the path are distinct then we say that the path is a **cycle**. A subset, O, of $A\Gamma$ is an **orientation** if for each $a \in A\Gamma$, either $a \in O$ or $\overline{a} \in O$. A **partial orientation** is a subset, O, of $A\Gamma$ such that for all $a \in A\Gamma$, if $a \in O$ then $\overline{a} \notin O$.

We define the **length** of a path to be the number of arcs in the sequence of arcs. If there is a path between two vertices, $u, v \in V$, then there is a path of minimal length between them. We call the length of this path the **distance** between u and v and denote it by $d(u, v)$. For $k \geq 1$ and $v \in V$ we denote by $B(v, k)$ the **ball** of radius k centred at v. This is the subgraph of Γ induced by all vertices that are a distance of at most k away from v.

A graph is **connected** if there is a path between any two distinct vertices. A **tree** is a non-empty simple, connected graph that contains no cycles. The **regular tree** of degree d — denoted by T_d — is a tree such that $\left|o^{-1}(v)\right| = d$ for all $v \in VT_d$.

5 Local Action Diagrams

5.1 Property (P)

Given a tree T and $H \leq \mathrm{Aut}(T)$ we define the following group

$$H^{(P)} := \left\{g \in \mathrm{Aut}(T) : \forall v \in V\ \exists h_v \in H : h_v|_{B(v,1)} = g|_{B(v,1)}\right\}.$$

This is known as the (P)-closure of H. We say that H is (P)**-closed** if $H = H^{(P)}$. [1] The set $H^{(P)}$ is the group of all automorphisms that 'act locally like H.'

5.2 Local Action Diagrams

Local action diagrams were first described by C. Reid and S. Smith. [1] It was proven by them that local action diagrams are in a one-to-one correspondence with (P)-closed groups.

Definition 5.1 *A* ***local action diagram*** $\Delta = (\Gamma, (X_a), (G(v)))$ *consists of*

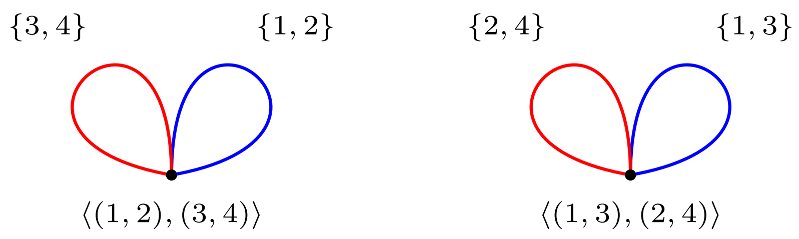


Figure 1: Two Isomorphic Local Action Diagrams

- *A connected graph* Γ.
- *For each vertex,* v*, of* Γ *a group* $G(v)$ *labelling the vertex.*
- *For each arc,* a*, of* Γ *a set* X_a *such that* X_a *is a distinct orbit of* $G(o(a))$.

For each $v \in V\Gamma$ we write $X_v \coloneqq \cup_{a\in o^{-1}(v)} X_a$. Note that when drawing local action diagrams in this report we use colours on the arcs to represent the reversal mapping. Arcs between two vertices with the same colour are the reverse of each other and arcs that loop on a vertex with the same colour are the reverse of each other.

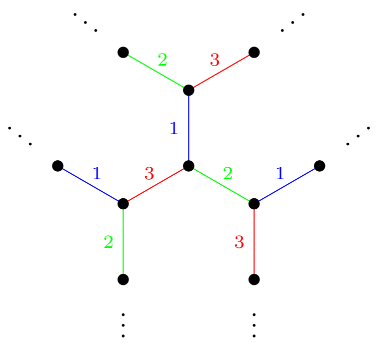
Definition 5.2 *If* $\Delta = (\Gamma, (X_a), (G(v)))$ *and* $\Delta' = (\Gamma', (X'_a), (G'(v)))$ *are two local action diagrams then we define an isomorphism* $\boldsymbol{\theta} = (\theta, (\theta_v))$ *between them to be a graph isomorphism* $\theta : \Gamma \to \Gamma'$ *and for each* $v \in V\Gamma$ *a bijection* $\theta_v : X_v \to X_{\theta(v)}$ *which restricts to a bijection from* X_a *to* $X_{\theta(a)}$ *and also such that* $\theta_v G(v) \theta_v^{-1} = G'(v)$.

Figure 1 shows two isomorphic local action diagrams. An isomorphism between them is $\boldsymbol{\theta} = (\theta, (\theta_1))$ where θ is one of the digraph isomorphisms and

$$\theta_1 = \begin{cases} 1 \mapsto 1 \\ 2 \mapsto 3 \\ 3 \mapsto 2 \\ 4 \mapsto 4 \end{cases}$$

If T is a (potentially infinite) tree and G is a (potentially infinite) (P)-closed group acting on T then the underlying graph, Γ, of the associated local action diagram is the quotient graph $G\backslash T$. Thus, if there are finitely many orbits of G acting on T then there are finitely many vertices in the local action diagrams and if T is locally finite — i.e. $\left|o^{-1}(v)\right| < \infty$ for all $v \in VT$ — then there are finitely many arcs in the local action diagram.

In Figure 2 the group acting on the tree in the left consists of all automorphisms of the tree that preserve the edge labelling. This group is (P)-closed and the corresponding local action diagram is on the right. The group only has one orbit which is why which is why the local action diagram only has one vertex.



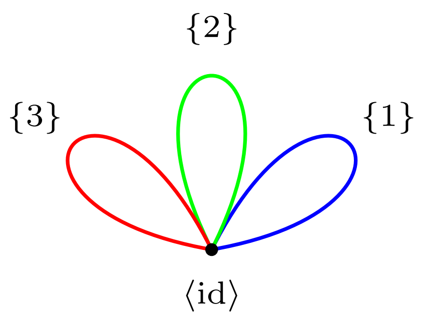


Figure 2: An Example of A Tree and Group Action (left) and The Associated Local Action Diagram (right)

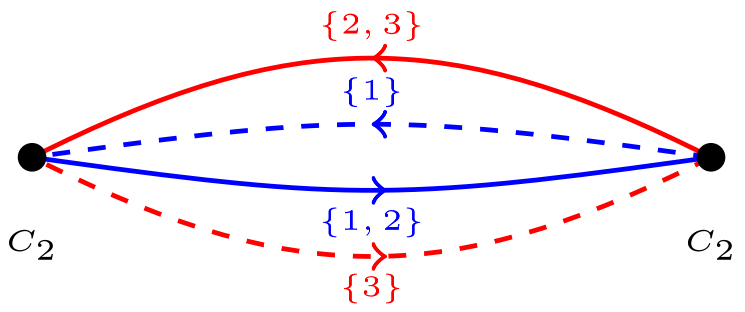


Figure 3: A Local Action Diagram With a Non-trivial Scopo (The Dashed Arcs)

5.3 Strongly Confluent Partial Orientations

Definition 5.3 *Let $\Delta = (\Gamma, (X_a), (G(v)))$ be a local action diagram. Then a* ***strongly confluent partial orientation (scopo)*** *of Δ is a subset O of $A\Gamma$ such that*

- *If $a \in O$ then $\overline{a} \notin O$ and $|X_a| = 1$.*
- *For all $v \in V\Gamma$, if O contains an arc originating at v then O contains all other arcs (apart from $\overline{a}$) that terminate at v.*

The first part of Definition 5.3 is the 'partial orientation part' and the second part of it is the 'strongly confluent part'. Also note that the empty scopo is a valid scopo. Scopos in a local action diagrams can be used to find invariant subtrees and ends in the associated tree with respect to the group action. For example, if a local action diagram only has the empty scopo then the associated group action is geometrically dense, which means that the group action leaves no invariant subtrees or ends in the tree, other than the whole tree.

Figure 3 shows a local action diagram with a non-trivial scopo. The dashed arcs are the arcs in the scopo and arcs with the same colour are reverses of each other.

6 Implementation In GAP

GAP is a computer algebra system with a focus on combinatorics and group theory. [2] It also has a number of community made packages which extend its functionality. Each variable in GAP has categories assigned to it. For example, a variable storing the number 1 is part of the `IsInt` and `IsPosRat` categories among others. Each variable can also have a number of attributes associated with it.

To implement local action diagrams in GAP we define the category `IsLocalActionDiagram` and provide a constructor. The constructor requires a digraph, list a vertex labels, list of edge labels, and a reversal mapping as input. It will either check that they form a valid local action diagram or won't depending on the users choice of constructor. It will then create a copy of the digraph, assign it to the category `IsLocalActionDiagram`, assign the vertex labels, edge labels, and reversal mappings as attributes, and then return this value. The digraphs package for GAP is used to implement the digraph of local action diagrams. [3]

6.1 Finding Local Action Diagram Isomorphisms

We use an exhaustive search to find if two local action diagrams are isomorphic. Algorithm 1 shows the implementation in pseudocode.

Algorithm 1 returns the first isomorphism found if one exists; otherwise it returns `fail`, a value in GAP which means that nothing valid could be returned. The function `FindGroupConjugateBijection` will search through every valid bijection between the domains of the two vertex labels. It uses the digraph isomorphism mapping of the arcs so that it only searches through bijections which correctly restrict to the arc labels. If a local action diagrams isomorphism exists, then the function will return a list containing the digraph isomorphism and a list of bijections between the vertex labels.

6.2 Finding Scopos

We use an exhaustive search on a reduced search space to find all Scopos of a local action diagram. The search space is reduced by removing all arcs that violate at least one of the conditions for being part of a scopo. Algorithm 2 shows the implementation in pseudocode.

In Algorithm 2, the `ScopoSecondCondition` function checks if the second condition in Definition 5.3 holds for the given arc only — i.e. for each arc, a, it checks that each arc that terminates at $o(a)$ (apart from $\overline{a}$) has a label with cardinality one. Any arc which does not satisfy this condition can't be part of a scopo but an arc satisfying this condition is not sufficient for it to be part of a scopo.

For each arc, a, in a candidate scopo, the `IsScopo` function checks that each arc that terminates at $o(a)$ is in the candidate scopo and that $\overline{a}$ is not in the candidate scopo. The previous work to reduce the search space ensures that each arc label already has a cardinality of one.

The candidate scopos are all the subsets of the candidate arcs. This is why reducing the search space is necessary. If we did not reduce the search space then the number of candidate scopos would be $2^{|A\Gamma|}$ where Γ is

Algorithm 1: Local Action Diagrams Isomorphism Search

```
Data: lad_1, lad_2
Result: lad_isomorphism
digraph_isomorphisms ← EveryDigraphIsomorphism(lad_1, lad_2);
bijections ← EmptyList();
for iso in digraph_isomorphisms do
    for vertex in LocalActionDiagramVertices(lad_1) do
        group_1 ← LocalActionDiagramVertexLabel(vertex);
        group_2 ← LocalActionDiagramVertexLabel(iso(vertex));
        bijection ← FindGroupConjugateBijection(group_1, group_2, iso);
        if bijection = fail then
            bijections ← EmptyList();
            break;
        end
        else
            Add(bijections, bijection);
        end
    end
    if Length(bijections) = Length(LocalActionDiagramVertices(lad_1)) then
        Return(iso, bijections);
    end
end
Return(fail);
```

Algorithm 2: Local Action Diagram Scopo Search

```
Data: lad
Result: lad_scopos
all_arcs ← LocalActionDiagramArcs(lad);
candidate_arcs ← EmptyList();
lad_scopos ← EmptyList();
for arc in all_arcs do
    if Length(ArcLabel(arc)) = 1 and ScopoSecondCondition(arc) then
        Add(candidate_arcs, arc);
    end
end
for candidate_scopo in PowerSet(candidate_arcs) do
    if IsScopo(candidate_scopo) then
        Add(lad_scopos, candidate_scopo);
    end
end
Return(lad_scopos);
```

the underlying digraph of the local action diagram. This quickly grows to sizes that are impractical to compute and reducing the search space allows this to be computed for more local action diagrams.

6.3 Searching For All Local Action Diagrams

For a fixed degree d, we want to find all (P)-closed groups acting on T_d with a given number of vertex orbits. We are able to do this by finding all local action diagrams up to an isomorphism which correspond to the given conditions. The case for one vertex orbit — i.e. vertex-transitive (P)-closed groups — was implemented by S. Tornier in GAP. [1] This implementation was adapted to use our implementation of a local action diagram category in GAP. Algorithm 3 shows our implementation for the case with two vertex orbits — i.e. local action diagrams with two vertices.

In Algorithm 3, the groups labelling the vertices are representatives of the conjugacy classes of subgroups of $\mathrm{Sym}(d)$. We use unordered tuples of them because in the case with two vertices swapping the order of the group labels produces isomorphic local action diagrams.

The number of arcs originating at each vertex is determined by the number of orbits of the group labelling the vertex. There must be at least one pair of arcs between the two vertices as local action diagrams must be connected. The maximum number of arcs between the two vertices is determined by the minimum number of orbits of the groups labelling the vertices. Any 'left over' arcs must be loops on the vertex they originate at. This process iterates through every underlying digraph for each group labelling the vertices.

Algorithm 3: All Local Action Diagrams Search

```
Data: d
Result: all_lads
all_lads ← EmptyList();
conjugate_group_pairs ← UnorderedTuples(ConjugacyClassesSubgroups(SymmetricGroup(d)));
for group_pair in conjugate_group_pairs do
    arcs ← [Orbits(group_pair[1]), Orbits(group_pair[2])];
    no_orbits ← [Length(arcs[1]), Length(arcs[2])];
    for arcs_between in Range(1..Min(no_orbits)) do
        digraph ←
         Digraph([[no_oribts[1]-arcs_between, arcs_between], [arcs_between, no_orbits[2]-arcs_between]]);
        for arc_label_one in Arrangements(arcs[1]) do
            for arc_label_two in Arrangements(arcs[2]) do
                arc_labels ← Concatenation(arc_label_one, arc_label_two);
                for reversal_map in OrderTwoPermutations(Length(arc_labels)) do
                    if IsValidReversalMap(digraph, reversal_map) then
                        temp_lad ← LocalActionDiagram(digraph, group_pair, arc_labels, reversal_map);
                        for lad in all_lads do
                            if IsIsomorphicLocalActionDiagrams(lad, temp_lad) then
                                Continue(2);
                            end
                        end
                        Add(all_lads, temp_lad);
                    end
                end
            end
        end
    end
end
Return(all_lads);
```

Table 1: Number of Local Action Diagrams With Two Vertices Labelled With Subgroups of T_d

d	Number of Local Action Diagrams
1	1
2	4
3	26
4	200
5	?

The labels for the arcs are also determined by the vertex orbits. Every arrangement of the vertex orbits is iterated through for each underlying digraph. The reversal maps are determined by iterating through all order two permutations on $|A\Gamma|$ elements — i.e. all $r \in \mathrm{Sym}(|A\Gamma|)$ such that $r^2 = \mathrm{id}$ — and checking which are valid reversal maps for the underlying local action diagram. For each local action diagram found, we add it to the output list if it is not isomorphic to any previously found local action diagrams.

7 Searching For All (P)-closed Subgroups of T_d With Two Vertex Orbits

We ran the process described in Algorithm 3 for a number of d. Table 1 shows the number of local action diagrams found up to an isomorphism and thus the number of (P)-closed subgroups of T_d up to a conjugacy.

We were able to run Algorithm 3 for $d = 1, 2, 3, 4$ but it takes too long to run for $d = 5$. To run it on $d = 5$ we will need to improve the search algorithm by reducing the search space. For example, if both vertices are labelled with the trivial group acting on $\{1, 2, 3, 4, 5\}$ then the orbits of both groups are $\{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}\}$. In this case we don't need to search through every possible arrangement because conjugating the trivial group by any bijection results in the trivial group which means that we can always pick a bijection that correctly maps the orbits labelling the arcs. Thus we only need to search through one arrangement, reducing the search space.

It's interesting the note that the sequence $1, 4, 26, 200$ corresponds to first four elements of sequences A271935 and A246509 on the OEIS. [4, A271935, A245609] Both of these sequences differ at the fifth element so it will be interesting to optimize the code and run it for $d = 5$ to see if the number of local action diagrams matches up with one of these sequences.

8 Conclusion

Local action diagrams can be used to describe (P)-closed groups with a finite amount of information. We have implemented a category for local action diagrams in GAP and some functions to perform computations with them, namely: searching for a local action diagram isomorphism, searching for scopos in a local action diagram, and enumerating local action diagrams with one or two vertices.

Possible future directions are:

- Improving the algorithms used in the implementation so faster computations can be performed.
- Running the enumeration algorithm for larger values of d to build a library of local action diagrams.
  - Perhaps only enumerating sub-cases, such as the local action diagrams which are trees.
- Implementing further computations, such as testing if a local action diagram is a local subaction diagram of another.
- Submit the package for redistribution with GAP.

9 Acknowledgements

I would like to thank Stephan Tornier for supervising this project. His help with understating local action diagrams and GAP has been invaluable and his code to draw local action diagrams in TikZ is appreciated.

10 Appendix: Using The GAP Package Produced

The GAP package can be downloaded from `https://github.com/marcusc31415/LocalActionDiagrams`. To install it, the `LocalActionDiagrams` folder needs to be placed in the `pkg` folder of a GAP root directory. A list of GAP root directories can be found in the `GAPInfo.RootPaths` variable in a GAP session.

Once it is installed, to load the package in a GAP session you use `LoadPackage("localactiondiagrams");`. This will also load the digraphs package. To make a local action diagram you need an immutable digraph, a list of groups to label each vertex, a list of sets to label each edge, and a reversal mapping. The function `LocalActionDiagramFromData` will check that these form a valid local action diagram and return the local action diagram; the `LocalActionDiagramFromDataNC` function won't check this and so can be used if you've already checked that the data is valid.

The `IsomorhpismLocalActionDiagrams` function will return an isomorphism between two input local action diagrams or `fail` if one doesn't exist. The `Scopo` attribute of a local action diagram will first calculate all the scopos of a local action diagram and store them; it will use the stored value on subsequent calls. The `AllLocalActionDiagrams` function will enumerate all local action diagrams satisfying given condition. It currently supports only one and two vertices.

11 References

[1] C. D. Reid and S. M. Smith, *Groups acting on trees with tits' independence property (p)*, 2020. DOI: `10.48550/ARXIV.2002.11766`. [Online]. Available: `https://arxiv.org/abs/2002.11766`.

[2] *GAP – Groups, Algorithms, and Programming, Version 4.12.2*, The GAP Group, 2022. [Online]. Available: https://www.gap-system.org.

[3] J. D. Beule, J. Jonušas, J. D. Mitchell, M. Torpey, M. Tsalakou, and W. A. Wilson, *Digraphs - GAP package, version 1.6.1*, Dec. 2022. [Online]. Available: https://digraphs.github.io/Digraphs.

[4] OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences*, Published electronically at http://oeis.org, 2023.