

Properties of relations on a set determined by a matrix

Mr Ashlin Darius Govindasamy
University of South Africa
Department of Computer Science and Mathematics

November 21, 2022

Abstract

This paper introduces a technique of converting a relation set into a matrix of \mathbb{R}^n space using that matrix classifying the properties for the relational set. Using this technique, we can find the properties of a relation set by using some properties of the matrix. In programming, we can use this technique to determine our properties of a relation set by using the matrix. Code snippets are provided to show how this technique can be used in programming.

Contents

1	Relations	2
1.0.1	What is a relation?	2
1.0.1.1	Some specific relations	2
1.0.2	Properties of relations	3
1.0.2.1	Reflexive	3
1.0.2.2	Irreflexive	4
1.0.2.3	Symmetric	5
1.0.2.4	Antisymmetric	6
1.0.2.5	Transitive	6
1.0.2.6	Trichotomy	7
1.0.2.7	Equivalence relations	8
1.0.2.8	Order relations	8
2	Computer Program	10
2.1	Python Class	10

Chapter 1

Relations

1.0.1 What is a relation?

A (binary) relation R between sets A and B is a subset of $A \times B$. ($A \times B$ is a Cartesian product.)

Thus, a relation is a set of pairs.

The interpretation of this subset is that it contains all the pairs for which the relation is true. We write aRb if the relation is true for A and B (equivalently B , if $(A, B) \in R$).

A and B can be the same set, in which case the relation is said to be "on" rather than "between":

A binary relation R on a set A is a $\subseteq A \times A$. ($A \times A$ is a Cartesian product.)

Example of a relation using $A = \{0, 1, 2, 3\}$

$$R = \{(0, 0), (1, 1), (2, 2), (3, 3)\}$$

Relations may also be of other arities. An n -ary relation R between sets X_1, \dots , and $X_n \subseteq n$ -ary product $X_1 \dots X_n$, in which case R is a set of n -tuples.

1.0.1.1 Some specific relations

The empty relation between sets X and Y , or on E , is the empty set \emptyset .

The empty relation is false for all pairs.

The full relation (or universal relation) between sets X and Y is the set $X \times Y$.

The full relation on set E is the set $E \times E$.

The full relation is true for all pairs.

The identity relation on set E is the set $(x, x) | x \in E$.

The identity relation is true for all pairs whose first and second element are identical.

1.0.2 Properties of relations

A relation R is..	if ...
<i>reflexive</i>	xRx
<i>symmetric</i>	xRy implies yRx
<i>transitive</i>	xRy and yRz implies xRz
<i>irreflexive</i>	xRy implies $x \neq y$
<i>antisymmetric</i>	xRy and yRx implies $x=y$
<i>trichotomy</i>	xRy or $x=y$ or yRx

1.0.2.1 Reflexive

Let Relation R on A

Where $R \subseteq A \times A$

The reflexive property of a relation is that $\forall a \in A, aRa$ is true.

Also could be written as $R \subseteq A \times A$ and $\forall a \in A$, then $(a, a) \in R$

Example:

Let $A = \{1, 2, 3, 4\}$

Let $R = \{(1, 1), (2, 2), (2, 3), (3, 4), (3, 3), (4, 4), (4, 2)\}$

We could draw a matrix to show the relation R on A

To draw the matrix we plot the elements of A on the x-axis and y-axis.

Then we plot the pairs of R on the matrix as 1.

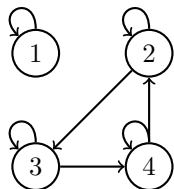
If the pair is not in R then we plot a 0.

	1	2	3	4
1	1	0	0	0
2	0	1	1	0
3	0	0	1	1
4	0	1	0	1

We can also draw a directional graph to show the relation R on A

We plot the elements of A as nodes.

We plot the pairs of R as directed edges.



Thus resulting in the relation R being reflexive.

This is because for all $a \in A, aRa$ is true.

For example, $1R1$ is true, $2R2$ is true, $3R3$ is true, $4R4$ is true.

Therefore R is reflexive.

In the matrix, we can see that the diagonal is all 1's.
 If you notice, the diagonal is the pairs (a, a) for all $a \in A$.
 In programming we look at this problem as a 2D array.

Using Mathematics logic, we can write this as:
 $R \subseteq A \times A$ and $\forall a \in A$, then $(a, a) \in R$

Where R is a 2D array.

Or in psuedocode:

```
function isReflexive(R)
    bValid = True
    for i = 0 to R.length
        for j = 0 to R.length
            if R[i][j] = 0
                bValid = False
            end if
        end for
    end for
    return bValid
```

1.0.2.2 Irreflexive

Let Relation R on A

Where $R \subseteq A \times A$

The irreflexive property of a relation is that $\forall a \in A$, aRa is false.

Also could be written as $R \subseteq A \times A$ and $\forall a \in A$, then $(a, a) \notin R$

Example:

Let $A = \{1, 2, 3, 4\}$

Let $R = \{(1, 2), (2, 3), (3, 4), (4, 1)\}$

We could draw a matrix to show the relation R on A

To draw the matrix we plot the elements of A on the x-axis and y-axis.

Then we plot the pairs of R on the matrix as 1.

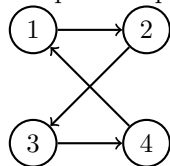
If the pair is not in R then we plot a 0.

	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

We can also draw a directional graph to show the relation R on A

We plot the elements of A as nodes.

We plot the pairs of R as directed edges.



Thus resulting in the relation R being irreflexive.

This is because for all $a \in A$, aRa is false.

For example, $1R1$ is false, $2R2$ is false, $3R3$ is false, $4R4$ is false.

Therefore R is irreflexive.

In the matrix, we can see that the diagonal is all 0's.

If you notice, the diagonal is the pairs (a, a) for all $a \in A$.

In programming we look at this problem as a 2D array.

Using Mathematics logic, we can write this as:
 $R \subseteq A \times A$ and $\forall a \in A$, then $(a, a) \notin R$
 Where R is a 2D array.
 Or in psuedocode:

```
function isIrreflexive(R)
  bValid = True
  for i = 0 to R.length
    for j = 0 to R.length
      if R[i][j] = 1
        bValid = False
      end if
    end for
  end for
  return bValid
```

If you want to be real and save time coding.

```
function isIrreflexive(R)
  return isReflexive(R)
end function
```

1.0.2.3 Symmetric

Let Relation R on A be symmetric if $\forall a, b \in A$ then $(a, b) \in R \implies (b, a) \in R$

Example: $R = (1, 1), (1, 3), (2, 3), (2, 4), (3, 1), (3, 2), (4, 2)$

We can draw a matrix to show the relation R on A

To draw the matrix we plot the elements of A on the x-axis and y-axis.

Then we plot the pairs of R on the matrix as 1.

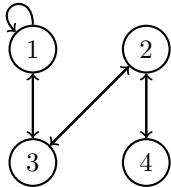
If the pair is not in R then we plot a 0.

	1	2	3	4
1	1	0	1	0
2	0	0	1	1
3	1	1	0	0
4	0	1	0	0

We can also draw a directional graph to show the relation R on A

We plot the elements of A as nodes.

We plot the pairs of R as directed edges.



Notice that the relation R is symmetric.

This is because for all $a, b \in A$, aRb is true.

aRb is true if and only if bRa is true.

For example, $1R1$ is true, $1R3$ is true, $2R3$ is true, $2R4$ is true, $3R1$ is true, $3R2$ is true, $4R2$ is true.

Therefore R is symmetric.

In the matrix, we can see that the matrix is symmetric.

Using our knowledge of matrices, we can write this as:

$$R \subseteq A \times A \text{ and } R = R^T$$

Where R is a 2D array.

Remember that R^T is the transpose of R .
The transpose of a matrix is the matrix found by interchanging its rows into columns or columns into rows.

In programming we look at this problem as a 2D array.

```
function isSymmetric(R)
    RT = transpose(R)
    return R == RT
end function
```

1.0.2.4 Antisymmetric

Let Relation R on A be antisymmetric if $\forall a, b \in A$ then $(a, b) \in R$ $(b, a) \in R$
Or in psuedocode:

```
function isAntisymmetric(R)
    bSymmetric = isSymmetric(R)
    if bSymmetric == True
        return False
    else
        return True
    end if
end function
```

1.0.2.5 Transitive

A relation R on A is transitive if $\forall a, b, c \in A$ then $(a, b) \in R$ and $(b, c) \in R$ then $(a, c) \in R$

Example: $R = (1, 1), (1, 3), (2, 3), (2, 4), (3, 1), (3, 2), (4, 2)$

We can draw a matrix to show the relation R on A

To draw the matrix we plot the elements of A on the x-axis and y-axis.

Then we plot the pairs of R on the matrix as 1.

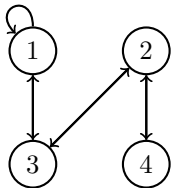
If the pair is not in R then we plot a 0.

	1	2	3	4
1	1	0	1	0
2	0	0	1	1
3	1	1	0	0
4	0	1	0	0

We can also draw a directional graph to show the relation R on A

We plot the elements of A as nodes.

We plot the pairs of R as directed edges.



For this problem we calculate the M^k where each element identifies the number of paths of length k .
So for example M^2 is the number of paths of length 2.

$$M^2 = M \cdot M$$

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 & 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 & 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 & 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 \\ 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 & 0 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 & 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 & 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 \\ 1 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 & 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 & 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 & 1 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 \\ 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 & 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 & 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 & 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 2 & 1 & 2 & 0 \\ 0 & 0 & 2 & 2 \\ 2 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

Looking at the matrix there must be at least one path of length 2 and length 1
Thus this is a transitive relation.

Looking at our psuedocode code.

```
def transitive(R):
    Rs = R**2
    bOnePass = False
    for i in range(len(Rs)):
        for j in range(len(Rs)):
            if Rs[i][j] == 1:
                bOnePass = True
    if bOnePass:
        bTwoPass = False
        for i in range(len(R)):
            for j in range(len(R)):
                if R[i][j] == 2:
                    bTwoPass = True
                    break
    if bOnePass and bTwoPass:
        return True
    else:
        return False
```

1.0.2.6 Trichotomy

In mathematics, the law of trichotomy states that every real number is either positive, negative, or zero.

More generally, a binary relation R on a set X is trichotomous if for all x and y in X , exactly one of xRy , yRx and $x=y$ holds. Writing R as $<$, this is stated in formal logic as:

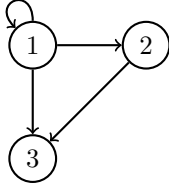
$$\forall x \in X \forall y \in X ([x < y \wedge \neg(y < x) \wedge \neg(x = y)] \vee [\neg(x < y) \wedge y < x \wedge \neg(x = y)] \vee [\neg(x < y) \wedge \neg(y < x) \wedge x = y])$$

A relation is trichotomous if, and only if, it is asymmetric and connected. If a trichotomous relation is also transitive, then it is a strict total order; this is a special case of a strict weak order.

In a much easier to understand way.
For example, the relation R on A is trichotomous.

$$R = (1, 1), (1, 2), (1, 3), (2, 3)$$

$$R = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$



If you notice the relation is trichotomous the relation R is asymmetric and edges are connected only once.

In our pseudocode we can see that we are checking if the relation is trichotomous.

```

def isTrichotomy(self):
    #every pair of nodes has one and only one edge between them.
    # each node can only have two edges but if the edge as the node as the source and
    for i in range(len(self.M)):
        count = 0
        for j in range(len(self.M)):
            if self.M[i][j] == 1:
                # if the edge is from the node to itself , then it is ok
                if i == j:
                    continue
                else:
                    count += 1
        if count > 2:
            return False
    # must be asymmetric
    if self.isSymmetric():
        return False
    return True

```

1.0.2.7 Equivalence relations

An **equivalence** relation is a relation that is **reflexive**, **symmetric**, and **transitive**.

An equivalence relation partitions its domain E into disjoint equivalence classes. Each equivalence class contains a set of elements of E that are equivalent to each other, and all elements of E equivalent to any element of the equivalence class are members of the equivalence class. The equivalence classes are disjoint: there is no $x \in E$ such that x is in more than one equivalence class. The equivalence classes exhaust E : there is no $x \in E$ such that x is in no equivalence class. Any element of an equivalence class may be its representative; the representative stands for all the members of its equivalence class.

1.0.2.8 Order relations

An order (or partial order) is a relation that is antisymmetric and transitive.

A **strict** order is one that is **irreflexive** and **transitive**; such an order is also trivially **antisymmetric** because there is no x and y such that xRy and yRx .

A **non-strict (weak)** order is one that is **reflexive**, **antisymmetric**, and **transitive**.

An order relation R on E is a **total order** if either xRy or $yRx \forall x, y \in E$.

An order relation R on E is a **partial order** if there is a $x, y \in E$ for which neither xRy nor yRx .

And order relation R on E is a **Triohotomy** if either xRy , yRx , or $x=y \forall x, y \in E$.

A weak total order is reflexive, antisymmetric, transitive and trichotomy.
A strict total order is irreflexive, transitive, and trichotomy.

Chapter 2

Computer Program

2.1 Python Class

Using all of our knowledge of matrices and relations, we can create a class that can be used to determine the properties of a relation set. This class can be used to determine the properties of a relation set by using the matrix.

I will use the following relation set as an example:

$$R = \{(\emptyset, \{a\}), (\emptyset, \{b\}), (\emptyset, \{a, b\}), (\{a\}, \{b\}), (\{a\}, \{a, b\}), (\{b\}, \{a, b\})\}$$

The class is shown below:

```
import numpy as np

class Relations():
    def __init__(self, s, R):
        self.s = s
        self.R = R
        self.M = np.zeros((len(s), len(s)))
        for i in R:
            self.M[i[0]-1][i[1]-1] = 1

    def returnMatrix(self):
        return self.M

    def isReflexive(self):
        for i in range(len(self.M)):
            if self.M[i][i] == 0:
                return False
        return True

    def isIrreflexive(self):
        return not self.isReflexive()

    def isSymmetric(self):
        T = self.M.transpose()
        if np.array_equal(T, self.M):
            return True
        return False

    def isAntisymmetric(self):
        if self.isSymmetric():
            return False
        return True

    def isTransitive(self):
        M = self.M
        Msquare = np.dot(M, M)
        bFoundOne = False
        bFoundTwo = False
        # if only two elements are in the relation, then it is transitive (We
```

```

        cannot prove that R is not transitive. Such a proof actually has a
        special name: it is vacuously true that R is transitive.)
    if len(self.R) == 2:
        return True

    for i in range(len(Msquare)):
        for j in range(len(Msquare)):
            if Msquare[i][j] == 1:
                bFoundOne = True
            if Msquare[j][i] == 2:
                bFoundTwo = True
        if bFoundOne and bFoundTwo:
            return True
    return False

def isTrichotomy(self):
    #every pair of nodes has one and only one edge between them.
    # each node can only have two edges but if the edge as the node as the
    # source and destination, then it is ok
    for i in range(len(self.M)):
        count = 0
        for j in range(len(self.M)):
            if self.M[i][j] == 1:
                # if the edge is from the node to itself, then it is ok
                if i == j:
                    continue
                else:
                    count += 1
            if count > 2:
                return False
    # must be asymmetric
    if self.isSymmetric():
        return False
    return True

def EquivalenceRelation(self):
    if self.isReflexive() and self.isSymmetric() and self.isTransitive():
        return True
    return False

def WeakPartialOrder(self):
    if self.isReflexive() and self.isTransitive() and self.isAntisymmetric():
        return True
    return False

def StrictPartialOrder(self):
    if self.isTransitive() and self.isAntisymmetric() and self.isIrreflexive():
        return True
    return False

def WeakTotalOrder(self):
    if self.isReflexive() and self.isTransitive() and self.isTrichotomy():
        return True
    return False

def StrictTotalOrder(self):
    if self.isTransitive() and self.isTrichotomy() and self.isIrreflexive():
        return True
    return False

def StrictEquivalenceRelation(self):
    if self.EquivalenceRelation() and self.isIrreflexive():
        return True
    return False

def printPropertiesOfRelation(self):
    # print only the properties that are true
    if self.isReflexive():
        print("Reflexive")
    if self.isIrreflexive():

```

```

        print("Irreflexive")
    if self.isSymmetric():
        print("Symmetric")
    if self.isAntisymmetric():
        print("Antisymmetric")
    if self.isTransitive():
        print("Transitive")
    if self.isTrichotomy():
        print("Trichotomy")
    if self.EquivalenceRelation():
        print("Equivalence_Relation")
    if self.StrictEquivalenceRelation():
        print("Strict_Equivalence_Relation")
    if self.WeakPartialOrder():
        print("Weak_Partial_Order")
    if self.StrictPartialOrder():
        print("Strict_Partial_Order")
    if self.WeakTotalOrder():
        print("Weak_Total_Order")
    if self.StrictTotalOrder():
        print("Strict_Total_Order")

# Using set
#s = {0,a,b,ab}
#R = {(0,{a}),(0,{b}),(0,{a,b}),({a},{b}),({a},{a,b}),({b},{a,b})}

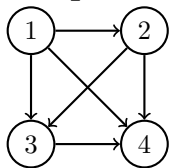
# since our class uses numerical values, we need to convert the set to a list of
# numbers and tuples of ordered pairs
s = [1, 2, 3,4]
#   0  a  b  ab
R = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]
r1 = Relations(s, R)
r1.printPropertiesOfRelation()

# std output is:
# Irreflexive
# Antisymmetric
# Transitive
# Strict Partial Order

```

Graphing the relation as a matrix

$$R = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



You may get the code from my [GitHub Gist](#) here.

I hope this helps.

[Source Code](#)