# **Abstract**

This project presents a Boolean satisfiability (SAT) model for analyzing genetic inheritance patterns in family pedigrees. Using recursive counting and constraint satisfaction, our model first validates the biological correctness of the pedigree structure by ensuring parents are of different sexes and not blood related, before determining whether traits follow dominant/recessive and X-linked/autosomal inheritance patterns. The implementation utilizes three distinct theories: an invalidity theory that tests pedigree structure constraints, a recessive theory that becomes unsatisfiable when dominant inheritance is detected, and an X-linked theory that becomes unsatisfiable when autosomal inheritance is identified.

The model processes pedigree data through multiple generations, tracking affected status, gender, and family relationships. For X-linked analysis, we employ a recursive counting approach where n(i,k)n(i, k) represents having kk affected individuals in the first ii people of a generation. This allows precise tracking of male-to-female ratios of affected individuals across generations.

A pedigree must first satisfy basic biological constraints before inheritance analysis proceeds. A trait is then classified as recessive when unaffected parents have affected offspring, and dominant otherwise. Similarly, a trait is determined to be X-linked when no generation has more affected males than females, and autosomal otherwise. The satisfiability of our theories directly maps to these inheritance patterns:

* Invalidity theory satisfiable ⟹ invalid pedigree structure (stop analysis)
* Recessive theory satisfiable ⟹ recessive inheritance
* X-linked theory satisfiable ⟹ X-linked inheritance
* Recessive theory unsatisfiable ⟹ dominant inheritance
* X-linked theory unsatisfiable ⟹ autosomal inheritance

# **Propositions**

Proposition to check for pedigree validity

* **InvalidPedigree():** represent that the pedigree chart is not valid
* The rest of the analysis will only run if **~ InvalidPedigree()**

Basic status propositions

* **BloodRelated(id):** represent that person with ID is a blood relative
  + For example, **BloodRelated(1) means** that the person with an ID of 1 is a blood relative
  + **~ BloodRelated(id):** represent that person with id is not a blood relative
* **Male(id):** represents that person with ID is male
  + For example, **Male(4)** means the person with an ID of 4 is a male
  + **~Male(id)**: represents that person with id is female
* **Affected(id):** represents that person with id has the trait
  + For example, **Affected(3)** means the person with the id of 3 is affected by the trait
* **Child(child id, parent1 id, parent2 id):** represents parent-child relationships
  + For example, Child(10, 3, 4) means that the person with an ID of 10 is the child of the people with IDs 3 and 4

Propositions for counting in sequences

* **Count(g, i, k, m):** represents that in generation g, first i people contain k affected individuals, where m is True for male counting and False for female counting

Count tracking propositions

* **MaleCount(g, i, k):** represents k affected males in first i people of generation g
  + For example, **MaleCounting(2, 4, 3)** means there are 2 affected males in the first 4 people counted in generation 3
* **FemaleCount(g, i, k):** represents k affected females in first i people of generation g
  + For example, **FemaleCount (1, 5, 2)** means there is 1 affected female in the 5 people counted so far in generation 2

Pattern analysis propositions

* **MoreMale(g):** represents that generation g has more affected males than affected females
  + For example **MoreMale(2):** means generation 2 has more affected males than females

# **Constraints**

Recursive counting constraints for generation analysis

1. Initial case (i=1) for male counting: When counting the first person (i=1), a count of 0 males means either they aren't male OR aren't affected
   * When counting the first person (i=1), a count of 0 males means either they aren't male OR aren't affected:

**MaleCount(g,1,0) ⇔ ¬Male(1) ∨ ¬Affected(1)**

* + A count of 1 male means they must be both male AND affected:

**MaleCount(g,1,1) ⇔ Male(1) ∧ Affected(1)**

1. Initial case (i=1) for female counting:
   * When counting the first person (i=1), a count of 0 females means either they are male OR aren't affected:

**FemaleCount(g,1,0) ⇔ Male(1) ∨ ¬Affected(1)**

* + A count of 1 female means they must be both not male AND affected:

**FemaleCount(g,1,1) ⇔ ¬Male(1) ∧ Affected(1)**

1. Recursive case for counting (i>1):
   * For each additional person, the male count either increases by 1 (if they are male AND affected) OR stays the same (if they're not male OR not affected):

**MaleCount(g,i,k) ⇔ ((MaleCount(g,i-1,k-1) ∧ Male(i) ∧ Affected(i)) ∨ (MaleCount(g,i-1,k) ∧ (¬Male(i) ∨ ¬Affected(i))))**

* + For each additional person, the female count either increases by 1 (if they are not male AND affected) OR stays the same (if they're male OR not affected):

**FemaleCount(g,i,k) ⇔ ((FemaleCount(g,i-1,k-1) ∧ ¬Male(i) ∧ Affected(i)) ∨ (FemaleCount(g,i-1,k) ∧ (Male(i) ∨ ¬Affected(i))))**

Constraint for recessive trait model

1. If person p1 and person p2 are unaffected and person c is affected AND c is the child of p1 and p2, then the trait is recessive
   * **(¬Affected(p1) ∧ ¬Affected(p2) ∧ Affected(c) ∧ Child(c,p1,p2)) ⟹ recessive**

Constraint for X-linked model

1. If in all generation (1-n) there are more affected males than affected females then trait is X-linked
   * **MaleCount(g,i,k) ∧ FemaleCount(g,i-1,k-1) ⟹ MoreMale(g)**
   * **(MoreMale(g1) ∧ MoreMale(g2)… ∧ MoreMale(gn)) ⟹ X-linked**

Constraints for invalid pedigree:

1. If any of the parents are related or the same gender the pedigree is not valid
   * If in any family units, person p1 AND person p2 are blood related AND person c is their child, the pedigree is not valid
     + Example of invalid family unit: **BloodRelated(P1)∧ BloodRelated(P2) ∧Child(c,p1,p2)**
     + **Invalid family unit proposition 1∨ Invalid family unit proposition 2… ∨Invalid family unit proposition n>>InvalidPedigree()**
   * If in any family units, person p1 is male AND person p2 is male OR person p1 is female AND person p2 is female AND person c is their child, the pedigree is invalid
     + Example of invalid family unit: **((Male(P1) ∧ Male(P2)) ∨ (¬Male(P1) ∧ ¬Male(P2)) )**
     + **Invalid family unit proposition 1∨ Invalid family unit proposition 2…∨ Invalid family unit proposition n>>InvalidPedigree()**

# **Model Exploration**

## Building pedigree

1. **First draft:** Implemented a 2d array based on number of generations and the number of people in the previous generation. We determined the size of the previous generation by dividing the current one.
   * **Issue with model:** too restrictive, limits the amount of generations and number of children per family.

A computer screen shot of a computer code

Description automatically generated

1. **Second Draft:** Recursive 2d dictionary that build up a person, then siblings then parents
   * **Issue with model:** the function does not follow the format of our propositions and constraints. The connections between people, like spouse, over complicated the build of the family tree.

A screenshot of a computer program

Description automatically generated

1. **Third Draft**: a dictionary of individual family members with their IDs as the key. Assign values to their keys by giving them characteristics (affected, gender, blood relation) using Char propositions. Then create a dictionary using for immediate families (siblings+parents) by linking individual family members using Rel propositions, then create a recursive function that builds the pedigree up using the dictionary
   * **Issue with draft**: the structure of the final pedigree dictionary is too complicated and hard to navigate. The proposition sibling ended up being unnecessary and the propositions were created incorrectly.

A computer screen shot of a program

Description automatically generated

A screen shot of a computer screen

Description automatically generated

1. **Final Draft:** Simplified the third draft Store the pedigree in 3 separate dictionaries:
   1. GENERATION Dictionary:
      1. Keys: Generation numbers (1, 2, 3)
      2. Values: Dictionary containing:
         1. id': List of person IDs in that generation
         2. 'count': Number of people in that generation
2. A computer code with numbers and symbols

   Description automatically generated
   1. PEOPLE Dictionary
      1. Keys: Person ID number
      2. Values: Dictionary containing person's characteristics:
         1. 'is\_male': Boolean for gender
         2. 'is\_affected': Boolean for disease status
         3. 'is\_blood\_relative': Boolean for blood relation status
   2. FAMILIES Dictionary
      1. List of dictionaries, each representing a family unit each with:
         1. 'parents': List of two parent IDs
         2. 'children': List of their children's IDs

A black background with text

Description automatically generated

This structure allows for:

1. Looking up individual characteristics
2. Following inheritance patterns through family units to determine if trait is recessive.
3. Tracking generations separately makes counting affected male and female for each generating easy. Simplifies x-linked analysis.

## X-linked theory

**In the first version,** the following functions were used to test for X-linked:

1. Count\_Affected (g, gender):
   1. Counts affected individuals separately for males and females in each generation
   2. Uses a loop to directly count everything rather than recursion
   3. Returns final count propositions (Affected\_Male\_Count or Affected\_Female\_Count)
2. Create\_all\_more\_male\_combo(g):
   1. Creates all possible combinations where males > females
   2. Iterates through possible male counts (m)
   3. For each male count, finds valid female counts (f < m)
   4. Returns combinations as logical expressions
3. Theory
   1. Compile the following constraints:
      1. Set Affected\_Male\_Count and Affected\_Female\_Count as true
      2. Then then set all\_more\_male\_combos >> more\_male
      3. Finally we have more\_male>>x-linked and ~ more\_male>>autosomal

**Limitations:**

1. Count\_affected did not count with logic.
2. Since a trait is always either x-linked or autosomal, x linked was always satisfied so the outputs were inaccurate.

**Updated version:**

In this version, count\_affected uses logic-based counting which is done recursively:

* Base case:
  + Count is 0 if person is not male OR not affected
  + Count is 1 if person is male AND affected

A screen shot of a computer code

Description automatically generated

* Recursive Cases (Additional People):
  + For each new person:
    - Either increment count if male AND affected
    - Or keep the same count

A screen shot of a computer program

Description automatically generated

This version also got rid of ~ more\_male>>autosomal, instead the constraints are set so that the theory will be satisfied if more\_male is true. This way if the theory is satisfiable, the trait is X-linked, and if it not, then the trait is autosomal.

## Characteristic assignment

**Initial versions**:

User input was the only way to assign characteristics. A series of questions would be asked about each individual to determine their characteristics.

**Limitations**:

During testing we found that this made the process tedious and it was easy to make mistakes.

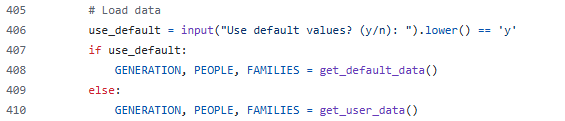
A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

**Updated Version:**

Later, we opted to include another option for data input, a default tree with data that can be loaded in. The data is provided from default\_tree.py. With the data presented in lists and dictionaries visually, it is more convenient for users to test and customize the pedigree chart.



And when the user is running the program, which they can input “y” or “n” to determine whether they want to preload a tree or input one themselves. In this picture, we have put “y” to tell the function to use the data from the default tree:



The main() output will be something like this, which uses the option of using the data of default\_tree.py:



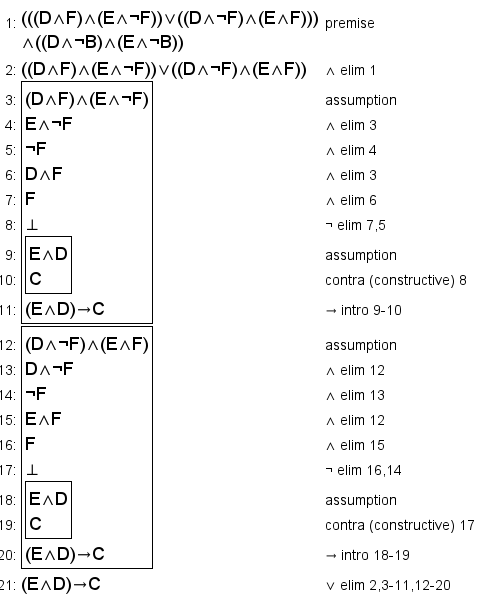
# **Jape Proof Ideas**

## Variables for the Jape Proof:

* D: True if person is first parent of a child
* E: True if person is second parent of a child
* F: True if the person is male
* B: True if blood related
* G: True if the disease is x-linked
* R: True if the disease is recessive

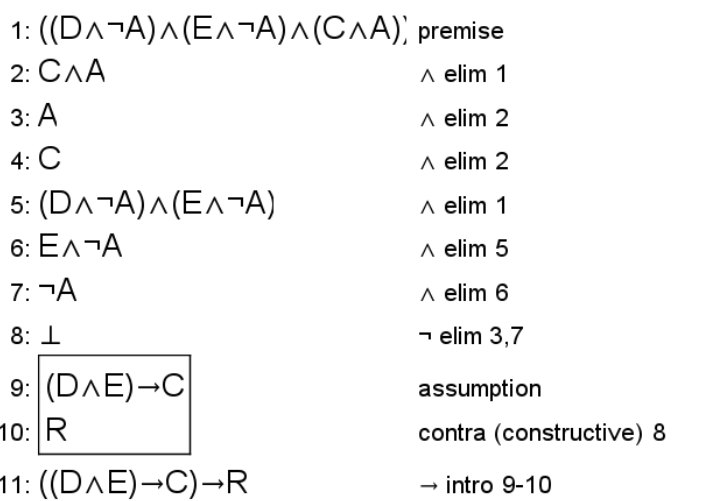
## Ideas:

1. One parent must be male and the other must not be male and the two parent cannot be blood related, then parent 1 and parent 2 implies true to be parents of the child, C, for C is true.

* Premise:
  + (((D∧F) ∧ (E∧¬F)) ∨ ((D∧¬F) ∧ (E∧F))) ∧ ((D∧¬B) ∧ (E∧¬B))
* Conclusion:
  + (E∧D) →C
* 

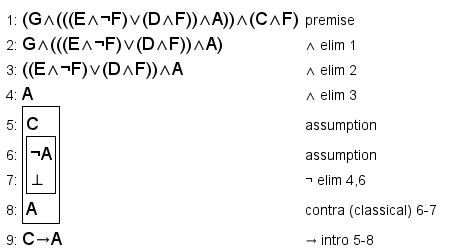
1. If both parents are not affected, but the child is, then the parents of the and the child implies that the disease is recessive.

* Premise:
  + ((D∧¬A) ∧ (E∧¬A) ∧ (C∧A))
* Conclusion:
  + ((D∧E) →C) →R



1. The diseases are X-linked and the mother, either parent 1 or 2 that is male, affected and the child is male, then it proves that the child is affected as well.

* Premise:
  + (G ∧ (((E∧¬F) ∨ (D∧F)) ∧A)) ∧ (C∧F)
* Conclusion:
  + C→A



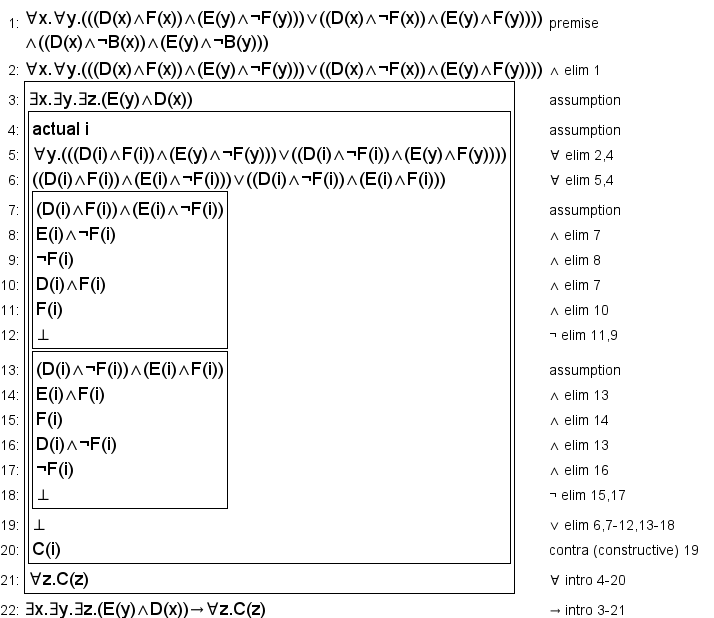
# **First Order Extension**

## Predicates:

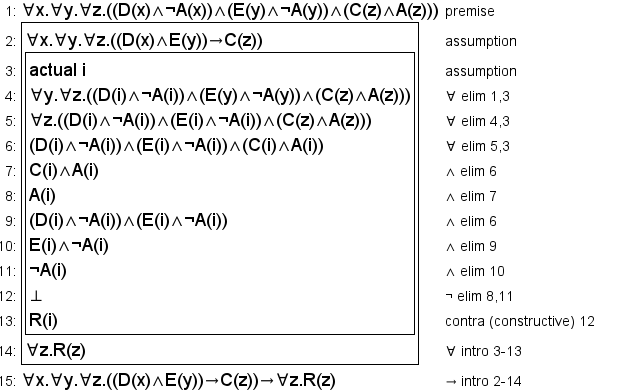
* D(x): x is first parent of a child
* E(x): x is second parent of a child
* F(x): x is male
* B(x): x blood related
* G(x): x is x-linked
* R(x): x is recessive

## Constraints:

1. For all x, y, where x is parent one and is either male or female, and not blood related to y, and y is parent two and is neither female or male depending on parent one, then there exist x,y,z where y is parent two with x, parent one, implies true for all z, a child. For every x, y, two valid parents, there exist a child that is also valid.



1. There exist x,y,z, that x is parent one and is not affect and y is parent two and is not affected and z is a child that is affected, then there exist x,y,z, where x is parent one and y is parent two of the child, z, implies that for z is infected with the recessive disease. For all x that is affected not and parent 1, and y that is not affected and the other parent, and z that is the child and affect, then for all x,y,z, x is parent one, y is parent 2 and z is the child and implies that for all z, the child, in this situation is affected by a recessive disease.



1. For all x, y, x is affected by the x-linked disease, x is either parent one or two that is female, which either is the mother, and affected and y the child is male, then there exist y, a child, implies that for all child y, is also affected by the x-linked disease. So, for all mother that is affected by a x-linked disease and have sons, there exist for all sons that they are affected by the disease.

