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

Alireza Dehbozorgi

- F-Logic

```
class entity {
  isa entity
}

class object {
  isa entity
}

class property {
  isa entity
}

class relation {
  isa entity
  inverse relation
  plays subject, object
}

class correspondence {
  isa entity
  plays property1, property2
}

```

```
class linear_correspondence {
  isa correspondence
  property1: property
  property2: property
  axioms {
    forall x, y, z: object |
      relation(x, y) and relation(y, z) implies
        correspondence(property1(x, y), property2(y, z))
  }
}
```

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 betweenthe 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**

```python
class Entity:
    pass

class Object(Entity):
    pass

class Property(Entity):
    pass

class Relation(Entity):
    def __init__(self, subject, object):
        self.subject = subject
        self.object = object

class Correspondence(Entity):
    def __init__(self, property1, property2):
        self.property1 = property1
        self.property2 = property2

class LinearCorrespondence(Correspondence):
    def __init__(self, property1, property2):
        super().__init__(property1, property2)

    def is_linear_correspondence(self, relation1, relation2):
        for x in relation1.subject:
            for y in relation1.object:
                for z in relation2.object:
                    if relation1(x, y) and relation2(y, z):
                        if not (self.property1(x, y) == self.property1(y, z) and self.property2(x, y) == self.property2(y, z)):
                            return False
        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

```c
#include <stdio.h>

typedef struct Entity Entity;
typedef struct Object Object;
typedef struct Property Property;
typedef struct Relation Relation;
typedef struct Correspondence Correspondence;
typedef struct LinearCorrespondence LinearCorrespondence;

struct Entity {
    int type;
};

struct Object {
    Entity entity;
};
```

```c

struct Property {
    Entity entity;
};

struct Relation {
    Entity entity;
    Object subject;
    Object object;
};

struct Correspondence {
    Entity entity;
    Property property1;
    Property property2;
};

struct LinearCorrespondence {
    Correspondence correspondence;
};

int is_linear_correspondence(LinearCorrespondence* lc, Relation* relation1, Relation* relation2) {
    for (int i = 0; i < relation1->subject; i++) {
        for (int j = 0; j < relation1->object; j++) {
            for (int k = 0; k < relation2->object; k++) {
                if (relation1(i, j) && relation2(j, k)) {
                    if (!(lc->correspondence.property1(i, j) == lc->correspondence.property1(j, k) && lc->correspondence.property2(i, j) == lc->correspondence.property2(j, k))) {
                    return 0;
                }
            }
        }
    }
    return 1;
}

int main() {
    Object john, mary, book, table;
    john.type = 1;
    mary.type = 1;
    book.type = 2;
    table.type = 2;

    Property owns, reads;
    owns.type = 3;
    reads.type = 3;

    Relation r1, r2;
    r1.subject = john;
    r1.object = book;
    r2.subject = book;
    r2.object = table;

    Correspondence c;
    c.property1 = owns;
    c.property2 = reads;

    LinearCorrespondence lc;
    lc.correspondence = c;

    if (is_linear_correspondence(&lc, &r1, &r2)) {
        printf("The linear correspondence axiom holds!\n");
    } else {
        printf("The linear correspondence axiom does not
hold.\n");
    }

    return 0;
}
```

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#**

```
using System;

class Entity { }

class Object : Entity { }

class Property : Entity { }

class Relation : Entity
{
    public Object Subject { get; set; }
    public Object Object { get; set; }
}

class Correspondence : Entity
{
    public Property Property1 { get; set; }
    public Property Property2 { get; set; }
}

class LinearCorrespondence : Correspondence
{
    public bool IsLinearCorrespondence(Relation relation1, Relation relation2)
    {
        for (int i = 0; i < relation1.Subject; i++)
        {
            for (int j = 0; j < relation1.Object; j++)
            {
                for (int k = 0; k < relation2.Object; k++)
                {
                    if (relation1[i, j] && relation2[j, k])
                    {
                        if (!(Property1[i, j] == Property1[j, k] && Property2[i, j] == Property2[j, k]))
                        {
                            return false;
                        }
                    }
                }
            }
        }
        return true;
    }
}

class Program
{
```

```csharp
    static void Main(string[] args)
    {
        Object john = new Object();
        Object mary = new Object();
       Object book = new Object();
        Object table = new Object();

        Property owns = new Property();
        Property reads = new Property();

        Relation r1 = new Relation { Subject = john, Object = book };
        Relation r2 = new Relation { Subject = book, Object = table };

        Correspondence c = new Correspondence { Property1 = owns, Property2 = reads };

        LinearCorrespondence lc = new LinearCorrespondence { Property1 = owns, Property2 = reads };

        if (lc.IsLinearCorrespondence(r1, r2))
        {
            Console.WriteLine("The linear correspondence axiom holds!");
        }
        else
        {
            Console.WriteLine("The linear correspondence axiom does not hold.");
        }
    }
}
```

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++**

```cpp
#include <iostream>
#include <vector>

class Entity {};

class Object : public Entity {};

class Property : public Entity {};

class Relation : public Entity {
    public:
        Object subject;
        Object object;
};

class Correspondence : public Entity {
    public:
        Property property1;
        Property property2;
};

class LinearCorrespondence : public Correspondence {
    public:
        bool is_linear_correspondence(Relation* relation1, Relation* relation2) {
            for (int i = 0; i < relation1->subject; i++) {
                for (int j = 0; j < relation1->object; j++) {
                    for (int k = 0; k < relation2->object; k++) {
                        if (relation1[i][j] && relation2[j][k]) {
                            if (!(property1[i][j] == property1[j][k] && property2[i][j] == property2[j][k])) {
                                return false;
                            }
                        }
                    }
                }
            }
            return true;
        }
};

int main() {
    Object john, mary, book, table;
    Property owns, reads;
    Relation r1, r2;
```

```
        Correspondence c;
        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:

```
#include <iostream>
#include <vector>

class Entity {};

class Object : public Entity {};

class Property : public Entity {};
```

```cpp

class Relation : public Entity {
public:
Object subject;
Object object;
};

class Correspondence : public Entity {
public:
Property property1;
Property property2;
};

class LinearCorrespondence : public Correspondence {
public:
bool is_linear_correspondence(Relation* relation1, Relation* relation2) {
for (int i = 0; i < relation1->subject; i++) {
for (int j = 0; j < relation1->object; j++) {
for (int k = 0; k < relation2->object; k++) {
if (relation1[i][j] && relation2[j][k]) {
if (!(property1[i][j] == property1[j][k] && property2[i][j] == property2[j][k])) {
return false;
}
}
}
}
}
return true;
}
};

int main() {
Object john, mary, book, table;
Property owns, reads;
Relation r1, r2;
```

```
Correspondence c;
LinearCorrespondence lc;
```

```
// Set up objects
john = Object();
mary = Object();
book = Object();
table =Object();

// Set up properties
owns = Property();
reads = Property();

// Set up relations
r1.subject = john;
r1.object = book;
r2.subject = book;
r2.object = table;

// Set up correspondence and linear correspondence
c.property1 = owns;
c.property2 = reads;
lc.property1 = owns;
lc.property2 = reads;

// Test for linear correspondence
if (lc.is_linear_correspondence(&r1, &r2)) {
    std::cout << "The linear correspondence axiom holds!"
<< std::endl;
} else {
    std::cout << "The linear correspondence axiom does not
hold." << std::endl;
}

return 0;

```

```
}
```

- **Java**

```java
class Entity {}

class Object extends Entity {}

class Property extends Entity {}

class Relation extends Entity {
    Object subject;
    Object object;
}

class Correspondence extends Entity {
    Property property1;
    Property property2;
}

class LinearCorrespondence extends Correspondence {
    boolean isLinearCorrespondence(Relation relation1, Relation relation2) {
        for (int i = 0; i < relation1.subject; i++) {
            for (int j = 0; j < relation1.object; j++) {
                for (int k = 0; k < relation2.object; k++) {
                    if (relation1[i][j] && relation2[j][k]) {
                        if (!(property1[i][j] == property1[j][k] && property2[i][j] == property2[j][k])) {
                            return false;
                        }
                    }
                }
            }
        }
        return true;
    }
}

public class Main {
    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 thecorresponding 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**

```
class Entity {}

class Object extends Entity {}

class Property extends Entity {}

class Relation extends Entity {
  constructor(subject, object) {
    this.subject = subject
    this.object = object
  }
}

class Correspondence extends Entity {
  constructor(property1, property2) {
    this.property1 = property1
    this.property2 = property2
  }
}

class LinearCorrespondence extends Correspondence {
  isLinearCorrespondence(relation1, relation2) {
    for (let i = 0; i < relation1.subject; i++) {
      for (let j = 0; j < relation1.object; j++) {
        for (let k = 0; k < relation2.object; k++) {
          if (relation1[i][j] && relation2[j][k]) {
            if (
              !(
                this.property1[i][j] == this.property1[j][k] &&
                this.property2[i][j] == this.property2[j][k]
              )
            ) {
              return false
            }
          }
        }
      }
    }
    return true
  }
}

const john = new Object()
const mary = new Object()
const book = new Object()
const table = new Object()

const owns = new Property()
constreads = new Property()

const r1 = new Relation(john, book)
const r2 = new Relation(book, table)

const c = new Correspondence(owns, reads)
const lc = new LinearCorrespondence(owns, reads)

if (lc.isLinearCorrespondence(r1, r2)) {
  console.log('The linear correspondence axiom holds!')
} else {
  console.log('The linear correspondence axiom does not hold.')
}
```

- **Prolog**

```
% Define entities
entity(_).

% Define objects
```

```
object(X) :-
    entity(X).

% Define properties
property(X) :-
    entity(X).

% Define relations
relation(X, Y) :-
    entity(X),
    entity(Y).

% Define correspondences
correspondence(X, Y) :-
    property(X),
    property(Y).

% Define linear correspondences
linear_correspondence(X, Y) :-
    correspondence(X, Y),
    forall((relation(R1, R2), R1 = relation(S1, O1), R2 = relation(S2, O2), S1 = O2), (X[S1][O1] =:= X[S2][O2], Y[S1][O1] =:= Y[S2][O2])).

% Test linear correspondence
john = object(john),
mary = object(mary),
book = object(book),
table = object(table),
owns = property(owns),
reads = property(reads),
r1 = relation(john, book),
r2 = relation(book, table),
c = correspondence(owns, reads),
lc = linear_correspondence(owns, reads),
owns[john][book] = 1,
reads[john][book] = 0,
owns[mary][book] =0,
reads[mary][book] = 1,
owns[book][table] = 1,
reads[book][table] = 1,
(lc → write('The linear correspondence axiom holds!');
    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:

```
// Define entities
module entity_node #(parameter num_properties = 0)(
input [num_properties-1:0] in_data,
output [num_properties-1:0] out_data
);
endmodule

// Define objects
module object_node #(
parameter num_properties = 0
)(
input [num_properties-1:0] in_data,
output [num_properties-1:0] out_data
);
endmodule

// Define properties
module property_node #(
parameter num_entities = 0
)(
input [num_entities-1:0] in_data,
output [num_entities-1:0] out_data
);
endmodule

// Define relations
module relation_node #(
parameter num_subjects = 0,
parameter num_objects = 0
)(
```

```verilog
input [num_subjects-1:0] subject,
input [num_objects-1:0] object,
input [num_subjectsnum_objects-1:0] in_data,
output [num_subjectsnum_objects-1:0] out_data
);
assign out_data = in_data; // identity function
endmodule

// Define correspondences
module correspondence_node #(
parameter num_properties = 0
)(
input [num_propertiesnum_properties-1:0] in_data,
output[num_propertiesnum_properties-1:0] out_data
);
endmodule

// Define linear correspondences
module linear_correspondence_node #(
parameter num_properties = 0,
parameter num_subjects = 0,
parameter num_objects = 0
)(
input [num_subjects-1:0] subject1,
input [num_objects-1:0] object1,
input [num_subjects-1:0] subject2,
input [num_objects-1:0] object2,
input [num_propertiesnum_propertiesnum_subjectsnum_objects-1:0] properties,
output logic is_linear_correspondence
);
integer i, j, k;
logic [num_properties-1:0] prop1, prop2;
logic [num_propertiesnum_properties-1:0] prop_matrix1, prop_matrix2;
```

- **MATLAB**

```matlab
% Define entities
entity = struct();

% Define objects
object = struct();

% Define properties
property = struct();

% Define relations
relation = struct();

% Define correspondences
correspondence = struct();

% Define linear correspondences
function is_linear = linear_correspondence(prop1, prop2, rel1, rel2, corr)
    is_linear = true;
    for i = 1:numel(rel1)
        for j = 1:numel(rel2)
            if rel1(i).subject == rel2(j).subject && rel1(i).object ~= rel2(j).object
                prop1_ij = corr.(prop1)(rel1(i).object, rel1(i).subject);
                prop1_jk = corr.(prop1)(rel2(j).object, rel2(j).subject);
                prop2_ij = corr.(prop2)(rel1(i).object, rel1(i).subject);
                prop2_jk = corr.(prop2)(rel2(j).object, rel2(j).subject);
                if ~(prop1_ij == prop1_jk && prop2_ij == prop2_jk)
                    is_linear = false;
                    return
                end
            end
        end
    end
end

% Test linear correspondence
entity.john = object();
entity.mary = object();
object.book =struct();
property.color = struct();
property.size = struct();
relation.on = struct();
correspondence.color_size = rand(2);
correspondence.size_color = inv(correspondence.color_size);
relation.on(1).subject = entity.john;
relation.on(1).object = object.book;
relation.on(2).subject = entity.mary;
relation.on(2).object = object.book;
relation.on(1).properties = [property.color, property.size];
relation.on(2).properties = [property.color, property.size];

% Check if on relation satisfies the LCA
is_linear = linear_correspondence('color_size', 'size_color', relation.on(1:1), relation.on(2:2), correspondence);
if is_linear
    disp('The on relation satisfies the Linear Correspondence Axiom.');
else
    disp('The on relation does not satisfy the Linear Correspondence Axiom.');
end
```

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**

```
(* Define entities *)
entity = ◁ ▷;

(* Define objects *)
object = ◁ ▷;

(* Define properties *)
property = ◁ ▷;

(* Define relations *)
relation = ◁ ▷;

(* Define correspondences *)
correspondence = ◁ ▷;

(* Define linear correspondences *)
linearCorrespondence[prop1_, prop2_, rel1_, rel2_, corr_]
:=
 Module[{isLinear = True},
  Do[
   Do[
    If[rel1[[i]]["subject"] == rel2[[j]]["subject"] &&
      rel1[[i]]["object"] ≠ rel2[[j]]["object"],
     prop1ij = corr[prop1][[rel1[[i]]["object"], rel1[[i]]
["subject"]]];
```

```
     prop1jk = corr[prop1][[rel2[[j]]["object"], rel2[[j]]["subject"]]];
     prop2ij = corr[prop2][[rel1[[i]]["object"], rel1[[i]]["subject"]]];
     prop2jk = corr[prop2][[rel2[[j]]["object"], rel2[[j]]["subject"]]];
     If[!(prop1ij == prop1jk && prop2ij == prop2jk),
      isLinear = False; Return[isLinear]]
     ]
   ],
   {i, Length[rel1]}, {j, Length[rel2]}
   ];
  isLinear
 ]

(* Test linear correspondence *)
entity["john"] = <| |>;
entity["mary"] = <| |>;
object["book"] = <| |>;
property["color"] = <| |>;
property["size"] = <| |>;
relation["on"] = {};
correspondence["colorSize"] = RandomReal[1, {2, 2}];
correspondence["sizeColor"] = Inverse[correspondence["colorSize"]];
AppendTo[relation["on"],
 Association[
  "subject" -> entity["john"],
  "object" -> object["book"],
  "properties" -> {property["color"], property["size"]}
 ]]
AppendTo[relation["on"],
 Association[
  "subject" -> entity["mary"],
  "object" -> object["book"],
  "properties" -> {property["color"], property["size"]}
 ]]
```

```

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

- **Vue**

```
" Define entities
let entity = {}

" Define objects
let object = {}

" Define properties
let property = {}

" Define relations
let relation = {}

" Define correspondences
let correspondence = {}

" Define linear correspondences
function! LinearCorrespondence(prop1, prop2, rel1, rel2, corr)
    let isLinear = 1
    for i in range(len(rel1))
```

```
        for j in range(len(rel2))
            if rel1[i].subject == rel2[j].subject && rel1[i].object != rel2[j].object
                let prop1ij = corr[prop1][rel1[i].object][rel1[i].subject]
                let prop1jk = corr[prop1][rel2[j].object][rel2[j].subject]
                let prop2ij = corr[prop2][rel1[i].object][rel1[i].subject]
                let prop2jk = corr[prop2][rel2[j].object][rel2[j].subject]
                if !(prop1ij == prop1jk && prop2ij == prop2jk)
                    let isLinear = 0
                    return isLinear
                endif
            endif
        endfor
    endfor
    return isLinear
endfunction

" Test linear correspondence
let entity.john = {}
```

Here's the complete Vim snippet:

```
" Define entities
let entity = {}

" Define objects
let object = {}

" Define properties
let property = {}
```

```

" Define relations
let relation = {}

" Define correspondences
let correspondence = {}

" Define linear correspondences
function! LinearCorrespondence(prop1, prop2, rel1, rel2, corr)
let isLinear = 1
for i in range(len(rel1))
for j in range(len(rel2))
if rel1[i].subject == rel2[j].subject && rel1[i].object != rel2[j].object
let prop1ij = corr[prop1][rel1[i].object][rel1[i].subject]
let prop1jk = corr[prop1][rel2[j].object][rel2[j].subject]
let prop2ij = corr[prop2][rel1[i].object][rel1[i].subject]
let prop2jk = corr[prop2][rel2[j].object][rel2[j].subject]
if !(prop1ij == prop1jk && prop2ij == prop2jk)
let isLinear = 0
return isLinear
endif
endif
endfor
endfor
return isLinear
endfunction

" Test linear correspondence
let entity.john = {}
let entity.mary = {}
letme.book = {}
let property.color = {}
let property.size = {}
let relation.on = []
let correspondence.colorSize = [[0.2, 0.8], [0.6, 0.4]]
let correspondence.sizeColor =
invert(correspondence.colorSize)
call add(relation.on, {'subject': entity.john, 'object':
object.book, 'properties': [property.color,
property.size]})
call add(relation.on, {'subject': entity.mary, 'object':
object.book, 'properties': [property.color,
property.size]})

" Check if on relation satisfies the LCA
let isLinear = LinearCorrespondence('colorSize',
'sizeColor', relation.on[0:0], relation.on[1:1],
correspondence)
if isLinear
echo 'The on relation satisfies the Linear Correspondence
Axiom.'
else
echo 'The on relation does not satisfy the Linear
Correspondence Axiom.'
endif
```

- **R**

```
# Define entities
entity <- list()

# Define objects
object <- list()

# Define properties
property <- list()

# Define relations
relation <- list()
```

```r

# Define correspondences
correspondence <- list()

# Define linear correspondences
linear_correspondence <- function(prop1, prop2, rel1, rel2, corr) {
  is_linear <- TRUE
  for (i in seq_along(rel1)) {
    for (j in seq_along(rel2)) {
      if (identical(rel1[[i]]$subject, rel2[[j]]$subject) && rel1[[i]]$object != rel2[[j]]$object) {
        prop1_ij <- corr[[prop1]][[rel1[[i]]$object, rel1[[i]]$subject]]
        prop1_jk <- corr[[prop1]][[rel2[[j]]$object, rel2[[j]]$subject]]
        prop2_ij <- corr[[prop2]][[rel1[[i]]$object, rel1[[i]]$subject]]
        prop2_jk <- corr[[prop2]][[rel2[[j]]$object, rel2[[j]]$subject]]
        if (!(prop1_ij == prop1_jk && prop2_ij== prop2_jk)) {
          is_linear <- FALSE
          return(is_linear)
        }
      }
    }
  }
  is_linear
}

# Test linear correspondence
entity$john <- list()
entity$mary <- list()
object$book <- list()
property$color <- list()
property$size <- list()
```

```r
relation$on ← list()
correspondence$color_size ← matrix(c(0.2, 0.8, 0.6, 0.4), nrow = 2, ncol = 2, byrow = TRUE)
correspondence$size_color ← solve(correspondence$color_size)
relation$on[[1]] ← list(subject = entity$john, object = object$book, properties = list(property$color, property$size))
relation$on[[2]] ← list(subject = entity$mary, object = object$book, properties = list(property$color, property$size))

# Check if on relation satisfies the LCA
is_linear ← linear_correspondence("color_size", "size_color", relation$on[1:1], relation$on[2:2], correspondence)
if (is_linear) {
  print("The on relation satisfies the Linear Correspondence Axiom.")
} else {
  print("The on relation does not satisfy the Linear Correspondence Axiom.")
}
```

- **Julia**

```julia
# Define entities
entity = Dict()

# Define objects
object = Dict()

# Define properties
property = Dict()
```

```julia

# Define relations
relation = Dict()

# Define correspondences
correspondence = Dict()

# Define linear correspondences
function linear_correspondence(prop1, prop2, rel1, rel2, corr)
    is_linear = true
    for i in 1:length(rel1)
        for j in 1:length(rel2)
            if rel1[i]["subject"] == rel2[j]["subject"] && rel1[i]["object"] != rel2[j]["object"]
                prop1_ij = corr[prop1][rel1[i]["object"], rel1[i]["subject"]]
                prop1_jk = corr[prop1][rel2[j]["object"], rel2[j]["subject"]]
                prop2_ij = corr[prop2][rel1[i]["object"], rel1[i]["subject"]]
                prop2_jk = corr[prop2][rel2[j]["object"], rel2[j]["subject"]]
                if !(prop1_ij == prop1_jk && prop2_ij == prop2_jk)
                    is_linear = false
                    return is_linear
                end
            end
        end
    end
    is_linear
end

# Test linear correspondence
entity["john"] = Dict()
entity["mary"] = Dict()
```

```julia
object["book"] =Dict()

# Define properties
property["color"] = Dict()
property["size"] = Dict()

# Define relations
relation["on"] = []

# Define correspondences
correspondence["color_size"] = [0.2 0.8; 0.6 0.4]
correspondence["size_color"] = inv(correspondence["color_size"])

# Set up example relation
push!(relation["on"], Dict("subject" => entity["john"], "object" => object["book"], "properties" => [property["color"], property["size"]]))
push!(relation["on"], Dict("subject" => entity["mary"], "object" => object["book"], "properties" => [property["color"], property["size"]]))

# Check if on relation satisfies the LCA
is_linear = linear_correspondence("color_size", "size_color", relation["on"][1:1], relation["on"][2:2], correspondence)
if is_linear
    println("The on relation satisfies the Linear Correspondence Axiom.")
else
    println("The on relation does not satisfy the Linear Correspondence Axiom.")
end
```

- **OWL2**

```
Prefix(ex: http://example.com/)
Prefix(owl: http://www.w3.org/2002/07/owl#)
Prefix(rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#)
Prefix(rdfs: http://www.w3.org/2000/01/rdf-schema#)

# Define entities
Class(ex:Entity)

# Define objects
Class(ex:Object)

# Define properties
Class(ex:Property)

# Define relations
Class(ex:Relation)

# Define correspondences
Class(ex:Correspondence)

# Define linear correspondences
Class(ex:LinearCorrespondence)
SubClassOf(ex:Correspondence)

# Define subject and object properties
ObjectProperty(ex:subjectOf)
Domain(ex:Entity)
Range(ex:Relation)

# ObjectProperty(ex:objectOf)
Domain(ex:Object)
Range(ex:Relation)

# Define property and relation properties
ObjectProperty(ex:hasProperty)
Domain(ex:Relation)
```

```
Range(ex:Property)

# ObjectProperty(ex:hasCorrespondence)
Domain(ex:Relation)
Range(ex:Correspondence)

# Define correspondence mapping properties
DatatypeProperty(ex:mapsTo)
Domain(ex:Correspondence)
Range(xsd:float)

# Define example entities
Individual(ex:john)
Type(ex:john ex:Entity)

# Individual(ex:mary)
Type(ex:mary ex:Entity)

# Individual(ex:book)
Type(ex:book ex:Object)

# Define properties
Individual(ex:color)
Type(ex:color ex:Property)

# Individual(ex:size)
Type(ex:size ex:Property)

# Define relations
Individual(ex:on1)
Type(ex:on1 ex:Relation)
ObjectPropertyAssertion(ex:subjectOf ex:john ex:on1)
ObjectPropertyAssertion(ex:objectOf ex:book ex:on1)
ObjectPropertyAssertion(ex:hasProperty ex:on1 ex:color)
ObjectPropertyAssertion(ex:hasProperty ex:on1 ex:size)

Individual(ex:on2)
```

```
Type(ex:on2 ex:Relation)
ObjectPropertyAssertion(ex:subjectOf ex:mary ex:on2)
ObjectPropertyAssertion(ex:objectOf ex:book ex:on2)
ObjectPropertyAssertion(ex:hasProperty ex:on2 ex:color)
ObjectPropertyAssertion(ex:hasProperty ex:on2 ex:size)

# Define correspondences
Individual(ex:color_size)
Type(ex:color_size ex:LinearCorrespondence)
DatatypePropertyAssertion(ex:mapsTo ex:color_size
"0.2"^^xsd:float ex:color ex:size)
DatatypePropertyAssertion(ex:mapsTo ex:color_size
"0.8"^^xsd:float ex:size ex:color)
Individual(ex:size_color)
Type(ex:size_color ex:LinearCorrespondence)
DatatypePropertyAssertion(ex:mapsTo ex:size_color
"0.6"^^xsd:float ex:color ex:size)
DatatypePropertyAssertion(ex:mapsTo ex:size_color
"0.4"^^xsd:float ex:size ex:color)

# Check if on relation satisfies the LCA
ClassAssertion(ex:LinearCorrespondence ex:color_size)
ClassAssertion(ex:LinearCorrespondence ex:size_color)
ObjectPropertyAssertion(ex:hasCorrespondence ex:on1
ex:color_size)
ObjectPropertyAssertion(ex:hasCorrespondence ex:on2
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**

```go
package main

import (
  "fmt"
)

// Define entities
type Entity struct{}

// Define objects
type Object struct{}

// Define properties
type Property struct{}

// Define relations
type Relation struct {
  subject Entity
  object  Object
  props   []Property
}

// Define correspondences
type Correspondence struct {
  mapping map[Property]map[Entity]float64
}

// Define linear correspondences
type LinearCorrespondence struct {
  Correspondence
}
```

```go

// Define function to check if a pair of relations satisfy the Linear Correspondence Axiom
func isLinearCorrespondence(prop1 Property, prop2 Property, rel1 Relation, rel2 Relation, corr Correspondence) bool {
  isLinear := true
  for _, r1 := range rel1.props {
    for _, r2 := range rel2.props {
      if rel1.subject == rel2.subject && rel1.object != rel2.object {
        prop1_ij := corr.mapping[prop1][r1]
        prop1_jk := corr.mapping[prop1][r2]
        prop2_ij := corr.mapping[prop2][r1]
        prop2_jk := corr.mapping[prop2][r2]
        if !(prop1_ij == prop1_jk && prop2_ij == prop2_jk) {
          isLinear = false
          return isLinear
        }
      }
    }
  }
  return isLinear
}

func main() {
  // Define example entities
  john := Entity{}
  mary := Entity{}
  book := Object{}

  // Define example properties
  color := Property{}
  size := Property{}

  // Define example relations
  on1 := Relation{subject: john, object: book, props: []Property{color, size}}
  on2 := Relation{subject: mary, object: book, props: []Property{color, size}}

  // Define example correspondences
  colorSizeCorr := Correspondence{
    mapping: map[Property]map[Entity]float64{
      color: map[Entity]float64{
        size: 0.8,
      },
      size: map[Entity]float64{
        color: 0.2,
      },
    },
  }
  sizeColorCorr := Correspondence{
    mapping: map[Property]map[Entity]float64{
      color: map[Entity]float64{
        size: 0.6,
      },
      size: map[Entity]float64{
        color: 0.4,
      },
    },
  }
  colorSizeLinCorr := LinearCorrespondence{colorSizeCorr}
  sizeColorLinCorr := LinearCorrespondence{sizeColorCorr}

  // Check if on1 and on2 satisfy the Linear CorrespondenceAxiom
  if isLinearCorrespondence(color, size, on1, on2, colorSizeLinCorr) && isLinearCorrespondence(size, color, on1, on2, sizeColorLinCorr) {
    fmt.Println("The Linear Correspondence Axiom is satisfied.")
  } else {
        fmt.Println("The Linear Correspondence Axiom is not satisfied.")
    }
}
```

- **Scala**

```
lass Entity

class Object

class Property

class Relation(val subject: Entity, val obj: Object, val properties: List[Property])

class Correspondence(val mapping: List[List[Double]], val properties: List[Property])

class LinearCorrespondence(val correspondence: Correspondence) {
  def isLinear(prop1: Property, prop2: Property, rel1: Relation, rel2: Relation): Boolean = {
    for (r1 ← rel1.properties) {
      for (r2 ← rel2.properties) {
        if (rel1.subject eq rel2.subject) && (rel1.obj ne rel2.obj) {
          val prop1_ij = correspondence.mapping(properties.indexOf(prop1))(rel1.properties.indexOf(r1))
          val prop1_jk = correspondence.mapping(properties.indexOf(prop1))(rel2.properties.indexOf(r2))
          val prop2_ij = correspondence.mapping(properties.indexOf(prop2))(rel1.properties.indexOf(r1))
          val prop2_jk = correspondence.mapping(properties.indexOf(prop2))(rel2.properties.indexOf(r2))
          if (prop1_ij ≠ prop1_jk || prop2_ij ≠ prop2_jk) {
            return false
          }
        }
      }
    }
    true
  }
}

//Example usage
val john = new Entity
val mary = new Entity
val book = new Object

val color = new Property
val size = new Property

val on1 = new Relation(john, book, List(color, size))
val on2 = new Relation(mary, book, List(color, size))

val color_size_corr = new Correspondence(List(List(0.0, 0.8), List(0.2, 0.0)), List(color, size))
val size_color_corr = new Correspondence(List(List(0.0, 0.6), List(0.4, 0.0)), List(size, color))
val color_size_lin_corr = new LinearCorrespondence(color_size_corr)

color_size_lin_corr.isLinear(color, size, on1, on2) // returns true
```

```

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