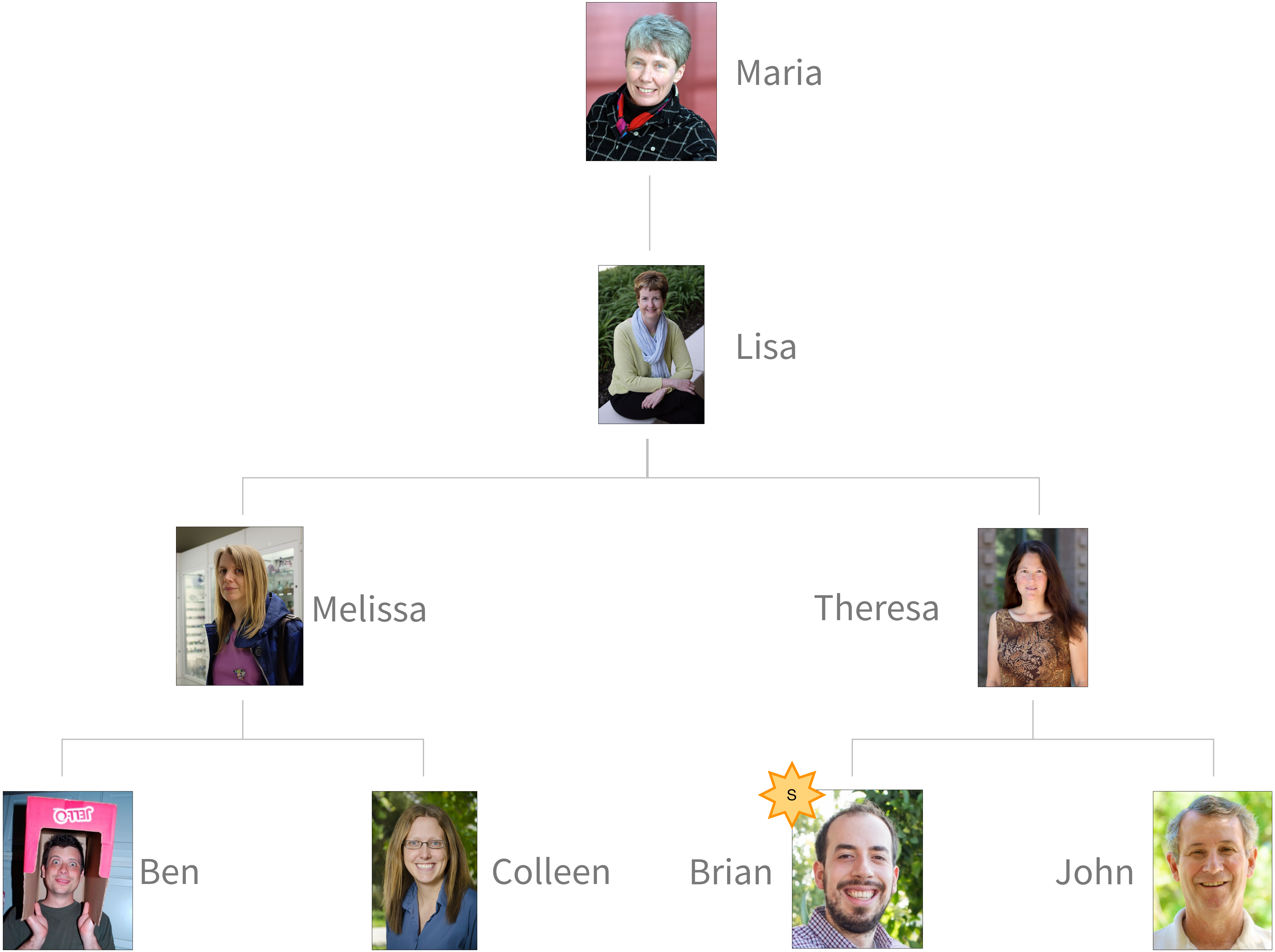
Let's use OOP to simulate something.


Maria
Lisa
Melissa
Theresa
Ben
Colleen
S
Brian
John

A person has a name.

A person can be a safety officer.

A person can have at most one boss.

We can access a person's name.

We can make someone be a safety officer.

We can determine whether a person is a safety officer.

We can access a person's boss.

Maria is Lisa’s boss.

Lisa is Melissa's boss.
Lisa is Theresa’s boss.

Melissa is Ben’s boss.
Melissa is Colleen’s boss.

Theresa is Brian’s boss.
Theresa is John’s boss.

Brian is a safety officer.

A person has a name.

```java
Person ben = new Person("Ben");
System.out.println(ben.getName());
```

```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    // getters…
    // auto-generated hashCode and equals…
}
```

A person can be a safety officer.

```
Person brian = new Person("Brian");
brian.makeSafetyOfficer();
```

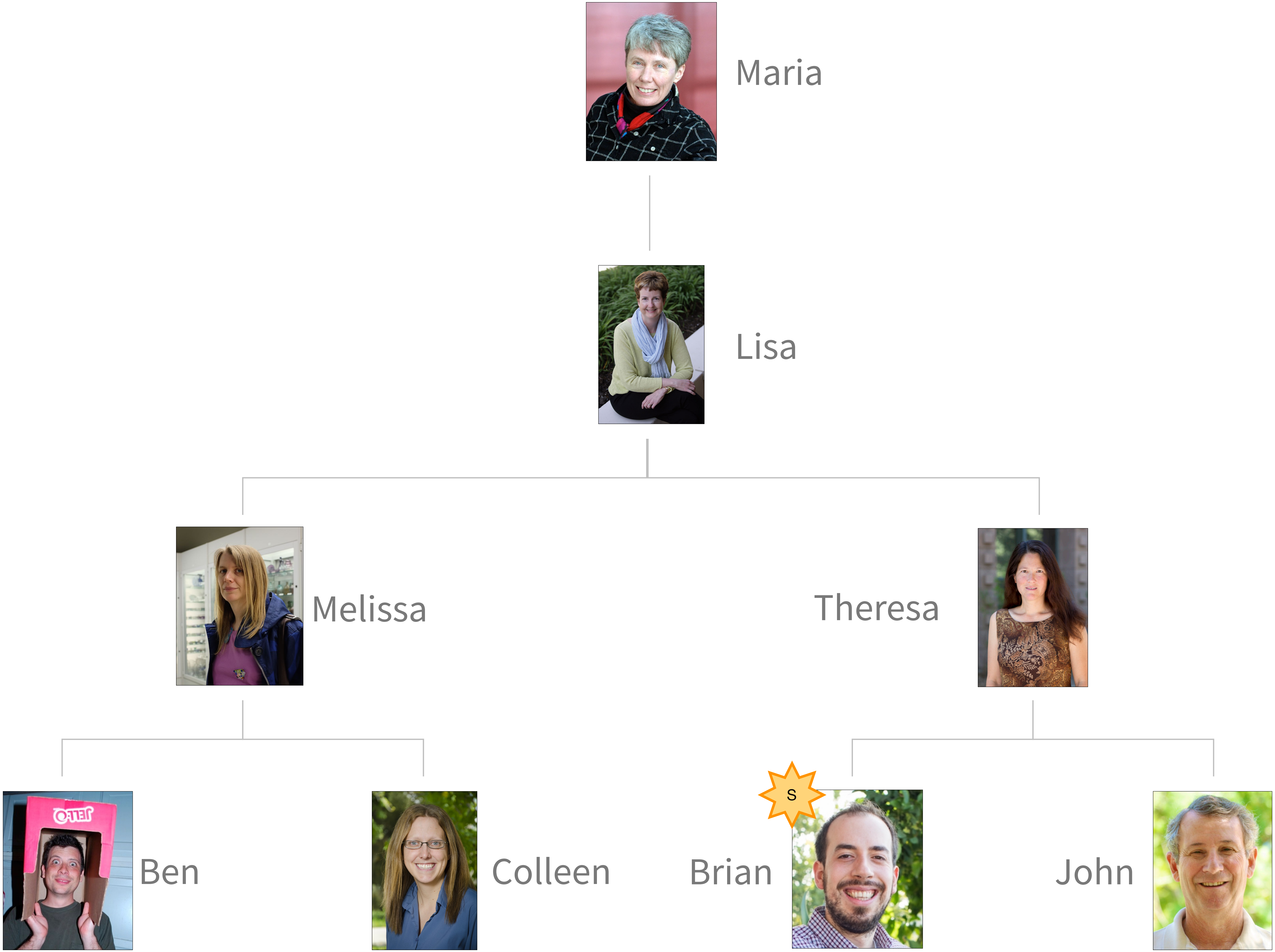
```
public class Person {
    ...
    private boolean isSafetyOfficer;
    ...
    public Person(String name) {
        this.name = name;
        this.isSafetyOfficer = false;
    }
    ...
    public void makeSafetyOfficer() {
        this.isSafetyOfficer = true;
    }
    ...
}
```

A person can have at most one boss.

```
Person lisa = new Person("Lisa", maria);
System.out.println(lisa.getBoss().getName());
System.out.println(maria.getBoss());
```

```
public class Person {
    ...
    private Person boss;
    ...
    public Person(String name) {
        this.name = name;
        this.isSafetyOfficer = false;
        this.boss = null;
    }

    public Person(String name, Person boss) {
        this(name);
        this.boss = boss;
    }
    ...
    public Person getBoss() {
        return this.boss;
    }
    ...
```


Maria
Lisa
Melissa
Theresa
Ben
Colleen
S
Brian
John

A person has a name.

A person can be a safety officer.

A person can have at most one boss.

We can access a person's name.

We can make someone be a safety officer.

We can determine whether a person is a safety officer.

We can access a person's boss.

Does A work with B? (i.e., are A and B coworkers?)
A and B are coworkers if and only if they have the same boss.

Is B the employee of A?
B is the employee of A if and only if A is B’s boss.

Who are A’s employees?
A’s employees are all the people whose boss is A.

Who are A's safety officers?
A’s safety officers are any coworkers of A who are safety officers.

Does *A* work with *B*?

A works with *B* if and only if they have the same boss.

```
ben.worksWith(colleen)
ben.worksWith(brian)
```

```
public boolean worksWith(Person person) {
    Person myBoss = this.getBoss();
    Person theirBoss = person.getBoss();
    return myBoss != null &&
           myBoss.equals(theirBoss);
}
```

Why can’t we just say this?

Maria is Lisa’s boss.

Lisa is Melissa's boss.
Lisa is Theresa’s boss.

Melissa is Ben’s boss.
Melissa is Colleen’s boss.

Theresa is Brian’s boss.
Theresa is John’s boss.

Brian is a safety officer.

Person *A* is an employee of person *B* if *B* is *A*'s boss.

Person *A* works with person *B* if they have the same boss.

Person *A*’s safety officer is anyone *A* works with who is a safety officer.

Behold: Prolog!

```
boss(maria, lisa).

boss(lisa, melissa).
boss(lisa, theresa).

boss(melissa, ben).
boss(melissa, colleen).

boss(theresa, brian).
boss(theresa, john).

safetyOfficer(brian).

employee(PersonA, PersonB) :- boss(PersonB, PersonA).

worksWith(PersonA, PersonB) :-
    boss(Boss, PersonA), boss(Boss, PersonB).

safetyOfficer(PersonA, PersonB) :-
    worksWith(PersonA, PersonB), safetyOfficer(PersonA).
```

4.1 Syntax of the Predicate Calculus

4-1 Definition. The syntax of the predicate calculus ($\mathcal{PC}$) consists of *symbols* and *formulas* as follows:

Symbols

parentheses: (,)

sentential connectives: $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$

quantifiers: $\forall$, $\exists$

$\mathcal{SC}$ letters (sentential letters): $A, B, \cdots Z$, and any of these letters with a positive Arabic numeral subscript.

predicate symbols: An n-ary predicate is an uppercase letter, $A, \cdots, Z$, with the numeral n as a superscript, where n denotes the arity of the predicate and $0 < n$. These uppercase letters may also have numerical subscripts. Note: We will usually omit the superscript when we know the arity of a predicate.

individual constants: lowercase letters $a, \cdots, r$, with or without numerical subscripts.

individual variables: lowercase letters $s, \cdots, z$, with or without numerical subscripts.

Formulas

The set of all predicate calculus ($\mathcal{PC}$) formulas is defined recursively, beginning with the atomic formulas.

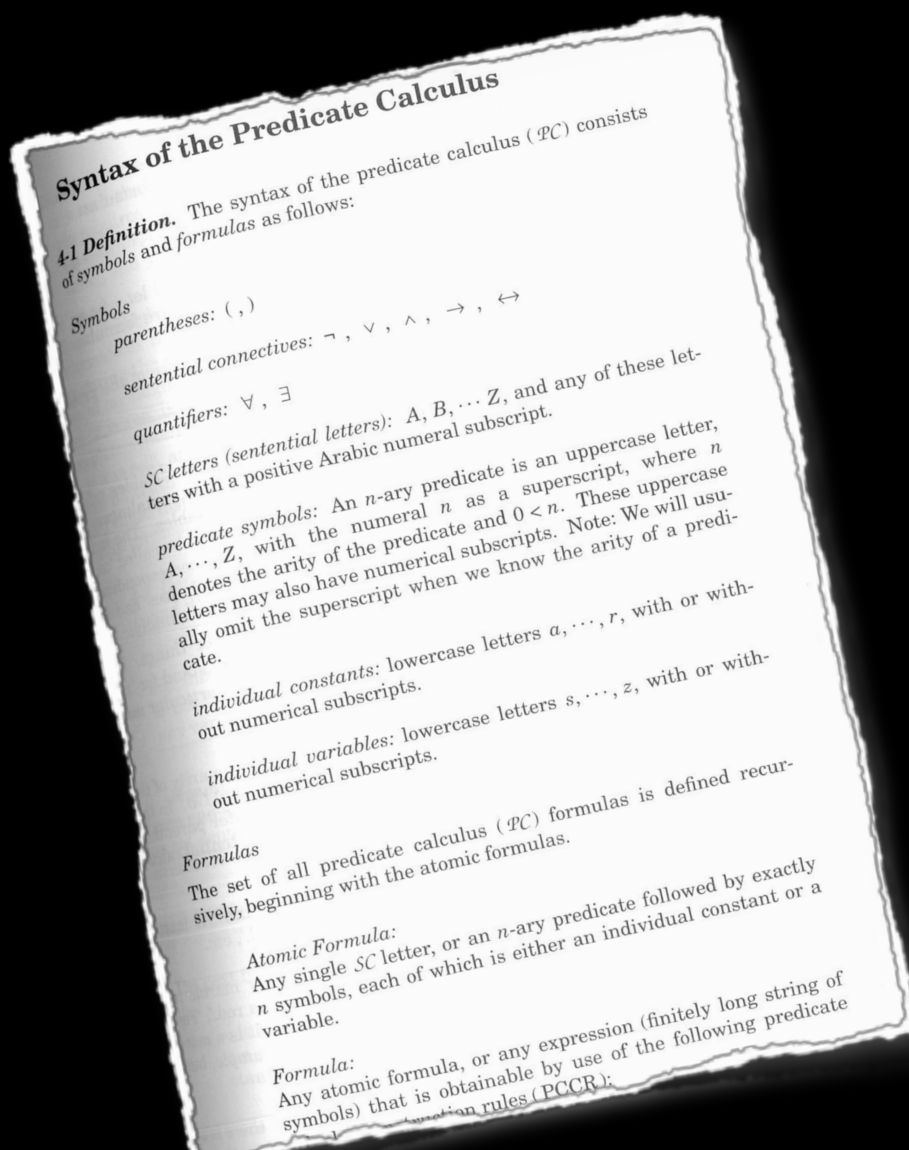
Atomic Formula:
Any single $\mathcal{SC}$ letter, or an n-ary predicate followed by exactly n symbols, each of which is either an individual constant or a variable.

Formula:
Any atomic formula, or any expression (finitely long string of symbols) that is obtainable by use of the following predicate calculus construction rules (PCCR):

Logic, sets, and Recursion

Robert L. Causey

Prolog is syntactic sugar for the predicate calculus.*



Syntax of the Predicate Calculus

4-1 Definition. The syntax of the predicate calculus ($\mathcal{PC}$) consists of symbols and formulas as follows:

Symbols

parentheses: (,)

sentential connectives: ¬ , ∨ , ∧ , → , ↔

quantifiers: ∀ , ∃

SC letters (sentential letters): A, B, ⋯ Z, and any of these letters with a positive Arabic numeral subscript.

predicate symbols: An n-ary predicate is an uppercase letter, A, ⋯, Z, with the numeral n as a superscript, where n denotes the arity of the predicate and $0 < n$. These uppercase letters may also have numerical subscripts. Note: We will usually omit the superscript when we know the arity of a predicate.

individual constants: lowercase letters $a, \cdots, r$, with or without numerical subscripts.

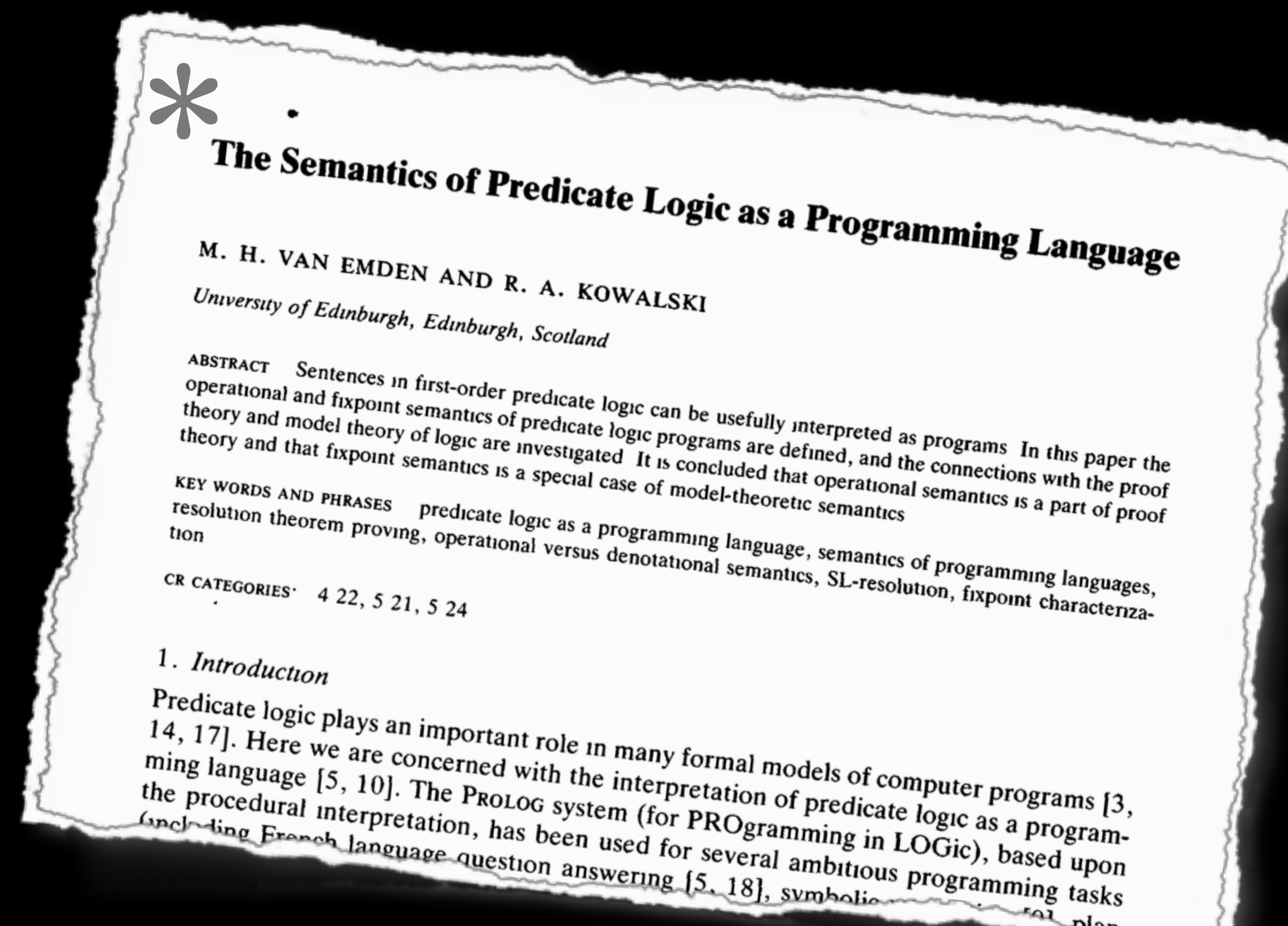
individual variables: lowercase letters $s, \cdots, z$, with or without numerical subscripts.

Formulas

The set of all predicate calculus ($\mathcal{PC}$) formulas is defined recursively, beginning with the atomic formulas.

Atomic Formula:
Any single *SC* letter, or an n-ary predicate followed by exactly n symbols, each of which is either an individual constant or a variable.

Formula:
Any atomic formula, or any expression (finitely long string of symbols) that is obtainable by use of the following predicate



*

The Semantics of Predicate Logic as a Programming Language

M. H. VAN EMDEN AND R. A. KOWALSKI

University of Edinburgh, Edinburgh, Scotland

ABSTRACT Sentences in first-order predicate logic can be usefully interpreted as programs In this paper the operational and fixpoint semantics of predicate logic programs are defined, and the connections with the proof theory and model theory of logic are investigated It is concluded that operational semantics is a part of proof theory and that fixpoint semantics is a special case of model-theoretic semantics

KEY WORDS AND PHRASES predicate logic as a programming language, semantics of programming languages, resolution theorem proving, operational versus denotational semantics, SL-resolution, fixpoint characterization

CR CATEGORIES 4 22, 5 21, 5 24

1. *Introduction*

Predicate logic plays an important role in many formal models of computer programs [3, 14, 17]. Here we are concerned with the interpretation of predicate logic as a programming language [5, 10]. The PROLOG system (for PROgramming in LOGic), based upon the procedural interpretation, has been used for several ambitious programming tasks

Hmmm ≡ Racket ≡ Prolog

↓ ↓ ↓

Turing Machine ≡ (Turing 1937) λ Calculus ≡ (Church (1936), Turing (1937)) Predicate Calculus

0 jumpn 0

((λ (x) (x x)) (λ (x) (x x)))

p :- p

1. Language influences thought.

2. Programs ≡ Data

and self-reference is powerful

3. Computers are powerful.

they change our lives for good and bad

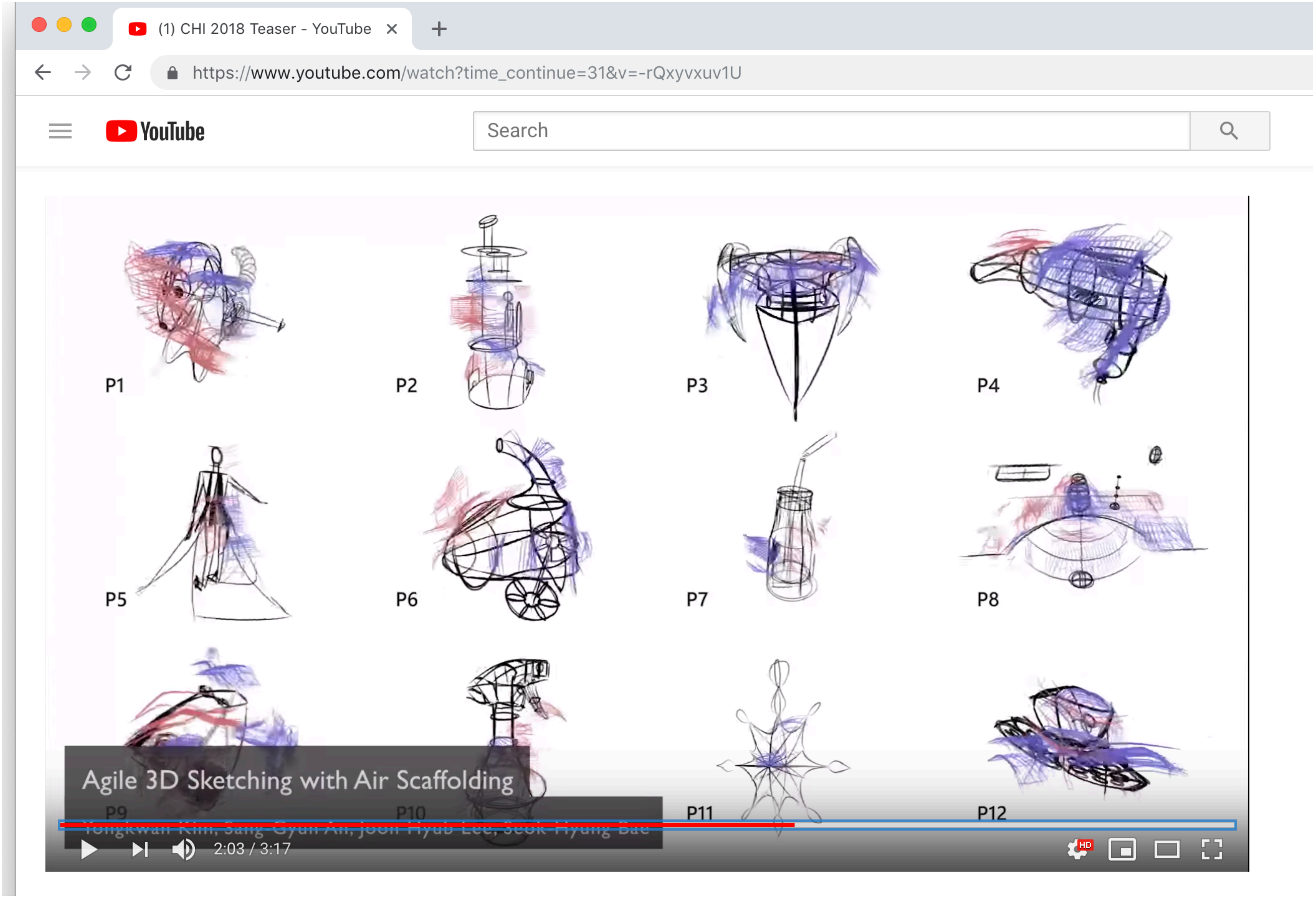
Commentary: Facebook's Algorithm vs. Democracy

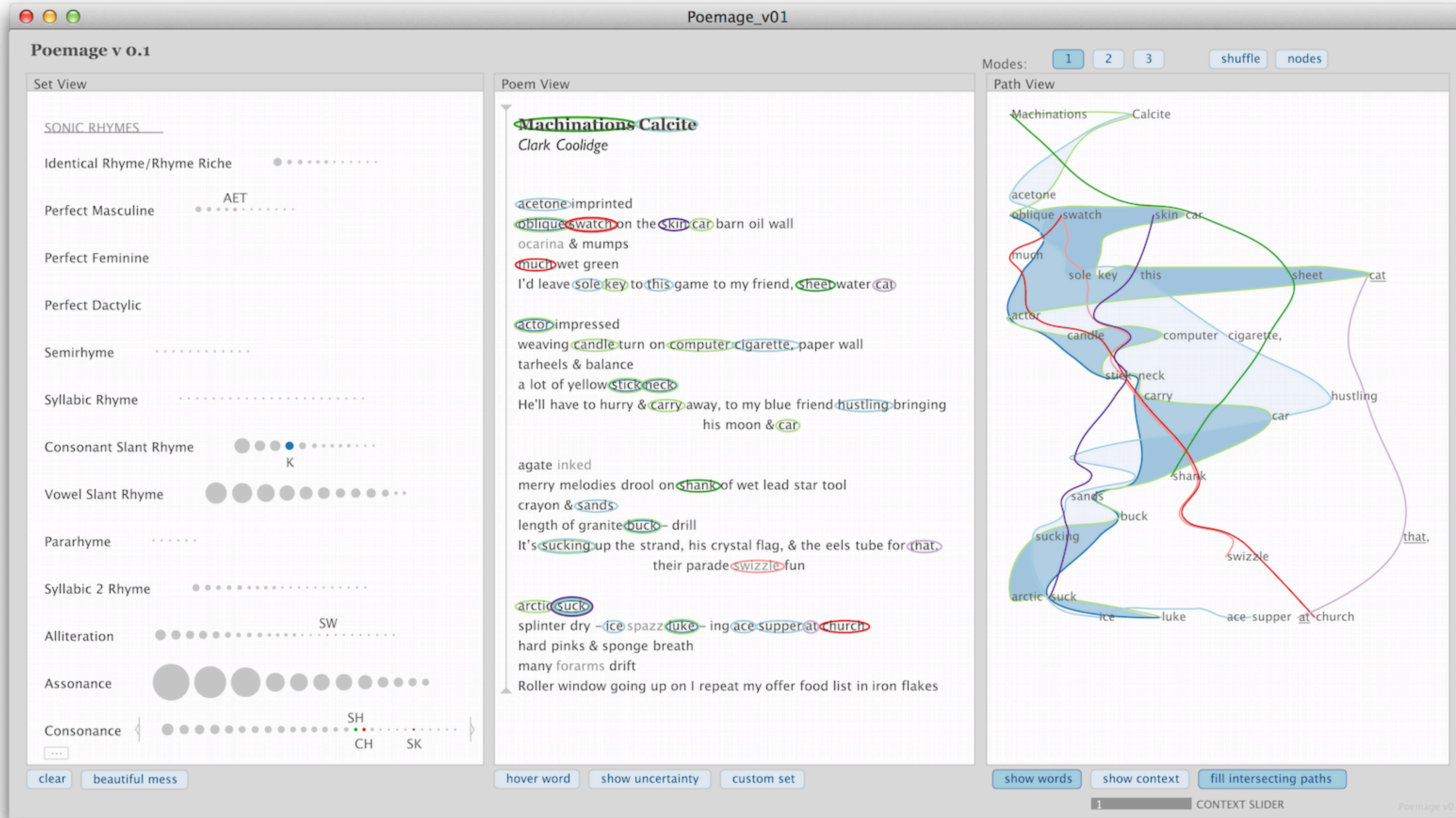
By Cathy O'Neil on Wed, 07 Dec 2016

Over the last several years, Facebook has been participating—unintentionally—in the erosion of democracy.

The social network may feel like a modern town square, but thanks to its tangle of algorithms, it's nothing like the public forums of the past. The company determines, according to its interests and those of its shareholders, what we see and learn on its social network. The result has been a loss of focus on critical national issues, an erosion of civil disagreement, and a threat to democracy itself.

Facebook is just one part—though a large part—of the Big Data economy, one


(1) CHI 2018 Teaser - YouTube
https://www.youtube.com/watch?time_continue=31&v=-rQxyvxuv1U
YouTube
Search
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
Agile 3D Sketching with Air Scaffolding
Yongkwan Kim, Sang-Gyun An, Joon Hyub Lee, Seok-Hyung Bae
2:03 / 3:17



upload.wikimedia.org/wikipedia/commons/7/73/Poemage.png