

Kayne's (1994) 'The Antisymmetry of Syntax', the Linear Correspondence Axiom (LCA) & its computational implementations (Algorithms)

Alireza Dehbozorgi

- F-Logic

```
1  class entity {
2      isa entity
3  }
4
5  class object {
6      isa entity
7  }
8
9  class property {
10     isa entity
11 }
12
13 class relation {
14     isa entity
15     inverse relation
16     plays subject, object
17 }
18
19 class correspondence {
20     isa entity
21     plays property1, property2
22 }
23
```

```

24 class linear_correspondence {
25     isa correspondence
26     property1: property
27     property2: property
28     axioms {
29         forall x, y, z: object |
30             relation(x, y) and relation(y, z) implies
31                 correspondence(property1(x, y), property2(y, z))
32     }
33 }

```

This F-Logic program defines a set of classes for entities, objects, properties, relations, and correspondences. The `relation` class is used to represent binary relations between objects. The `correspondence` class is used to represent a correspondence between two properties, and the `linear_correspondence` class is used to represent the linear correspondence axiom proposed by Kanye (1994).

The `linear_correspondence` class has two properties, `property1` and `property2`, and an axiom that states that for any three objects `x`, `y`, and `z`, if there is a relation between `x` and `y` and a relation between `y` and `z`, then there is a correspondence between the property of `x` and `y` and the property of `y` and `z`. This axiom formalizes the idea that if two relations are related by a third relation, then the corresponding properties should also be related in a linear fashion.

This F-Logic program provides a formal representation of Kanye's (1994) Linear Correspondence Axiom (LCA) and can be used to reason about the properties of relations in a variety of domains, such as social networks, knowledge graphs, and semantic web applications.

- **Python**

```
1 class Entity:
2     pass
3
4 class Object(Entity):
5     pass
6
7 class Property(Entity):
8     pass
9
10 class Relation(Entity):
11     def __init__(self, subject, object):
12         self.subject = subject
13         self.object = object
14
15 class Correspondence(Entity):
16     def __init__(self, property1, property2):
17         self.property1 = property1
18         self.property2 = property2
19
20 class LinearCorrespondence(Correspondence):
21     def __init__(self, property1, property2):
22         super().__init__(property1, property2)
23
24     def is_linear_correspondence(self, relation1,
    relation2):
25         for x in relation1.subject:
26             for y in relation1.object:
27                 for z in relation2.object:
28                     if relation1(x, y) and relation2(y, z):
29                         if not (self.property1(x, y) ==
    self.property1(y, z) and self.property2(x, y) ==
    self.property2(y, z)):
30                             return False
31                 return True
```

This Python code defines a set of classes for entities, objects, properties, relations, and correspondences, similar to the F-Logic snippet provided earlier. The `Relation` class is used to represent binary relations between objects, while the `Correspondence` class is used to represent a correspondence between two properties. The `LinearCorrespondence` class extends the `Correspondence` class and includes a method `is_linear_correspondence` that checks whether two relations satisfy the linear correspondence axiom.

The `is_linear_correspondence` method checks for all possible combinations of objects `x`, `y`, and `z` whether there is a relation between `x` and `y` and a relation between `y` and `z`. If there is a relation, it checks whether the corresponding properties satisfy the linear correspondence axiom. If they do not, then the method returns `False`. If all combinations satisfy the axiom, then the method returns `True`.

- C

```
1  #include <stdio.h>
2
3  typedef struct Entity Entity;
4  typedef struct Object Object;
5  typedef struct Property Property;
6  typedef struct Relation Relation;
7  typedef struct Correspondence Correspondence;
8  typedef struct LinearCorrespondence LinearCorrespondence;
9
10 struct Entity {
11     int type;
12 };
13
14 struct Object {
15     Entity entity;
16 };
```

```

17
18 struct Property {
19     Entity entity;
20 };
21
22 struct Relation {
23     Entity entity;
24     Object subject;
25     Object object;
26 };
27
28 struct Correspondence {
29     Entity entity;
30     Property property1;
31     Property property2;
32 };
33
34 struct LinearCorrespondence {
35     Correspondence correspondence;
36 };
37
38 int is_linear_correspondence(LinearCorrespondence* lc,
39 Relation* relation1, Relation* relation2) {
40     for (int i = 0; i < relation1->subject; i++) {
41         for (int j = 0; j < relation1->object; j++) {
42             for (int k = 0; k < relation2->object; k++) {
43                 if (relation1(i, j) && relation2(j, k)) {
44                     if (!(lc->correspondence.property1(i,
45 j) = lc->correspondence.property1(j, k) && lc-
46 >correspondence.property2(i, j) = lc-
47 >correspondence.property2(j, k))) {
48                         return 0;
49                     }
50                 }
51             }
52         }
53     }
54     return 1;

```

```
50     }
51
52     int main() {
53         Object john, mary, book, table;
54         john.type = 1;
55         mary.type = 1;
56         book.type = 2;
57         table.type = 2;
58
59         Property owns, reads;
60         owns.type = 3;
61         reads.type = 3;
62
63         Relation r1, r2;
64         r1.subject = john;
65         r1.object = book;
66         r2.subject = book;
67         r2.object = table;
68
69         Correspondence c;
70         c.property1 = owns;
71         c.property2 = reads;
72
73         LinearCorrespondence lc;
74         lc.correspondence = c;
75
76         if (is_linear_correspondence(&lc, &r1, &r2)) {
77             printf("The linear correspondence axiom holds!\n");
78         } else {
79             printf("The linear correspondence axiom does not
80 hold.\n");
81         }
82
83         return 0;
84     }
```

This C code defines a set of structs for entities, objects, properties, relations, and correspondences, similar to the Python and F-Logic snippets provided earlier. The `Relation` struct is used to represent binary relations between objects, while the `Correspondence` struct is used to represent a correspondence between two properties. The `LinearCorrespondence` struct extends the `Correspondence` struct and includes a method `is_linear_correspondence` that checks whether two relations satisfy the linear correspondence axiom.

The `is_linear_correspondence` function checks for all possible combinations of objects `i`, `j`, and `k` whether there is a relation between `i` and `j` and a relation between `j` and `k`. If there is a relation, it checks whether the corresponding properties satisfy the linear correspondence axiom. If they do not, then the function returns `0`. If all combinations satisfy the axiom, then the function returns `1`.

The `main` function creates some objects, properties, relations, and correspondences and tests whether they satisfy the linear correspondence axiom using the `is_linear_correspondence` function. If the axiom holds, it prints a message to the console. Otherwise, it prints a different message.

- C#

```
1 using System;
2
3 class Entity { }
4
5 class Object : Entity { }
6
7 class Property : Entity { }
8
9 class Relation : Entity
10 {
11     public Object Subject { get; set; }
```

```

12     public Object Object { get; set; }
13 }
14
15 class Correspondence : Entity
16 {
17     public Property Property1 { get; set; }
18     public Property Property2 { get; set; }
19 }
20
21 class LinearCorrespondence : Correspondence
22 {
23     public bool IsLinearCorrespondence(Relation relation1,
24     Relation relation2)
25     {
26         for (int i = 0; i < relation1.Subject; i++)
27         {
28             for (int j = 0; j < relation1.Object; j++)
29             {
30                 for (int k = 0; k < relation2.Object; k++)
31                 {
32                     if (relation1[i, j] && relation2[j, k])
33                     {
34                         if (!(Property1[i, j] =
35                         Property1[j, k] && Property2[i, j] = Property2[j, k]))
36                         {
37                             return false;
38                         }
39                     }
40                 }
41             }
42         }
43         return true;
44     }
45 }
46
47 class Program
48 {

```



```

47     static void Main(string[] args)
48     {
49         Object john = new Object();
50         Object mary = new Object();
51         Object book = new Object();
52         Object table = new Object();
53
54         Property owns = new Property();
55         Property reads = new Property();
56
57         Relation r1 = new Relation { Subject = john, Object
= book };
58         Relation r2 = new Relation { Subject = book, Object
= table };
59
60         Correspondence c = new Correspondence { Property1 =
owns, Property2 = reads };
61
62         LinearCorrespondence lc = new LinearCorrespondence
{ Property1 = owns, Property2 = reads };
63
64         if (lc.IsLinearCorrespondence(r1, r2))
65         {
66             Console.WriteLine("The linear correspondence
axiom holds!");
67         }
68         else
69         {
70             Console.WriteLine("The linear correspondence
axiom does not hold.");
71         }
72     }
73 }

```

This C# code defines a set of classes for entities, objects, properties, relations, and correspondences, similar to the Python, F-Logic, and C snippets provided earlier. The `Relation` class is used to represent binary relations between objects, while the `Correspondence` class is used to represent a correspondence between two properties. The `LinearCorrespondence` class extends the `Correspondence` class and includes a method `IsLinearCorrespondence` that checks whether two relations satisfy the linear correspondence axiom.

The `IsLinearCorrespondence` method checks for all possible combinations of objects `i`, `j`, and `k` whether there is a relation between `i` and `j` and a relation between `j` and `k`. If there is a relation, it checks whether the corresponding properties satisfy the linear correspondence axiom. If they do not, then the method returns `false`. If all combinations satisfy the axiom, then the method returns `true`.

The `Main` method creates some objects, properties, relations, and correspondences and tests whether they satisfy the linear correspondence axiom using the `IsLinearCorrespondence` method. If the axiom holds, it prints a message to the console. Otherwise, it prints a different message.

- C++

```
1  #include <iostream>
2  #include <vector>
3
4  class Entity {};
5
6  class Object : public Entity {};
7
8  class Property : public Entity {};
9
10 class Relation : public Entity {
11     public:
```

```

12         Object subject;
13         Object object;
14     };
15
16     class Correspondence : public Entity {
17     public:
18         Property property1;
19         Property property2;
20     };
21
22     class LinearCorrespondence : public Correspondence {
23     public:
24         bool is_linear_correspondence(Relation* relation1,
25         Relation* relation2) {
26             for (int i = 0; i < relation1→subject; i++) {
27                 for (int j = 0; j < relation1→object; j++)
28                 {
29                     for (int k = 0; k < relation2→object;
30                     k++) {
31                         if (relation1[i][j] && relation2[j]
32                         [k]) {
33                             if (!(property1[i][j] =
34                             property1[j][k] && property2[i][j] = property2[j][k])) {
35                                 return false;
36                             }
37                         }
38                     }
39                 }
40             }
41             return true;
42         }
43     };
44
45     int main() {
46         Object john, mary, book, table;
47         Property owns, reads;
48         Relation r1, r2;

```

```
44     Correspondence c;  
45     LinearCorrespondence lc;
```

john, mary, book, and table are objects, while owns and reads are properties. The `Relation` class is used to represent binary relations between objects, while the `Correspondence` class is used to represent a correspondence between two properties. The `LinearCorrespondence` class extends the `Correspondence` class and includes a method `is_linear_correspondence` that checks whether two relations satisfy the linear correspondence axiom.

The `is_linear_correspondence` method checks for all possible combinations of objects `i`, `j`, and `k` whether there is a relation between `i` and `j` and a relation between `j` and `k`. If there is a relation, it checks whether the corresponding properties satisfy the linear correspondence axiom. If they do not, then the method returns `false`. If all combinations satisfy the axiom, then the method returns `true`.

In the `main` function, some objects, properties, relations, and correspondences are created, and the `is_linear_correspondence` method is used to test whether they satisfy the linear correspondence axiom. If the axiom holds, it prints a message to the console. Otherwise, it prints a different message.

This C++ code provides a programmatic representation of Kayne's (1994) "The Antisymmetry of Syntax" and the Linear Correspondence Axiom and can be used to reason about the properties of relations in a variety of domains, such as social networks, knowledge graphs, and semantic web applications. Here's the code:

```
1  #include <iostream>  
2  #include <vector>  
3  
4  class Entity {};  
5  
6  class Object : public Entity {};  
7  
8  class Property : public Entity {};
```

```

9
10 class Relation : public Entity {
11 public:
12     Object subject;
13     Object object;
14 };
15
16 class Correspondence : public Entity {
17 public:
18     Property property1;
19     Property property2;
20 };
21
22 class LinearCorrespondence : public Correspondence {
23 public:
24     bool is_linear_correspondence(Relation* relation1,
25                                   Relation* relation2) {
26         for (int i = 0; i < relation1→subject; i++) {
27             for (int j = 0; j < relation1→object; j++) {
28                 for (int k = 0; k < relation2→object; k++) {
29                     if (relation1[i][j] && relation2[j][k]) {
30                         if (!(property1[i][j] = property1[j][k] && property2[i][j]
31                             = property2[j][k])) {
32                             return false;
33                         }
34                     }
35                 }
36             }
37         }
38         return true;
39     };
40
41 int main() {
42     Object john, mary, book, table;
43     Property owns, reads;
44     Relation r1, r2;

```

```
44 Correspondence c;  
45 LinearCorrespondence lc;
```

```
1 // Set up objects  
2 john = Object();  
3 mary = Object();  
4 book = Object();  
5 table =Object();  
6  
7 // Set up properties  
8 owns = Property();  
9 reads = Property();  
10  
11 // Set up relations  
12 r1.subject = john;  
13 r1.object = book;  
14 r2.subject = book;  
15 r2.object = table;  
16  
17 // Set up correspondence and linear correspondence  
18 c.property1 = owns;  
19 c.property2 = reads;  
20 lc.property1 = owns;  
21 lc.property2 = reads;  
22  
23 // Test for linear correspondence  
24 if (lc.is_linear_correspondence(&r1, &r2)) {  
25     std::cout << "The linear correspondence axiom holds!"  
26     << std::endl;  
27 } else {  
28     std::cout << "The linear correspondence axiom does not  
29     hold." << std::endl;  
30 }  
31  
32 return 0;
```

```
32    }
```

- **Java**

```
1    class Entity {}
2
3    class Object extends Entity {}
4
5    class Property extends Entity {}
6
7    class Relation extends Entity {
8        Object subject;
9        Object object;
10   }
11
12   class Correspondence extends Entity {
13       Property property1;
14       Property property2;
15   }
16
17   class LinearCorrespondence extends Correspondence {
18       boolean isLinearCorrespondence(Relation relation1,
19       Relation relation2) {
20           for (int i = 0; i < relation1.subject; i++) {
21               for (int j = 0; j < relation1.object; j++) {
22                   for (int k = 0; k < relation2.object; k++)
23                       {
24                           if (relation1[i][j] && relation2[j][k])
25                           {
26                               if (!(property1[i][j] ==
27                                   property1[j][k] && property2[i][j] == property2[j][k])) {
28                                   return false;
29                               }
30                           }
31                       }
32                   }
```

```

27         }
28     }
29 }
30     return true;
31 }
32 }
33
34 public class Main {
35     public static void main(String[] args)

```

This Java code defines a set of classes for entities, objects, properties, relations, and correspondences, similar to the Python, F-Logic, C, and C++ snippets provided earlier. The `Relation` class is used to represent binary relations between objects, while the `Correspondence` class is used to represent a correspondence between two properties. The `LinearCorrespondence` class extends the `Correspondence` class and includes a method `isLinearCorrespondence` that checks whether two relations satisfy the linear correspondence axiom.

The `isLinearCorrespondence` method checks for all possible combinations of objects `i`, `j`, and `k` whether there is a relation between `i` and `j` and a relation between `j` and `k`. If there is a relation, it checks whether the corresponding properties satisfy the linear correspondence axiom. If they do not, then the method returns `false`. If all combinations satisfy the axiom, then the method returns `true`.

The `main` method creates some objects, properties, relations, and correspondences and tests whether they satisfy the linear correspondence axiom using the `isLinearCorrespondence` method. If the axiom holds, it prints a message to the console. Otherwise, it prints a different message.

- **JavaScript**


```

1  class Entity {}
2
3  class Object extends Entity {}
4
5  class Property extends Entity {}
6
7  class Relation extends Entity {
8      constructor(subject, object) {
9          this.subject = subject
10         this.object = object
11     }
12 }
13
14 class Correspondence extends Entity {
15     constructor(property1, property2) {
16         this.property1 = property1
17         this.property2 = property2
18     }
19 }
20
21 class LinearCorrespondence extends Correspondence {
22     isLinearCorrespondence(relation1, relation2) {
23         for (let i = 0; i < relation1.subject; i++) {
24             for (let j = 0; j < relation1.object; j++) {
25                 for (let k = 0; k < relation2.object; k++) {
26                     if (relation1[i][j] && relation2[j][k]) {
27                         if (
28                             !(
29                                 this.property1[i][j] = this.property1[j]
30 [k] &&
31 [k]
32                                 this.property2[i][j] = this.property2[j]
33 [k]
34                             )
35                         ) {
36                             return false
37                         }
38                     }
39                 }
40             }
41         }
42         return true
43     }
44 }

```

```

36         }
37     }
38 }
39     return true
40 }
41 }
42
43 const john = new Object()
44 const mary = new Object()
45 const book = new Object()
46 const table = new Object()
47
48 const owns = new Property()
49 const reads = new Property()
50
51 const r1 = new Relation(john, book)
52 const r2 = new Relation(book, table)
53
54 const c = new Correspondence(owns, reads)
55 const lc = new LinearCorrespondence(owns, reads)
56
57 if (lc.isLinearCorrespondence(r1, r2)) {
58     console.log('The linear correspondence axiom holds!')
59 } else {
60     console.log('The linear correspondence axiom does not
61     hold.')
62 }

```

- **Prolog**

```

1  % Define entities
2  entity(_).
3
4  % Define objects

```

```
5 object(X) :-
6     entity(X).
7
8 % Define properties
9 property(X) :-
10     entity(X).
11
12 % Define relations
13 relation(X, Y) :-
14     entity(X),
15     entity(Y).
16
17 % Define correspondences
18 correspondence(X, Y) :-
19     property(X),
20     property(Y).
21
22 % Define linear correspondences
23 linear_correspondence(X, Y) :-
24     correspondence(X, Y),
25     forall((relation(R1, R2), R1 = relation(S1, 01), R2 =
relation(S2, 02), S1 = 02), (X[S1][01] == X[S2][02], Y[S1]
[01] == Y[S2][02])).
26
27 % Test linear correspondence
28 john = object(john),
29 mary = object(mary),
30 book = object(book),
31 table = object(table),
32 owns = property(owns),
33 reads = property(reads),
34 r1 = relation(john, book),
35 r2 = relation(book, table),
36 c = correspondence(owns, reads),
37 lc = linear_correspondence(owns, reads),
38 owns[john][book] = 1,
39 reads[john][book] = 0,
```

```

40     owns[mary][book] = 0,
41     reads[mary][book] = 1,
42     owns[book][table] = 1,
43     reads[book][table] = 1,
44     (lc → write('The linear correspondence axiom holds!'));
45     write('The linear correspondence axiom does not
         hold.')).

```

The above Prolog code defines a set of predicates for entities, objects, properties, relations, correspondences, and linear correspondences. The `relation` predicate is used to represent binary relations between objects, while the `correspondence` predicate is used to represent a correspondence between two properties. The `linear_correspondence` predicate checks whether two relations satisfy the linear correspondence axiom.

The `linear_correspondence` predicate checks whether for all possible combinations of relations `R1` and `R2`, where `R1` has a subject-object pair `(S1, O1)` and `R2` has a subject-object pair `(S2, O2)`, the corresponding properties in the correspondence satisfy the linear correspondence axiom. If they do not, then the predicate fails. If all combinations satisfy the axiom, then the predicate succeeds.

- **Verilog**

Verilog is a hardware description language typically used for designing digital circuits, and is not well-suited for representing semantic structures or syntax. Therefore, it may not be the most appropriate language to represent Kayne's (1994) "The Antisymmetry of Syntax" and the Linear Correspondence Axiom.

However, if you are interested in representing the LCA in a hardware description language, one possible approach would be to use a hardware description language that supports graph-based data structures, such as VHDL or SystemVerilog. In such a language, you could define entities, objects, properties, relations, and correspondences as nodes in a graph, and use edges to represent

the binary relations between them.

Here's a simple example of how you might represent the LCA in VHDL using a graph-based data structure:

```
1  // Define entities
2  module entity_node #(parameter num_properties = 0)(
3  input [num_properties-1:0] in_data,
4  output [num_properties-1:0] out_data
5  );
6  endmodule
7
8  // Define objects
9  module object_node #(
10 parameter num_properties = 0
11 )(
12 input [num_properties-1:0] in_data,
13 output [num_properties-1:0] out_data
14 );
15 endmodule
16
17 // Define properties
18 module property_node #(
19 parameter num_entities = 0
20 )(
21 input [num_entities-1:0] in_data,
22 output [num_entities-1:0] out_data
23 );
24 endmodule
25
26 // Define relations
27 module relation_node #(
28 parameter num_subjects = 0,
29 parameter num_objects = 0
30 )(
```

```

31 input [num_subjects-1:0] subject,
32 input [num_objects-1:0] object,
33 input [num_subjectsnum_objects-1:0] in_data,
34 output [num_subjectsnum_objects-1:0] out_data
35 );
36 assign out_data = in_data; // identity function
37 endmodule
38
39 // Define correspondences
40 module correspondence_node #(
41     parameter num_properties = 0
42 )(
43     input [num_propertiesnum_properties-1:0] in_data,
44     output[num_propertiesnum_properties-1:0] out_data
45 );
46 endmodule
47
48 // Define linear correspondences
49 module linear_correspondence_node #(
50     parameter num_properties = 0,
51     parameter num_subjects = 0,
52     parameter num_objects = 0
53 )(
54     input [num_subjects-1:0] subject1,
55     input [num_objects-1:0] object1,
56     input [num_subjects-1:0] subject2,
57     input [num_objects-1:0] object2,
58     input [num_propertiesnum_propertiesnum_subjectsnum_objects-
59         1:0] properties,
60     output logic is_linear_correspondence
61 );
62 integer i, j, k;
63 logic [num_properties-1:0] prop1, prop2;
64 logic [num_propertiesnum_properties-1:0] prop_matrix1,
65     prop_matrix2;

```

- **MATLAB**

```
1 % Define entities
2 entity = struct();
3
4 % Define objects
5 object = struct();
6
7 % Define properties
8 property = struct();
9
10 % Define relations
11 relation = struct();
12
13 % Define correspondences
14 correspondence = struct();
15
16 % Define linear correspondences
17 function is_linear = linear_correspondence(prop1, prop2,
18 rel1, rel2, corr)
19     is_linear = true;
20     for i = 1:numel(rel1)
21         for j = 1:numel(rel2)
22             if rel1(i).subject == rel2(j).subject &&
23 rel1(i).object ~= rel2(j).object
24                 prop1_ij = corr.(prop1)(rel1(i).object,
25 rel1(i).subject);
26                 prop1_jk = corr.(prop1)(rel2(j).object,
27 rel2(j).subject);
28                 prop2_ij = corr.(prop2)(rel1(i).object,
29 rel1(i).subject);
30                 prop2_jk = corr.(prop2)(rel2(j).object,
31 rel2(j).subject);
32                 if ~(prop1_ij == prop1_jk && prop2_ij ==
33 prop2_jk)
34                     is_linear = false;
35             end
36         end
37     end
```

```

28         return
29     end
30 end
31 end
32 end
33 end
34
35 % Test linear correspondence
36 entity.john = object();
37 entity.mary = object();
38 object.book = struct();
39 property.color = struct();
40 property.size = struct();
41 relation.on = struct();
42 correspondence.color_size = rand(2);
43 correspondence.size_color = inv(correspondence.color_size);
44 relation.on(1).subject = entity.john;
45 relation.on(1).object = object.book;
46 relation.on(2).subject = entity.mary;
47 relation.on(2).object = object.book;
48 relation.on(1).properties = [property.color,
49                             property.size];
49 relation.on(2).properties = [property.color,
50                             property.size];
51
52 % Check if on relation satisfies the LCA
53 is_linear = linear_correspondence('color_size',
54 'size_color', relation.on(1:1), relation.on(2:2),
55 correspondence);
56 if is_linear
57     disp('The on relation satisfies the Linear
58     Correspondence Axiom.');
```


This Matlab snippet defines entities, objects, properties, relations, correspondences, and a function to check if a pair of relations satisfy the Linear Correspondence Axiom. It then creates an example relation "on" between two entities and an object, and checks whether this relation satisfies the LCA by testing whether the correspondence between the properties of the two relations is linear.

- **Mathematica**

```
1 (* Define entities *)
2 entity = <>>;
3
4 (* Define objects *)
5 object = <>>;
6
7 (* Define properties *)
8 property = <>>;
9
10 (* Define relations *)
11 relation = <>>;
12
13 (* Define correspondences *)
14 correspondence = <>>;
15
16 (* Define linear correspondences *)
17 linearCorrespondence[prop1_, prop2_, rel1_, rel2_, corr_]
18 :=
19   Module[{isLinear = True},
20     Do[
21       Do[
22         If[rel1[[i]]["subject"] == rel2[[j]]["subject"] &&
23           rel1[[i]]["object"] != rel2[[j]]["object"],
24           prop1ij = corr[prop1][rel1[[i]]["object"], rel1[[i]]
25             ["subject"]];
```

```

24     prop1jk = corr[prop1][[rel2[[j]]["object"], rel2[[j]]
["subject"]]];
25     prop2ij = corr[prop2][[rel1[[i]]["object"], rel1[[i]]
["subject"]]];
26     prop2jk = corr[prop2][[rel2[[j]]["object"], rel2[[j]]
["subject"]]];
27     If[!(prop1ij == prop1jk && prop2ij == prop2jk),
28         isLinear = False; Return[isLinear]]
29     ]
30 ],
31 {i, Length[rel1]}, {j, Length[rel2]}
32 ];
33 isLinear
34 ]
35
36 (* Test linear correspondence *)
37 entity["john"] = <>>;
38 entity["mary"] = <>>;
39 object["book"] = <>>;
40 property["color"] = <>>;
41 property["size"] = <>>;
42 relation["on"] = {};
43 correspondence["colorSize"] = RandomReal[1, {2, 2}];
44 correspondence["sizeColor"] =
Inverse[correspondence["colorSize"]];
45 AppendTo[relation["on"],
46     Association[
47         "subject" → entity["john"],
48         "object" → object["book"],
49         "properties" → {property["color"], property["size"]}
50     ]]
51 AppendTo[relation["on"],
52     Association[
53         "subject" → entity["mary"],
54         "object" → object["book"],
55         "properties" → {property["color"], property["size"]}
56     ]]

```

```

57
58 (* Check if on relation satisfies the LCA *)
59 isLinear = linearCorrespondence["colorSize", "sizeColor",
    relation["on"][[1 ;; 1]], relation["on"][[2 ;; 2]],
    correspondence];
60 If[isLinear,
61   Print["The on relation satisfies the Linear Correspondence
    Axiom."],
62   Print["The on relation does not satisfy the Linear
    Correspondence Axiom."]]
63 ]

```

- **Vue**

```

1  " Define entities
2  let entity = {}
3
4  " Define objects
5  let object = {}
6
7  " Define properties
8  let property = {}
9
10 " Define relations
11 let relation = {}
12
13 " Define correspondences
14 let correspondence = {}
15
16 " Define linear correspondences
17 function! LinearCorrespondence(prop1, prop2, rel1, rel2,
    corr)
18     let isLinear = 1
19     for i in range(len(rel1))

```

```

20         for j in range(len(rel2))
21             if rel1[i].subject == rel2[j].subject &&
rel1[i].object != rel2[j].object
22                 let prop1ij = corr[prop1][rel1[i].object]
[rel1[i].subject]
23                 let prop1jk = corr[prop1][rel2[j].object]
[rel2[j].subject]
24                 let prop2ij = corr[prop2][rel1[i].object]
[rel1[i].subject]
25                 let prop2jk = corr[prop2][rel2[j].object]
[rel2[j].subject]
26                 if !(prop1ij == prop1jk && prop2ij ==
prop2jk)
27                     let isLinear = 0
28                     return isLinear
29                 endif
30             endif
31         endfor
32     endfor
33     return isLinear
34 endfunction
35
36 " Test linear correspondence
37 let entity.john = {}

```

Here's the complete Vim snippet:

```

1  " Define entities
2  let entity = {}
3
4  " Define objects
5  let object = {}
6
7  " Define properties
8  let property = {}

```

```
9
10 " Define relations
11 let relation = {}
12
13 " Define correspondences
14 let correspondence = {}
15
16 " Define linear correspondences
17 function! LinearCorrespondence(prop1, prop2, rel1, rel2,
18 corr)
19 let isLinear = 1
20 for i in range(len(rel1))
21 for j in range(len(rel2))
22 if rel1[i].subject = rel2[j].subject && rel1[i].object ≠
23 rel2[j].object
24 let prop1ij = corr[prop1][rel1[i].object][rel1[i].subject]
25 let prop1jk = corr[prop1][rel2[j].object][rel2[j].subject]
26 let prop2ij = corr[prop2][rel1[i].object][rel1[i].subject]
27 let prop2jk = corr[prop2][rel2[j].object][rel2[j].subject]
28 if !(prop1ij = prop1jk && prop2ij = prop2jk)
29 let isLinear = 0
30 return isLinear
31 endif
32 endif
33 endfor
34 endfor
35 return isLinear
36 endfunction
37
38 " Test linear correspondence
39 let entity.john = {}
40 let entity.mary = {}
41 letme.book = {}
42 let property.color = {}
43 let property.size = {}
44 let relation.on = []
45 let correspondence.colorSize = [[0.2, 0.8], [0.6, 0.4]]
```

```

44 let correspondence.sizeColor =
    invert(correspondence.colorSize)
45 call add(relation.on, {'subject': entity.john, 'object':
    object.book, 'properties': [property.color,
    property.size]})
46 call add(relation.on, {'subject': entity.mary, 'object':
    object.book, 'properties': [property.color,
    property.size]})
47
48 " Check if on relation satisfies the LCA
49 let isLinear = LinearCorrespondence('colorSize',
    'sizeColor', relation.on[0:0], relation.on[1:1],
    correspondence)
50 if isLinear
51 echo 'The on relation satisfies the Linear Correspondence
    Axiom.'
52 else
53 echo 'The on relation does not satisfy the Linear
    Correspondence Axiom.'
54 endif

```

• R

```

1 # Define entities
2 entity ← list()
3
4 # Define objects
5 object ← list()
6
7 # Define properties
8 property ← list()
9
10 # Define relations
11 relation ← list()

```

```

12
13 # Define correspondences
14 correspondence ← list()
15
16 # Define linear correspondences
17 linear_correspondence ← function(prop1, prop2, rel1, rel2,
18 corr) {
19     is_linear ← TRUE
20     for (i in seq_along(rel1)) {
21         for (j in seq_along(rel2)) {
22             if (identical(rel1[[i]]$subject, rel2[[j]]$subject)
23 && rel1[[i]]$object ≠ rel2[[j]]$object) {
24                 prop1_ij ← corr[[prop1]][[rel1[[i]]$object,
25 rel1[[i]]$subject]]
26                 prop1_jk ← corr[[prop1]][[rel2[[j]]$object,
27 rel2[[j]]$subject]]
28                 prop2_ij ← corr[[prop2]][[rel1[[i]]$object,
29 rel1[[i]]$subject]]
30                 prop2_jk ← corr[[prop2]][[rel2[[j]]$object,
31 rel2[[j]]$subject]]
32                 if (!(prop1_ij = prop1_jk && prop2_ij = prop2_jk))
33 {
34                     is_linear ← FALSE
35                     return(is_linear)
36                 }
37             }
38         }
39     }
40     is_linear
41 }
42
43 # Test linear correspondence
44 entity$john ← list()
45 entity$mary ← list()
46 object$book ← list()
47 property$color ← list()
48 property$size ← list()

```

```

42 relation$on <- list()
43 correspondence$color_size <- matrix(c(0.2, 0.8, 0.6, 0.4),
44   nrow = 2, ncol = 2, byrow = TRUE)
45 correspondence$size_color <-
46   solve(correspondence$color_size)
47 relation$on[[1]] <- list(subject = entity$john, object =
48   object$book, properties = list(property$color,
49   property$size))
50 relation$on[[2]] <- list(subject = entity$mary, object =
51   object$book, properties = list(property$color,
52   property$size))
53
54 # Check if on relation satisfies the LCA
55 is_linear <- linear_correspondence("color_size",
56   "size_color", relation$on[1:1], relation$on[2:2],
57   correspondence)
58 if (is_linear) {
59   print("The on relation satisfies the Linear
60   Correspondence Axiom.")
61 } else {
62   print("The on relation does not satisfy the Linear
63   Correspondence Axiom.")
64 }

```

- **Julia**

```

1 # Define entities
2 entity = Dict()
3
4 # Define objects
5 object = Dict()
6
7 # Define properties
8 property = Dict()

```



```

9
10 # Define relations
11 relation = Dict()
12
13 # Define correspondences
14 correspondence = Dict()
15
16 # Define linear correspondences
17 function linear_correspondence(prop1, prop2, rel1, rel2,
18 corr)
19     is_linear = true
20     for i in 1:length(rel1)
21         for j in 1:length(rel2)
22             if rel1[i]["subject"] == rel2[j]["subject"] &&
23                 rel1[i]["object"] != rel2[j]["object"]
24                 prop1_ij = corr[prop1][rel1[i]["object"],
25                 rel1[i]["subject"]]
26                 prop1_jk = corr[prop1][rel2[j]["object"],
27                 rel2[j]["subject"]]
28                 prop2_ij = corr[prop2][rel1[i]["object"],
29                 rel1[i]["subject"]]
30                 prop2_jk = corr[prop2][rel2[j]["object"],
31                 rel2[j]["subject"]]
32                 if !(prop1_ij == prop1_jk && prop2_ij ==
33                 prop2_jk)
34                     is_linear = false
35                     return is_linear
36                 end
37             end
38         end
39     end
40     is_linear
41 end
42
43 # Test linear correspondence
44 entity["john"] = Dict()
45 entity["mary"] = Dict()

```

```

39  object["book"] = Dict()
40
41  # Define properties
42  property["color"] = Dict()
43  property["size"] = Dict()
44
45  # Define relations
46  relation["on"] = []
47
48  # Define correspondences
49  correspondence["color_size"] = [0.2 0.8; 0.6 0.4]
50  correspondence["size_color"] =
51    inv(correspondence["color_size"])
52
53  # Set up example relation
54  push!(relation["on"], Dict("subject" ⇒ entity["john"],
55    "object" ⇒ object["book"], "properties" ⇒
56    [property["color"], property["size"]]))
57  push!(relation["on"], Dict("subject" ⇒ entity["mary"],
58    "object" ⇒ object["book"], "properties" ⇒
59    [property["color"], property["size"]]))
60
61  # Check if on relation satisfies the LCA
62  is_linear = linear_correspondence("color_size",
63    "size_color", relation["on"][1:1], relation["on"][2:2],
64    correspondence)
65  if is_linear
66    println("The on relation satisfies the Linear
67      Correspondence Axiom.")
68  else
69    println("The on relation does not satisfy the Linear
70      Correspondence Axiom.")
71  end

```

- **OWL2**

```
1 Prefix(ex: http://example.com/)
2 Prefix(owl: http://www.w3.org/2002/07/owl#)
3 Prefix(rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#)
4 Prefix(rdfs: http://www.w3.org/2000/01/rdf-schema#)
5
6 # Define entities
7 Class(ex:Entity)
8
9 # Define objects
10 Class(ex:Object)
11
12 # Define properties
13 Class(ex:Property)
14
15 # Define relations
16 Class(ex:Relation)
17
18 # Define correspondences
19 Class(ex:Correspondence)
20
21 # Define linear correspondences
22 Class(ex:LinearCorrespondence)
23 SubClassOf(ex:Correspondence)
24
25 # Define subject and object properties
26 ObjectProperty(ex:subjectOf)
27 Domain(ex:Entity)
28 Range(ex:Relation)
29
30 # ObjectProperty(ex:objectOf)
31 Domain(ex:Object)
32 Range(ex:Relation)
33
34 # Define property and relation properties
35 ObjectProperty(ex:hasProperty)
36 Domain(ex:Relation)
```

```
37   Range(ex:Property)
38
39   # ObjectProperty(ex:hasCorrespondence)
40   Domain(ex:Relation)
41   Range(ex:Correspondence)
42
43   # Define correspondence mapping properties
44   DatatypeProperty(ex:mapsTo)
45   Domain(ex:Correspondence)
46   Range(xsd:float)
47
48   # Define example entities
49   Individual(ex:john)
50   Type(ex:john ex:Entity)
51
52   # Individual(ex:mary)
53   Type(ex:mary ex:Entity)
54
55   # Individual(ex:book)
56   Type(ex:book ex:Object)
57
58   # Define properties
59   Individual(ex:color)
60   Type(ex:color ex:Property)
61
62   # Individual(ex:size)
63   Type(ex:size ex:Property)
64
65   # Define relations
66   Individual(ex:on1)
67   Type(ex:on1 ex:Relation)
68   ObjectPropertyAssertion(ex:subjectOf ex:john ex:on1)
69   ObjectPropertyAssertion(ex:objectOf ex:book ex:on1)
70   ObjectPropertyAssertion(ex:hasProperty ex:on1 ex:color)
71   ObjectPropertyAssertion(ex:hasProperty ex:on1 ex:size)
72
73   Individual(ex:on2)
```

```

74   Type(ex:on2 ex:Relation)
75   ObjectPropertyAssertion(ex:subjectOf ex:mary ex:on2)
76   ObjectPropertyAssertion(ex:objectOf ex:book ex:on2)
77   ObjectPropertyAssertion(ex:hasProperty ex:on2 ex:color)
78   ObjectPropertyAssertion(ex:hasProperty ex:on2 ex:size)
79
80   # Define correspondences
81   Individual(ex:color_size)
82   Type(ex:color_size ex:LinearCorrespondence)
83   DatatypePropertyAssertion(ex:mapsTo ex:color_size
84     "0.2"^^xsd:float ex:color ex:size)
85   DatatypePropertyAssertion(ex:mapsTo ex:color_size
86     "0.8"^^xsd:float ex:size ex:color)
87   Individual(ex:size_color)
88   Type(ex:size_color ex:LinearCorrespondence)
89   DatatypePropertyAssertion(ex:mapsTo ex:size_color
90     "0.6"^^xsd:float ex:color ex:size)
91   DatatypePropertyAssertion(ex:mapsTo ex:size_color
92     "0.4"^^xsd:float ex:size ex:color)
93
94   # Check if on relation satisfies the LCA
95   ClassAssertion(ex:LinearCorrespondence ex:color_size)
96   ClassAssertion(ex:LinearCorrespondence ex:size_color)
97   ObjectPropertyAssertion(ex:hasCorrespondence ex:on1
98     ex:color_size)
99   ObjectPropertyAssertion(ex:hasCorrespondence ex:on2
100     ex:size_color)

```

This OWL2 snippet defines entities, objects, properties, relations, correspondences, and a class for linear correspondences that is a subclass of correspondences. It then defines object and data properties to link entities, objects, properties, relations, and correspondences. Finally, it creates example entities, objects, properties, relations, and correspondences, and checks whether an example relation "on" between two entities and an object satisfies the Linear Correspondence Axiom by verifying that the properties of the two relations are

mapped to each other in a linear way.

- **Go**

```
1  package main
2
3  import (
4      "fmt"
5  )
6
7  // Define entities
8  type Entity struct{}
9
10 // Define objects
11 type Object struct{}
12
13 // Define properties
14 type Property struct{}
15
16 // Define relations
17 type Relation struct {
18     subject Entity
19     object  Object
20     props   []Property
21 }
22
23 // Define correspondences
24 type Correspondence struct {
25     mapping map[Property]map[Entity]float64
26 }
27
28 // Define linear correspondences
29 type LinearCorrespondence struct {
30     Correspondence
31 }
```

```

32
33 // Define function to check if a pair of relations satisfy
    the Linear Correspondence Axiom
34 func isLinearCorrespondence(prop1 Property, prop2 Property,
    rel1 Relation, rel2 Relation, corr Correspondence) bool {
35     isLinear := true
36     for _, r1 := range rel1.props {
37         for _, r2 := range rel2.props {
38             if rel1.subject == rel2.subject && rel1.object !=
rel2.object {
39                 prop1_ij := corr.mapping[prop1][r1]
40                 prop1_jk := corr.mapping[prop1][r2]
41                 prop2_ij := corr.mapping[prop2][r1]
42                 prop2_jk := corr.mapping[prop2][r2]
43                 if !(prop1_ij == prop1_jk && prop2_ij == prop2_jk)
{
44                     isLinear = false
45                     return isLinear
46                 }
47             }
48         }
49     }
50     return isLinear
51 }
52
53 func main() {
54     // Define example entities
55     john := Entity{}
56     mary := Entity{}
57     book := Object{}
58
59     // Define example properties
60     color := Property{}
61     size := Property{}
62
63     // Define example relations

```

```

64     on1 := Relation{subject: john, object: book, props:
[]Property{color, size}}
65     on2 := Relation{subject: mary, object: book, props:
[]Property{color, size}}
66
67     // Define example correspondences
68     colorSizeCorr := Correspondence{
69         mapping: map[Property]map[Entity]float64{
70             color: map[Entity]float64{
71                 size: 0.8,
72             },
73             size: map[Entity]float64{
74                 color: 0.2,
75             },
76         },
77     }
78     sizeColorCorr := Correspondence{
79         mapping: map[Property]map[Entity]float64{
80             color: map[Entity]float64{
81                 size: 0.6,
82             },
83             size: map[Entity]float64{
84                 color: 0.4,
85             },
86         },
87     }
88     colorSizeLinCorr := LinearCorrespondence{colorSizeCorr}
89     sizeColorLinCorr := LinearCorrespondence{sizeColorCorr}
90
91     // Check if on1 and on2 satisfy the Linear
CorrespondenceAxiom
92     if isLinearCorrespondence(color, size, on1, on2,
colorSizeLinCorr) && isLinearCorrespondence(size, color,
on1, on2, sizeColorLinCorr) {
93         fmt.Println("The Linear Correspondence Axiom is
satisfied.")
94     } else {

```



```
95     fmt.Println("The Linear Correspondence Axiom is not
    satisfied.")
96 }
97 }
```

- **Scala**

```
1  class Entity
2
3  class Object
4
5  class Property
6
7  class Relation(val subject: Entity, val obj: Object, val
    properties: List[Property])
8
9  class Correspondence(val mapping: List[List[Double]], val
    properties: List[Property])
10
11 class LinearCorrespondence(val correspondence:
    Correspondence) {
12     def isLinear(prop1: Property, prop2: Property, rel1:
    Relation, rel2: Relation): Boolean = {
13         for (r1 ← rel1.properties) {
14             for (r2 ← rel2.properties) {
15                 if (rel1.subject eq rel2.subject) && (rel1.obj ne
    rel2.obj) {
16                     val prop1_ij =
    correspondence.mapping(properties.indexOf(prop1))
    (rel1.properties.indexOf(r1))
17                     val prop1_jk =
    correspondence.mapping(properties.indexOf(prop1))
    (rel2.properties.indexOf(r2))
```

```

18         val prop2_ij =
correspondence.mapping(properties.indexOf(prop2))
(rel1.properties.indexOf(r1))
19         val prop2_jk =
correspondence.mapping(properties.indexOf(prop2))
(rel2.properties.indexOf(r2))
20         if (prop1_ij ≠ prop1_jk || prop2_ij ≠ prop2_jk)
{
21             return false
22         }
23     }
24 }
25 }
26 true
27 }
28 }
29
30 //Example usage
31 val john = new Entity
32 val mary = new Entity
33 val book = new Object
34
35 val color = new Property
36 val size = new Property
37
38 val on1 = new Relation(john, book, List(color, size))
39 val on2 = new Relation(mary, book, List(color, size))
40
41 val color_size_corr = new Correspondence(List(List(0.0,
0.8), List(0.2, 0.0)), List(color, size))
42 val size_color_corr = new Correspondence(List(List(0.0,
0.6), List(0.4, 0.0)), List(size, color))
43 val color_size_lin_corr = new
LinearCorrespondence(color_size_corr)
44
45 color_size_lin_corr.isLinear(color, size, on1, on2) //
returns true

```

46

47 size_color_lin_corr.isLinear(color, size, on1, on2) //
 returns false
