# Expressing/Enforcing Policies

Vicky Weissman

Joint work with Joseph Halpern and Carl Lagoze

# What is a policy?

- A policy says that under certain conditions an action is permitted or forbidden.
- The ACM digital library "terms of use" document includes the statements
  - "Members may download articles"
  - "Members may not republish articles without explicit consent".

# What is a policy?

- A policy says that under certain conditions an action is permitted or forbidden.
- The ACM digital library “terms of use” document includes the statements
  - “Members may download articles”
  - “Members may not republish articles without explicit consent”.

# What is a policy?

- A policy says that under certain conditions an action is permitted or forbidden.
- The ACM digital library “terms of use” document includes the statements
  - “Members may download articles”
  - “Members may not republish articles without explicit consent”.

# What is a policy?

- A policy says that under certain conditions an action is permitted or forbidden.
- The ACM digital library “terms of use” document includes the statements
  - “Members may download articles”
  - “Members may not republish articles without explicit consent”.

# Goals

- Digital content providers want to write policies regulating access.
  - Want this process to be as easy as possible.
- They want their policies enforced.
  - Want this process to be correct and easy too!

# Recall

Last Friday, Peter Hirtle gave a talk.

- Said why automatic (computer/machine) enforcement is becoming crucial.
- My favorite example:

  The Greek Orthodox Archdiocese of America
- Automatic enforcement is not always possible because policies can be "fuzzy"
  - What counts as fair use?  legal search?  porn?

# But

- For many applications, some/all of the policies are not “fuzzy”
    - E.g., user agreements and other contracts
        - Cornell’s policies for computer use,
        - iTunes policies for who may download, play, and copy songs from their database,
        - tax law, HIPAA, and other federal regulations.
- If the policies aren’t “fuzzy”, how can we state/enforce them?

# Classic Approach

- Write policies in a natural language.
  - E.g., English, Russian, Chinese.
- How do we do enforcement?
  - Ask your favorite person in NLP (Natural Language Processing) to build a translator from natural languages to machine-readable code (e.g., XML).
  - That’s not going to work.

# Problem: Ambiguity

- Consider the policy “every good boy and girl is permitted to have a candy”.
  - We’re assuming policy is concrete; we know the meaning of “good” and “candy”.
  - “good” is an adjective modifying “boy”.
  - Does “good” modify “girl”? If Alice is girl who is not good, may she have candy?

# A Partial Solution

- Big Idea: Restrict the natural language to an unambiguous fragment.
  - E.g., ELi restricts English to simple sentences and if/then statements.
- Problems
  - Decrease usability.
  - Decrease expressive power.
  - How do you know that the restricted language is unambiguous?

# Example

- XrML is a popular XML-based language.
    - Has restricted syntax.
    - But has ambiguity too.
- XrML supports a notion of groups.
    - If Alice is an agent, and Bob is an agent, then Alice and Bob acting together is an agent.

# Relationship Groups/Members

- Suppose that Alice has property $P_A$ and the agent Alice + Bob has property $P_{AB}$.
- What should we infer?
  - Option 1: Nothing
  - Option 2: Alice + Bob has property $P_A$.
  - Option 3: Alice has property $P_{AB}$.

# Relationship Groups/Members

- Suppose that Alice has property $P_A$ and the agent Alice + Bob has property $P_{AB}$.
- What should we infer?
  - Option 1: Nothing
  - Option 2: Alice + Bob has property $P_A$.
  - Option 3: Alice has property $P_{AB}$.
- XrML chooses each option (in different parts of the spec).

# Solution

- To avoid ambiguity, a policy language needs syntax and formal semantics.
  - Syntax describes the symbols in the language.
  - Formal semantics say how the symbols can be combined.
- Can give a language formal semantics by providing a translation to a language that already has it (e.g., first-order logic).

# Example - ELi

- ELi is designed so that statements can be easily translated to first-order logic → no ambiguity.
- Basic facts translate to themselves.
  - Student(Alice) → Student(Alice)
- if/then statements translate to clauses.

agent x; resource y;

if Owns(x,y) and not Frozen(y) then Permitted(x,edit(y));

$$\rightarrow$$

$$\forall x\ \forall y\ (\mathrm{Owns}(x,y) \wedge \neg \mathrm{Frozen}(y) \Rightarrow \mathrm{Permitted}(x, \mathrm{edit}(y)))$$

# Enforceability

- Even if language is unambiguous, it may be hard to reason about.
- E.g., Consider the policies:
  - Alice is a custodian.
  - Custodians may enter the student lounge.
  - Only students may enter the student lounge.
  - Students are permitted to register for courses.
  - May Alice register for courses?

# Enforceability

- Even if language is unambiguous, may be hard to reason about.
- E.g., consider the policies:
  - Alice is a custodian.
  - Custodians may enter the student lounge.
  - Only students may enter the student lounge.
  - Students are permitted to register for courses.
  - May Alice register for courses?  Yes.

# Enforceability

- Even if language is unambiguous, it may be hard to reason about.
- E.g., Consider the statements:
  - Alice is a custodian.
  - Custodians may enter the student lounge.
  - Only students may enter the student lounge.
  - Students are permitted to register for courses.
  - May Alice register for courses?  Yes.

All custodians are students.

# Big Idea

- Restrict language so that we can enforce policies written in it.
- Option 1: Get rid of all negation.
  - If we couldn’t write “anyone who is not a student is not permitted to enter the lounge”, we’d have no trouble.
  - This approach is taken by XrML.
  - Clearly limits the language’s expressive power.

# Option 2: Redefine Negation

- XACML is a popular policy language that
  - is tractable and
  - allows unlimited use of negation.
- But the definition of negation is non-standard.
  - XACML doesn’t assume that a property either holds or it doesn’t.

# Example

- Consider the policies
  - “Members are always permitted to use the pool”, and
  - “Anyone who is not a member is permitted to use the pool on mornings from 6-8 AM.”
- Alice wants to go swim at 6 AM. May she?
- XACML says no unless Alice can prove that she is or is not a member.

# Option 3:
# Allow Some Negation

- Safe stratified Datalog (SSD) is a tractable fragment of first-order logic that allows some use of negation.
- Idea: Restrict policy language so every statement written in it translates to SSD.
- SSD does not allow negation in "then" clauses.
  - Can't say "if…then …is *not* permitted to do…"

# Prohibitions are Important

- Policy writers want to tell you what’s forbidden.
  - Explicit prohibitions decrease liability and clarify the rules.
- Prohibitions allow compliance checking.
  - We can check if hospital policy complies with federal regulation.
- Prohibitions allow us to detect some type of user error.

# Recall Example

- Consider the statements:
  - Alice is a custodian.
  - Custodians may enter the student lounge.
  - Only students may enter the student lounge.
  - Students are permitted to register for courses.
- We can detect implied facts and ask writer “should the fact be added or should policies be revised?”

All custodians are students.

# Option 3: Allow Some Prohibitions

- Recall: A fragment in ELi is of the form:

  (not) property(entity, …, entity)
- Example: “not Student(Alice)” is a fragment, Student is the property.
- A property P is *mixed* in a set S of statements if
  - P is in both an “if” and “then” clause or
  - P is in a positive and negative fragment.

# Example

- Let S include:
  - if Student(Bob) then Permitted(Bob, sing)
  - if not Student(Alice) then not Permitted(Alice, sing)
  - agent x;

    if Over21(x) then Adult(x)
  - if Adult(Alice) then Permitted(Alice, vote)
- Student, Permitted, and Adult are mixed.

# Example

- Let S include:
  - if Student(Bob) then Permitted(Bob, sing)
  - if not Student(Alice) then not Permitted(Alice, sing)
  - agent x;

    if Over21(x) then Adult(x)
  - if Adult(Alice) then Permitted(Alice, vote)
- Student, Permitted, and Adult are mixed.

# Example

- Let S include:
  - if Student(Bob) then Permitted(Bob, sing)
  - if not Student(Alice) then not Permitted(Alice, sing)
  - agent x;

    if Over21(x) then Adult(x)
  - if Adult(Alice) then Permitted(Alice, vote)
- Student, Permitted, and Adult are mixed.

# Example

- Let S include:
  - if Student(Bob) then Permitted(Bob, sing)
  - if not Student(Alice) then not Permitted(Alice, sing)
  - agent x;

    if Over21(x) then Adult(x)
  - if Adult(Alice) then Permitted(Alice, vote)
- Student, Permitted, and Adult are mixed.

# The Rule

- Let S be the set of statements for an app.
- Let S+ be S with the prohibitions removed.
- Let S- be S with the permissions removed.
- If every statement in S+ mentions at most one instance of a mixed property (and same with S-) then we can reason about the policies.
- Bottom line: As we go for more expressive power and tractability, we reduce usability.

# Big Picture

- In developing a policy language, you have to make trade-offs between
  - Usability.
  - Expressive Power.
  - Tractability.
- Each language makes different choices.
- Apps chose the language that’s right for them.
- Open question: Interoperability.