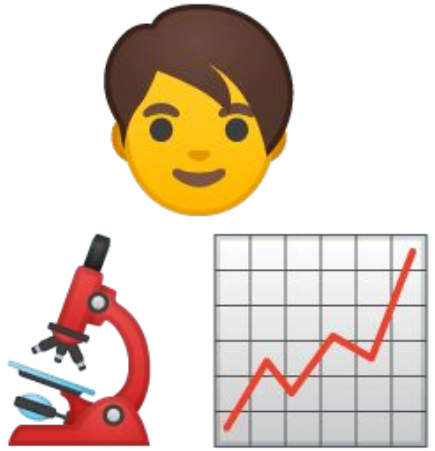
Explainable Decision Support using Defeasible Logic

Structure

David Pomerenke

ZingYang Zeng



Laurens Rutten

Progress

Demo

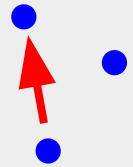
Outlook

David Pomerenke

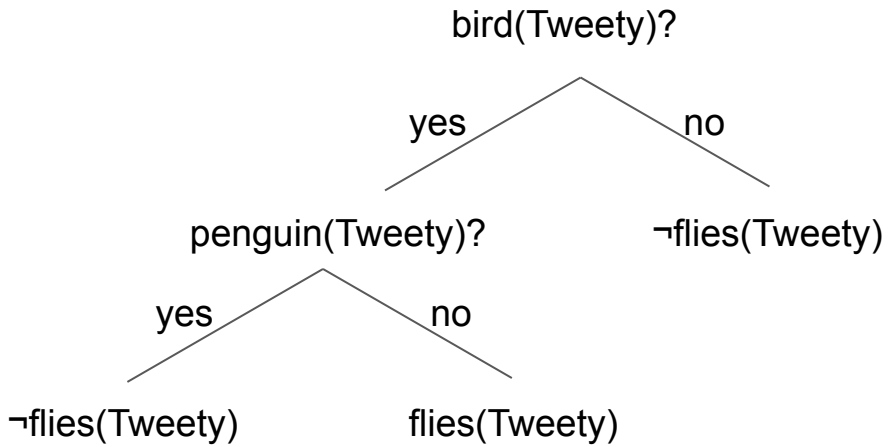
Progress

Introduction

bird(x) ~> flies(x)
penguin(x) ~> ¬flies(x)
penguin(x) --> bird(x)

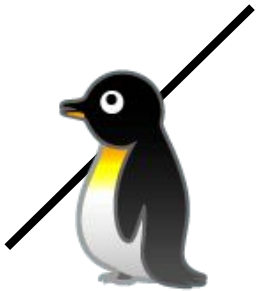


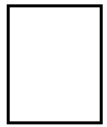
flies(Tweety)?



penguin(Tweety) **penguin(x) ~> ¬flies(x)**	¬flies(Tweety)

penguin(Tweety) penguin(x) --> bird(x) **bird(x) ~> flies(x)**	flies(Tweety)





Results

- Layperson’s website
- Presentation
- Argumentation Tableau

Explainable Artificial Intelligence

Explainable Decision Support Systems

Best Idea Ever: Implement a pseudo decision support system (Should I read this paper: Did you read it before? y-> Did you understand it? y-> Were there changes since the last time you read? n-> dun readit kek) Decision support systems become increasingly important in our every day lifes. Examples can be found in medicine and law, amongst other domains. Consider doctors that consult computer programmes to analyse a patient and get recommendations for potential treatments. Or tax software which asks the right questions for you to make the optimal financial decisions given the legal and financial situation you find yourself in. Already today we can see the impact that decision support systems have.
However, more often than not, those systems are so called black box systems, meaning that their inner workings are unknown, in some cases not only to the users, but even to the developers. This can have serious rammifications: Thinking of the medical example given above, the reader may ask themselves whether they would trust a decision made by a computer for reasons unkown to us! The savvy reader might be inclined to trust the machine, but the issue is at least controversial and shows that a lack of understanding a decision making process might lead to a lack of confidence in the decision came by that process.

Downloads/Links

- Second Period Presentation
- Final Report (to be added upon project completion)
- Documentation (to be added upon project completion)
- Code (permission will be granted on individual basis)
- Test Site

Beyond trust issues, even legal or moral implications need to be considered: Change your perspective from the patient to the doctor. Are you willing to take responsibility for the decision you make based on the computer-aided system? And if you do, can you be held accountable for them, given that you had no full knowledge of the underlying reasoning? These, again, are undecided matters.
Contemporarily, legislative instance in several countries pick up on that issue and regulations are being put in place [MIGHT ADD INFO BOX ABOUT COURT DECISIONS]

Arguments and Rules

To gain insights into the rationale behind decisions made or supported by a computer system, we laverage the relation between rules and arguments. Both rules and arguments exhibit a similiar logical structure: <\br> A rule checks whether a number of conditions apply. If they do, then the rule tells what action should follow. <\br> An argument checks whether a number of reasons hold. If they do, then the argument tells us what consequence should be concluded. <\br> Out approach, figuratively speaking, is to utilize this similarity in their logical structure to map the decisions of a decision support system to reasons which should be found in the application domain, i.e. the medical or juridical field. <\br> The advantage of that approach is that ...

Explainable Decisions via Argumentation

Our approach to explainable decisions support systems is to explicate the rules underlying a decision support system within a given domain. We then analize the relation between these rules and under which conditions they apply. Finally, we bring this analysis back to the decision support system so that given any decision, we can explicate the rationale leading to that decision. (This is then, by nature, limited to some kinds of decision support systems.

Progress

Demo

Outlook

Prototypes

Propositional Tableau

Enter a formular in propositional logic. All expressions except atoms need to be nested with brackets. The logical symbols can be typed as you prefer:

- implies, impl, -->, ->, -> will all work for the logial implication
- f, fAlSe, 0 will all work for false
- etc.

¬((p ∨ (q ∧ r)) → ((p ∨ q) ∧ (p ∨ r)))

Get Tableau

¬((p ∨ (q ∧ r)) → ((p ∨ q) ∧ (p ∨ r)))
p ∨ (q ∧ r)
¬((p ∨ q) ∧ (p ∨ r))
p
¬(p ∨ q)
¬p
¬q
❌
¬(p ∨ r)
¬p
¬r
❌
q ∧ r
q
r
¬(p ∨ q)
¬p
¬q
❌
¬(p ∨ r)
¬p
¬r
❌

Defeasible Tableau

Enter a formular in propositional logic. All expressions except atoms need to be nested with brackets. The logical symbols can be typed as you prefer:

- implies, impl, -->, ->, -> will all work for the logial implication
- f, fAlSe, 0 will all work for false
- etc.

Rules:

Employed ~> CanMakeRequestForChange
Employed & LessThanTenEmployees ~> ¬CanMakeRequestForChange
Employed & ReachedOldAgeInsurance ~> ¬CanMakeRequestForChange
Employed & MilitaryOfficial ~> ¬CanMakeRequestForChange

Initial information:

Employed
¬LessThanTenEmployees
¬ReachedOldAgeInsurance
MilitaryOfficial
WorkedForAtLeastTwentySixWeeks

Question:

CanMakeRequestForChange

Get Arguments

Pro:
({({Employed}, Employed ~> CanMakeRequestForChange)}, CanMakeRequestForChange)

Contra:
({({Employed, MilitaryOfficial}, Employed ∧ MilitaryOfficial ~> ¬CanMakeRequestForChange)}, ¬CanMakeRequestForChange)

Progress

Demo

Outlook

Implementation

Phase 1

- class Proposition
  - class Variable
  - class True
  - class False
  - class Complex
    - classes And, Or, ...
  - method evaluate(model=...)
- method parse('*(a & b) -> c*')
- class Node
  - method expand()
- class PropositionalTableau
  - method evaluate()

Progress Demo Outlook

Challenges

How to check whether the precondition of a rule is matched?

How to merge the closure supports for multiple branches?

How to implement test propositions?

How to eliminate repetitive arguments?

How to program effectively in a team?

How to combine expansion of nodes and addition of new arguments to the root?

How to prevent endless loops?

How to improve efficiency?

Progress

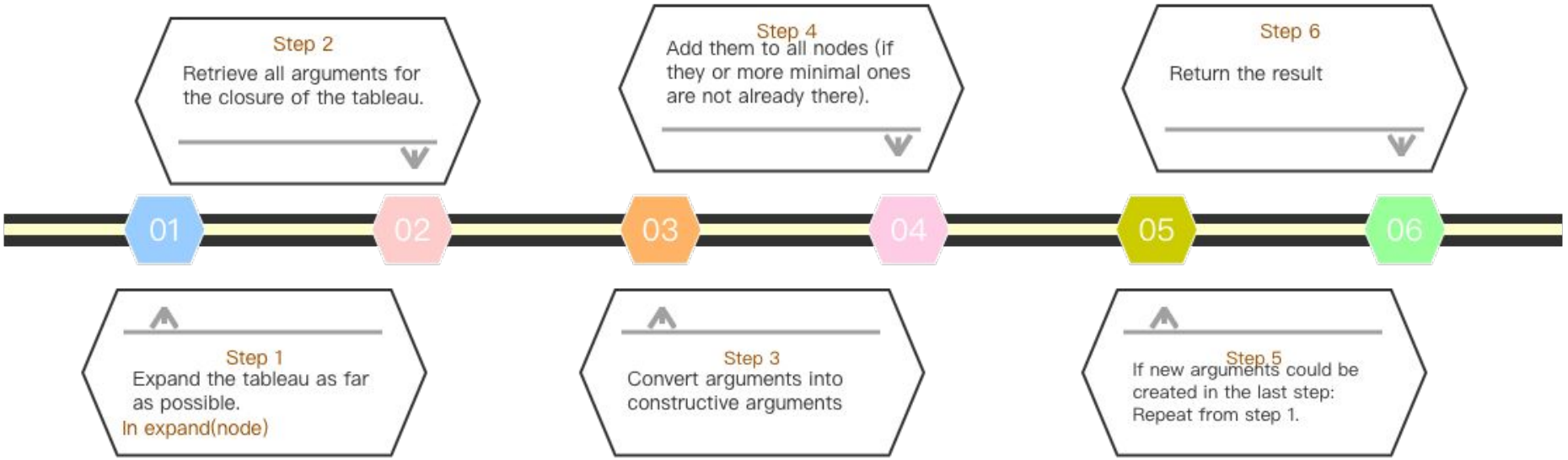
Demo

Outlook

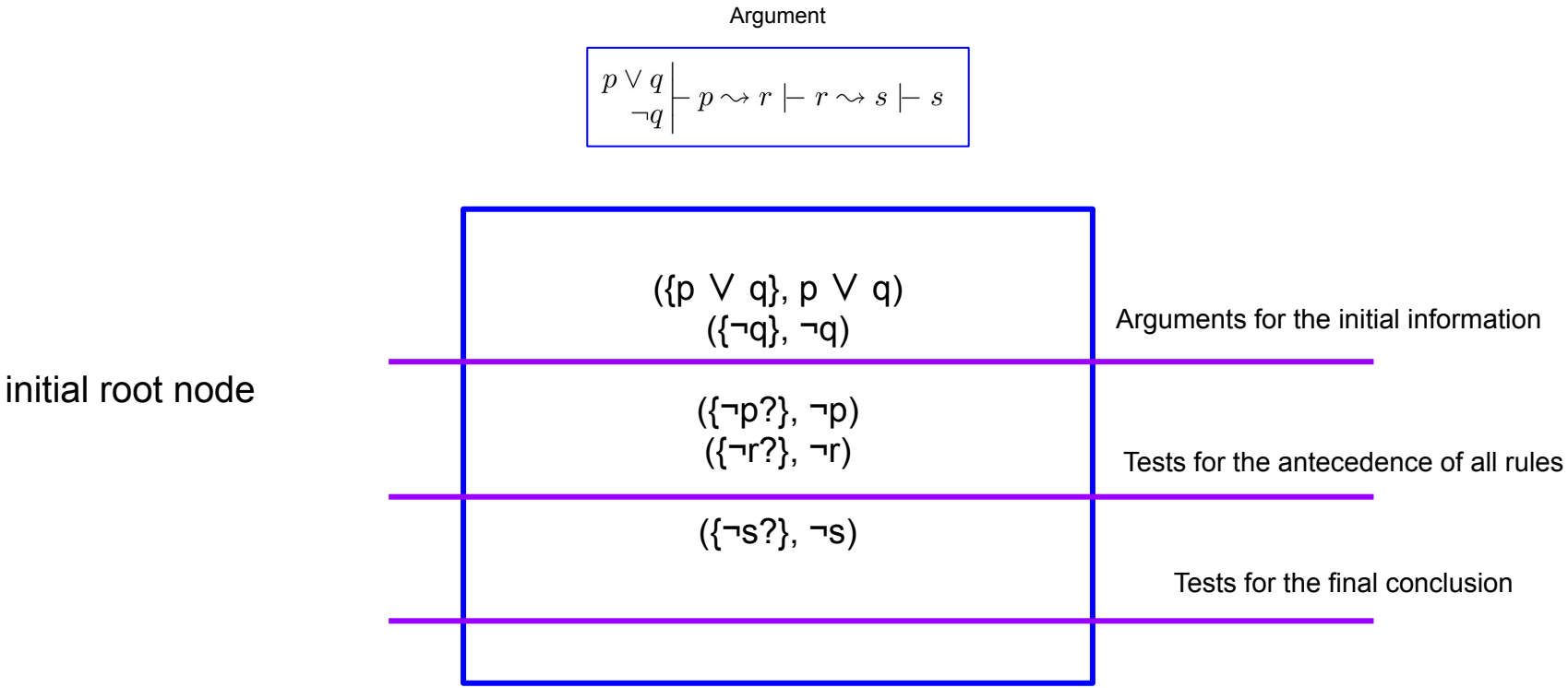
ZingYang Zeng

Demo

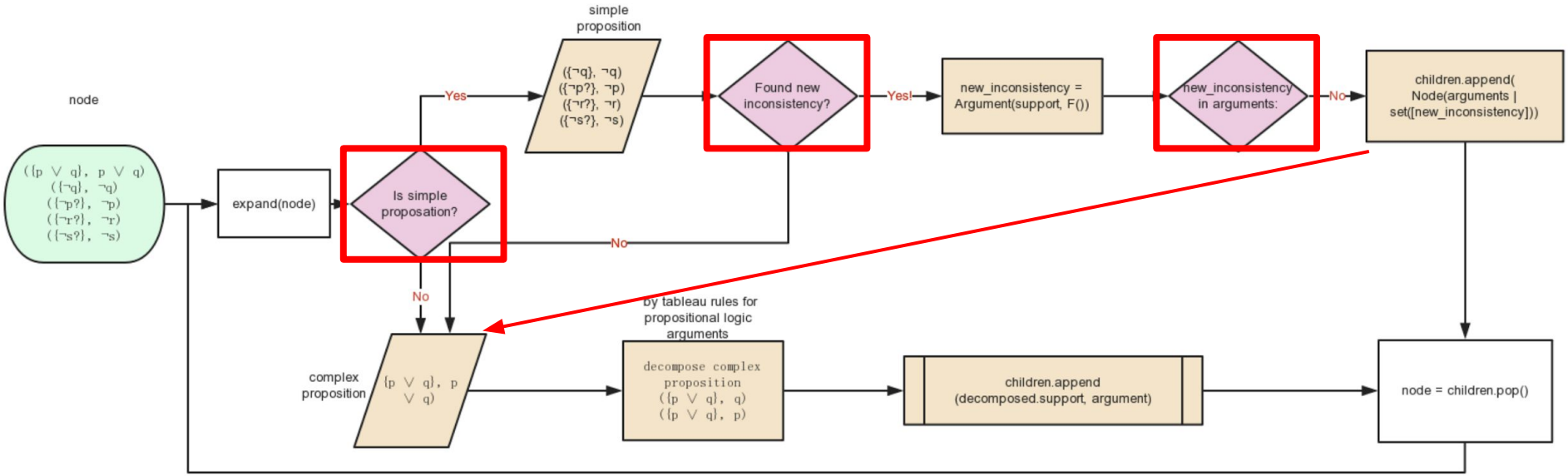
The Implementation steps



The Implementation



Expand the tableau as far as possible



Progress Demo Outlook

Defeasible Tableau

Enter a formular in propositional logic. All expressions except atoms need to be nested with brackets. The logical symbols can be typed as you prefer:

- implies, impl, -->, ->, -> will all work for the logical implication
- f, fAlSe, 0 will all work for false
- etc.

Rules:

```
Employed ~> CanMakeRequestForChange
Employed & LessThanTenEmployees ~> ¬CanMakeRequestForChange
Employed & ReachedOldAgeInsurance ~> ¬CanMakeRequestForChange
Employed & MilitaryOfficial ~> ¬CanMakeRequestForChange
```

Initial information:

```
Employed
¬LessThanTenEmployees
¬ReachedOldAgeInsurance
MilitaryOfficial
WorkedForAtLeastTwentySixWeeks
```

Question:

```
CanMakeRequestForChange
```

Defeasible Tableau

Enter a formular in propositional logic. All expressions except atoms need to be nested with brackets. The logical symbols can be typed as you prefer:

- implies, impl, -->, ->, -> will all work for the logical implication
- f, fAlSe, 0 will all work for false
- etc.

Rules:

```
Employed ~> CanMakeRequestForChange
Employed & LessThanTenEmployees ~> ¬CanMakeRequestForChange
Employed & ReachedOldAgeInsurance ~> ¬CanMakeRequestForChange
Employed & MilitaryOfficial ~> ¬CanMakeRequestForChange
```

Initial information:

```
Employed
¬LessThanTenEmployees
¬ReachedOldAgeInsurance
MilitaryOfficial
WorkedForAtLeastTwentySixWeeks
```

Question:

```
CanMakeRequestForChange
```

Get Arguments

Defeasible Tableau

Enter a formular in propositional logic. All expressions except atoms need to be nested with brackets. The logical symbols can be typed as you prefer:

- implies, impl, -->, ->, -> will all work for the logical implication
- f, fAlSe, 0 will all work for false
- etc.

Rules:

```
Employed ~> CanMakeRequestForChange
Employed & LessThanTenEmployees ~> ¬CanMakeRequestForChange
Employed & ReachedOldAgeInsurance ~> ¬CanMakeRequestForChange
Employed & MilitaryOfficial ~> ¬CanMakeRequestForChange
```

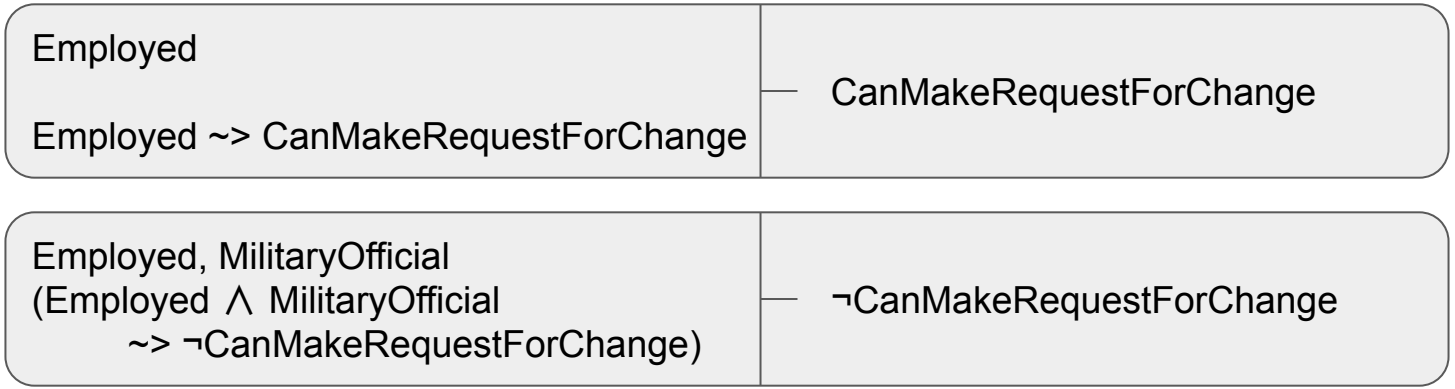
Initial information:

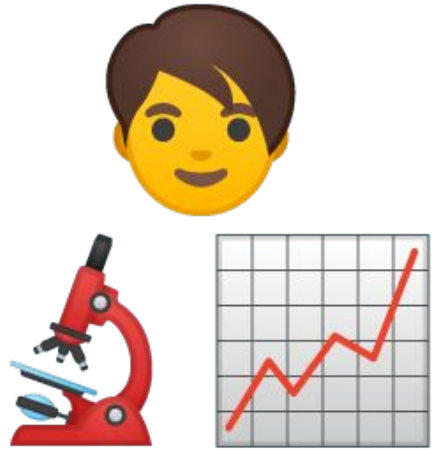
```
Employed
¬LessThanTenEmployees
¬ReachedOldAgeInsurance
MilitaryOfficial
WorkedForAtLeastTwentySixWeeks
```

Question:

```
CanMakeRequestForChange
```

Get Arguments

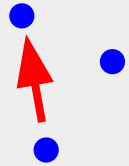

Employed
Employed ~> CanMakeRequestForChange
CanMakeRequestForChange
Employed, MilitaryOfficial
(Employed ∧ MilitaryOfficial
~> ¬CanMakeRequestForChange)
¬CanMakeRequestForChange



Laurens Rutten

Outlook

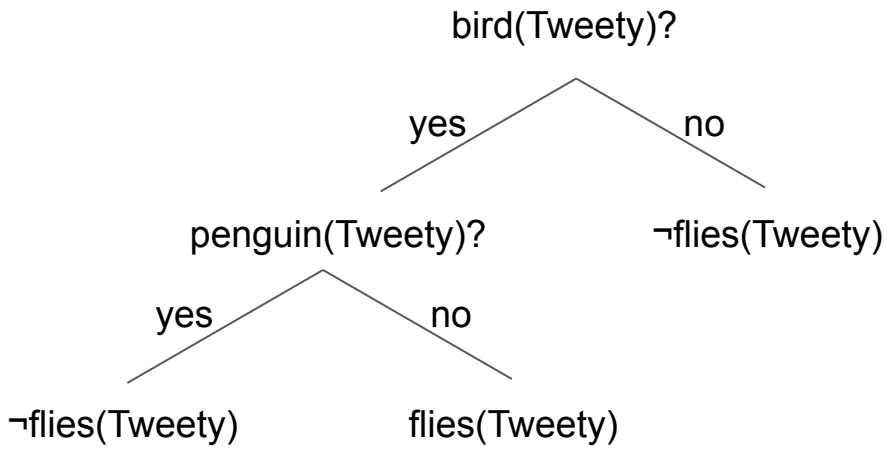
bird(x) ~> flies(x)
penguin(x) ~> ¬flies(x)
penguin(x) --> bird(x)

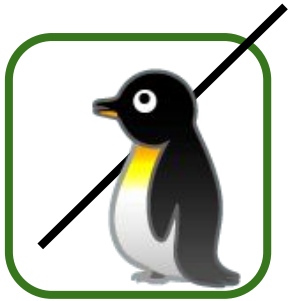


flies(Tweety)?

penguin(Tweety) **penguin(x) ~> ¬flies(x)**	⊢ ¬flies(Tweety)

penguin(Tweety) penguin(x) --> bird(x) **bird(x) ~> flies(x)**	⊢ flies(Tweety)





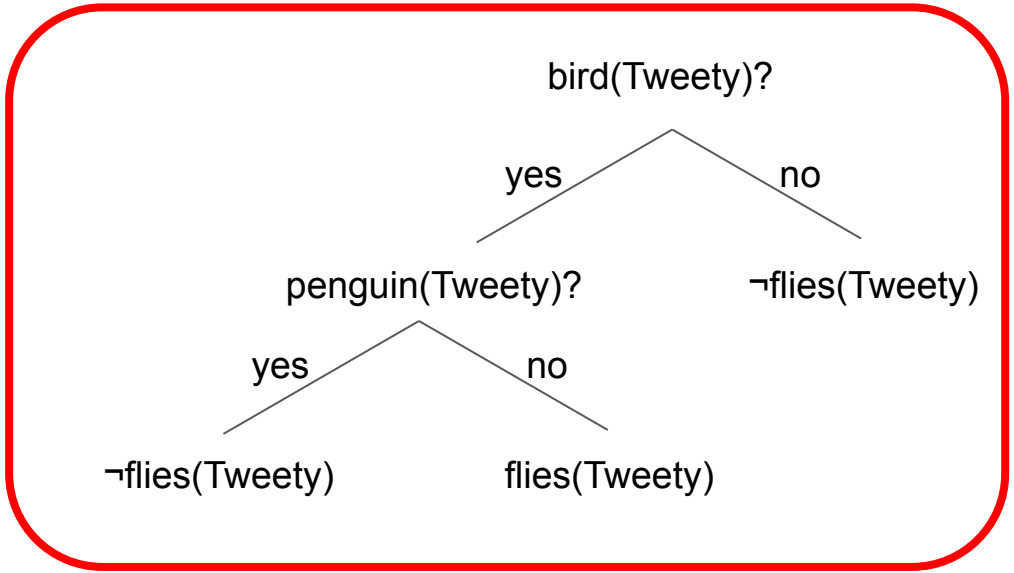
Progress

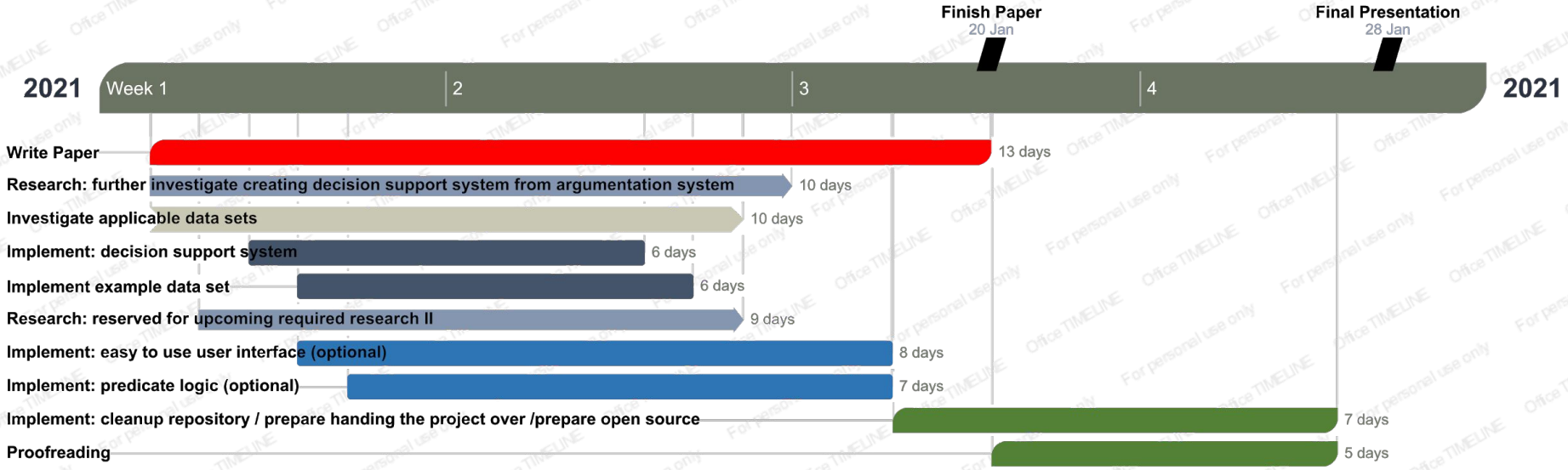
Demo

Outlook

Decision Support System

- Approach 1:
  - “brute force” generate outcomes for all possible worlds
  - apply conventional entropy-based decision tree generator
- Approach 2:
  - based on the relevant rules
  - apply more general rules first





Progress

Demo

Outlook

Done

- Defeasible logic
- Visualize results
- Layperson’s website

To Do

- Implement Decision Support System
- Apply to real world data
- Write the report
- Final presentation


Thank you for listening!
Questions?

Image sources

- Front slide
  piqsels.com/en/public-domain-photo-zkxit
  public domain