Knowledge Representation, Reasoning, and Privacy

Drexel University

Marcello Balduccini and Sarah Kushner

Partially supported by

Introduction and Privacy

Problem

We need to reason about privacy in a systematic, principled way.

Key questions that we will attempt to answer:

- What is privacy?
- When can we claim that privacy has been protected?
- Which obstacles need to be overcome?
- Which techniques can be used?

Motivating Example

- Can a university find which students practice any kind of religion?
  - … without the students' knowledge?
  - Information is not directly available
  - Information could be used to discriminate
    - Religion is protected by law
  - This would be an invasion of privacy!

Motivating Example

How can they do it?

- Swab pews in interfaith church
- Look up DNA in online gene database
  - List #1 - Last names of people whose DNA was found in the church
    - Smith
  - List #2 - university students

Motivating Example

- Through surnames, a genome can be tracked and identified.
- Genome datasets are commonly released without identifiers.
- However, last names can be recovered from the data by profiling the genome in a specific way.

http://data2discovery.org/dev/wp-content/uploads/2013/05/Gymrek-et-al.-2013-Genome-Hacking-Science-2013-Gymrek-321-4.pdf

Required Reading

- Identifying Personal Genomes by Surname Inference
  - http://data2discovery.org/dev/wp-content/uploads/2013/05/Gymrek-et-al.-2013-Genome-Hacking-Science-2013-Gymrek-321-4.pdf

Privacy

Definition: the ability of an individual or group to seclude information about themselves

adapted from Wikipedia: https://en.wikipedia.org/wiki/Privacy

Privacy

- Protection of privacy
  - HIPAA exists to protect people by keeping certain information private.
    - "The primary goal of the law is to make it easier for people to keep health insurance, protect the confidentiality and security of healthcare information and help the healthcare industry control administrative costs."

HIPAA Purpose

- HIPAA - Health Insurance Portability and Accountability Act
- Purpose is to protect health information by law by withholding identifiers from a dataset
- Dataset will then be "anonymized"

http://cphs.berkeley.edu/hipaa/hipaa18.html

HIPAA Identifiers

1. Names
2. All geographical subdivisions smaller than a State (street address, city, county, precinct, zip code)
3. All elements of dates (except year) for dates directly related to an individual
4. Phone numbers
5. Fax numbers
6. Electronic mail addresses
7. Social Security numbers
8. Medical record numbers
9. Health plan beneficiary numbers
10. Account numbers
11. Certificate/license numbers
12. Vehicle identifiers and serial numbers, including license plate numbers
13. Device identifiers and serial numbers
14. Web Universal Resource Locators (URLs)
15. Internet Protocol (IP) address numbers
16. Biometric identifiers, including finger and voice prints
17. Full face photographic images and any comparable images
18. Any other unique identifying number, characteristic, or code (note this does not mean the unique code assigned by the investigator to code the data)

HIPAA Limited Data Use Agreement

- A few identifiers remain in the dataset (i.e. only dates and zip codes)
- Still, dataset must be protected as if it were fully identified
- Researchers must make sure subjects agree to having information recorded and stored
- **Often, the risks aren't fully known**

http://www.hopkinsmedicine.org/institutional_review_board/hipaa_research/limited_data_set.html

HIPAA Shortcomings (1)

- Researcher publishes an article
- 90% of residents in a certain zip code have the same medical problem or deficiency
- Real estate values in the area go down
- From an additional source, it is known that air/water problems are responsible for causing this kind of medical condition
- What other consequences come from this connection? What if the information published wasn't accurate?

HIPAA Shortcomings (2)

- As long as researchers follow the law, Internal Review Boards will approve the request.
- The burden to protect themselves falls on the subjects.
- How is a subject expected to know what risks come from agreeing to release data?

Actual Example: A HIPAA Failure

- Massachusetts Group Insurance Commission released "anonymized data"
- Every hospital visit of state employees
- The state removed name, address, and Social Security number
- Researchers were able to find the governor's private records

http://arstechnica.com/tech-policy/2009/09/your-secrets-live-online-in-databases-of-ruin/

Actual Example: Process

- "Only six people in Cambridge shared his birth date, only three of them men, and of them, only he lived in his ZIP code."
- World Knowledge: Knew the governor lived in Cambridge
- Sources:
  - Voter roll: name, address, ZIP code, birth date, and sex of every voter
  - Released anonymized data: Hospital visits

Actual Example: Conclusions

- The health records included:
  - diagnoses
  - prescriptions
- Even worse, 87 percent of all Americans could be uniquely identified using only three bits of information: ZIP code, birthdate, and sex.

Motivating Example Revisited

- Who practices a religion at Drexel?
- People can choose not to have their gene data released to the public in the database.
  - Smith did not want his name publicly available.
- Their name cannot be found using the genetic database.
  - Only the name Schmidt was found for that DNA
- So no names from List #2 (student list) match List #1 (list of people who attended interfaith church)

Motivating Example Revisited

- Solution
  - World Knowledge: Some people have their last names changed when immigrating to the United States.
  - Connecting the World Knowledge to the two lists can lead to a breach in privacy even when measures are taken to protect it.

Intro to Knowledge Representation, Privacy Examples

Knowledge Representation & Reasoning (KR)

Definition: representing information about the world in a form that a computer system can utilize to solve complex tasks

- KR aims to describe knowledge and answer queries.
- Today's KR paradigm:
  - Describe knowledge.
  - Describe reasoning.
- Let inference engine algorithms draw conclusions (provably correct).

Introduction to Logic Symbols

Sets

x ∈ A : x is an element of A

x ∉ A : x is not an element of A

A ⊂ B : A is a proper subset of B; every element in A is in B and B has at least one element not in A

A ⊆ B : A is a subset of B; every element in A is also in B

A∩B : A intersect B; every element in both A and B

A∪B : A union B; every element in A or B

Propositional Logic

p∨q : p or q

p∧q : p and q

∀ p : for all p ...

∃ p : there exists a p ...

∄ p : there does not exist a p

p ⊨ q : p entails q

p ⊢ q : q is provable from p

¬p : not p

p⇔q : true only if both p and q are false, or both p and q are true

p⇒q : p implies q

Privacy Assessment Based on KR

$$I \cup KB \stackrel{?}{\models} S$$

`I` - information released

`KB` - knowledge base (world knowledge)

`S` - secret

Can information from both `I` and `KB` be connected to reveal a secret?

Solving the Motivating Example

$$I \cup KB \models S$$

- Information:
  - DNA in church found belonging to Schmidt
  - student with the last name Smith
- Knowledge Base:
  - Last name changed from Schmidt to Smith when immigrating to the United States
- Secret:
  - Smith attends interfaith church

Privacy

- Invasion of privacy
  - Even with HIPAA in place, information can still be found.
  - It isn't enough
- Need Knowledge Representation to show us the potential inferences that can be made using data.

Public Information

- On top of the privacy concerns centered around the connections of data, people are willing to give out information about themselves
  - Twitter
  - Facebook
  - Any social media

http://repository.cmu.edu/cgi/viewcontent.cgi?article=1066&context=jpc

Required Reading

- Privacy Violations Using Microtargeted Ads: A Case Study
  - http://repository.cmu.edu/cgi/viewcontent.cgi?article=1066&context=jpc

Public Information

- Some information is publicly available because it is not protected by law
  - Gene and Genealogy Databases
  - Public records (i.e. phone book)
- Public information can be a great source of world knowledge.

Examples of KR

- Tweety Bird Problem
- Lin's Suitcase Problem
- Nixon Diamond Problem

http://arxiv.org/pdf/1304.2361.pdf

http://plato.stanford.edu/entries/logic-ai/

https://en.wikipedia.org/wiki/Nixon_diamond

Required Reading

- Rational Nonmonotonic Reasoning
  - http://arxiv.org/pdf/1304.2361.pdf

Tweety Bird Problem

- Birds fly.
- Penguins are birds. Penguins do not fly.
- Tweety is a penguin.
- Does Tweety fly?
- No.

Non-Monotonic Reasoning (NMR)

- As we add knowledge, conclusions change.

- Birds fly.
- Penguins are birds.
- Tweety is a penguin.
- Does Tweety fly?
  - Yes.

- Penguins do not fly.
- Does Tweety fly?
  - No.

Lin's Suitcase Problem

- Suitcase has two locks.
- Suitcase is open iff both locks are open.
- Left lock of suitcase is open.
- Is the suitcase open?
  - No.
- Open the right lock
- Is the suitcase open?
  - No.
- The default conclusion is that the suitcase remains closed (wrong)

Nixon Diamond Problem

- Nixon is both a Quaker and Republican.
- Quakers are anti-war.
- Republicans are pro-war.
- Both war supporters and opponents are vocal about their position.

- Is Nixon pro-war or anti-war?
  - Unknown
- Is Nixon vocal about his position?
  - Yes

Ontologies and Answer Set Programming

Languages

- Ontologies
- LP - ASP
- Action Languages

Ontologies

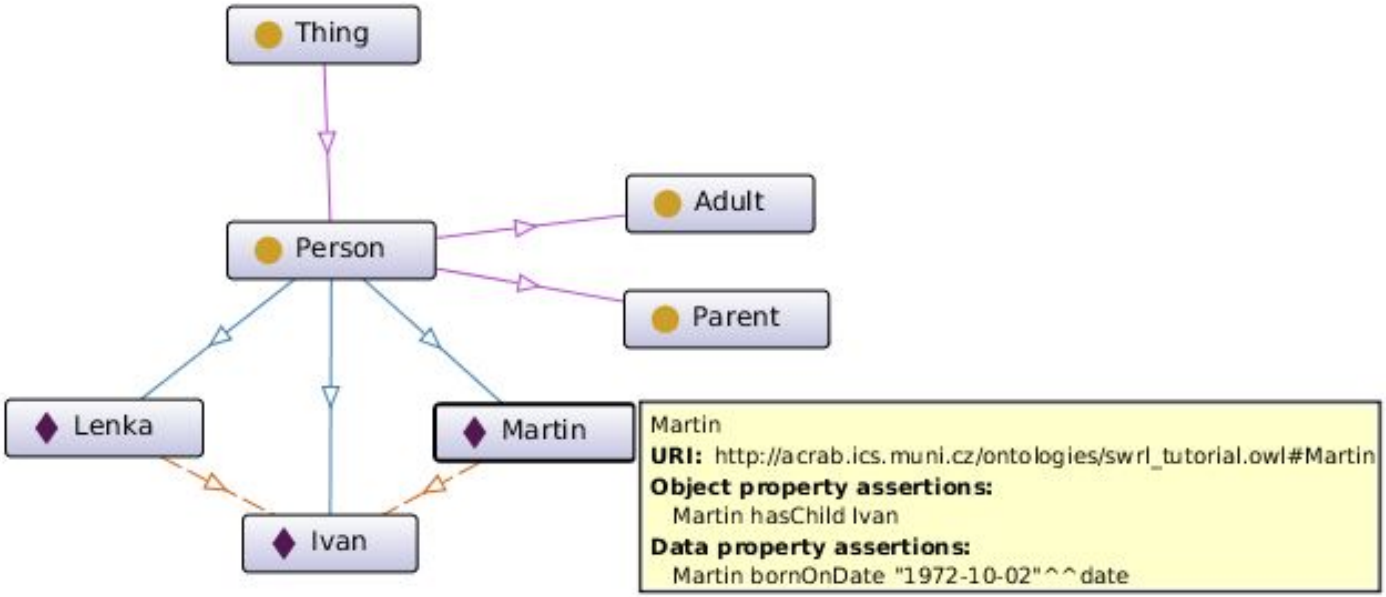
Definition: An ontology is a formal specification of a shared conceptualization

- hierarchical organization of data
- demonstrates how terms are related to each other

http://dior.ics.muni.cz/~makub/owl/

http://semanticweb.org/wiki/Ontology

Ontologies


Thing
Person
Adult
Parent
Lenka
Martin
Ivan
Martin
URI: http://acrab.ics.muni.cz/ontologies/swrl_tutorial.owl#Martin
Object property assertions:
Martin hasChild Ivan
Data property assertions:
Martin bornOnDate "1972-10-02"^^date

Required Reading

- What Are Ontologies, and Why Do We Need Them?
  - http://www.csee.umbc.edu/courses/771/current/papers/chandrasekaranetal99.pdf

Answer Set Programming

Declarative Languages – The Idea

- A declarative program (DP) is a collection of statements describing objects of a domain and their properties.
- The syntax defines legal statements of the language
- The semantics:
  - Defines the notion of a model of a DP (i.e. a possible state of the world compatible with the DP statements), and
  - Characterizes the collection of valid consequences of a program.
- Various tasks are reduced to finding models or computing consequences of a DP.
- Models are found and/or consequences are computed by general purpose reasoning algorithms often called *solvers* or *inference engines*.

Answer Set Programming

- Answer Set Programming (ASP) is a declarative language
  - Simple syntax
  - Intuitive semantics
  - Fast implementations
  - Solves many problems elegantly and efficiently, including reasoning about actions and change

Note:
ASP $\equiv$ Answer Set Programming $\equiv$ Answer Set Prolog $\equiv$ A-Prolog

ASP – Basic Terminology

A *signature* is a four-tuple

$$\Sigma = \langle \mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{V} \rangle$$

of disjoint sets.

- $\mathcal{O}$: *object symbols (or constants)*
- $\mathcal{F}$: *function symbols*
- $\mathcal{P}$: *predicate (or relation) symbols*
- $\mathcal{V}$: *(object) variables*

Predicate symbols name relations between the domain's objects.

Each function and predicate symbol is associated with an *arity*, i.e. an integer indicating the number of symbol's parameters.
Often, arity is determined from the context.

ASP – Basic Terminology

Examples:

$$\Sigma_1 = \langle \{john, mary\}, \{car(\cdot)\}, \{married(\cdot,\cdot)\}, \{X, Y\} \rangle$$

- *Object symbols (or constants)*: $\{john, mary\}$
- *Function symbols*: $\{car(\cdot)\}$
- *Predicate (or relation) symbols*: $\{married(\cdot,\cdot)\}$
- *(Object) variables*: $\{X, Y\}$

Arities:
$car(\cdot)$: 1
$married(\cdot,\cdot)$: 2

ASP – Basic Terminology

Terms (over $\Sigma = \langle \mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{V} \rangle$) are defined as follows:

- Variables and object symbols are terms.
- If $t_1, \ldots, t_n$ $(n \geq 1)$ are terms and $f \in \mathcal{F}$ then $f(t_1, \ldots, t_n)$ is a term.

Terms not containing variables are called *propositional* or *ground*.
Terms are used to name the domain's objects, e.g. $car(\cdot)$ names cars.

Terms or not terms??
$\Sigma_1 = \langle \{john, mary\}, \{car(\cdot)\}, \{married(\cdot, \cdot)\}, \{X, Y\} \rangle$

- $john$
- sam
- Y
- $car(john)$
- $car(john, mary)$
- $car(X)$
- $car(car(mary))$
- $married(john, mary)$

ASP – Basic Terminology

Given $\Sigma = \langle \mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{V} \rangle$), an *atom* is an expression of the form

$$p(t_1, \ldots, t_n)$$

where $p \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms.

If t_i's are ground then $p(t_1, \ldots, t_n)$ says that

"*Objects denoted by* $t_1, \ldots, t_n$ *satisfy property* p."

Example: $p(a, b)$ says "objects x, y denote property p".
With words: *prime*(3) says "3 is prime".

A *literal* is an atom, $p(t_1, \ldots, t_n)$, or its negation, $\neg p(t_1, \ldots, t_n)$.

ASP – Basic Terminology

Atoms or not atoms??
$\Sigma_1 = \langle \{john, mary\}, \{car(\cdot)\}, \{married(\cdot, \cdot)\}, \{X, Y\} \rangle$

- $john$
- $car(john)$
- $married(john, mary)$
- $married(john, car(john))$
- $married(mary, john(X))$
- $car(married(john, mary))$

ASP – Basic Terminology

Terms and atoms look alike. How do I tell them apart?

Atoms (and literals) represent statements that may be (intuitively) true or false.
Terms do not.

Term or atom?? $\Sigma_1 = \langle \{john, mary\}, \{car(\cdot)\}, \{married(\cdot,\cdot)\}, \{X, Y\} \rangle$

- $john$
- $car(john)$
- $married(john, mary)$
- $car(married(john, mary))$

The Syntax of ASP

A *rule* over signature Σ (or of signature Σ) is an expression of the form:

$$l_0 \text{ or } \ldots \text{ or } l_i \leftarrow l_{i+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n.$$

where l_i's are literals of Σ. "*not*" is called *default negation*. "*or*" is *disjunction*.

The left-hand side of a rule ($l_0 \text{ or } \ldots \text{ or } l_i$) is called the *head* and the right-hand side ($l_{i+1}, \ldots, l_m, \text{ not } l_{m+1}, \ldots, \text{ not } l_n$) the *body*. $l_{i+1}, \ldots, l_m$ is the *positive part* of the body. $\text{not } l_{m+1}, \ldots, \text{not } l_n$ is the *negative part* of the body.

Informal meaning: "*if you believe* $l_{i+1}, \ldots, l_m$ *and has no reason to believe* $l_{m+1}, \ldots, l_n$*, then believe one of* $l_0, \ldots, l_i$*.*"

The Syntax of ASP

Examples:

$married(john, mary) \leftarrow married(mary, john).$

$married(X, Y) \leftarrow married(Y, X).$

$starts(X) \leftarrow turned\_key(X),\ not\ broken(engine(X)).$
"*if X's key is turned and I have no reason to believe that X's engine is broken, then I believe that X will start*"

The Syntax of ASP

Both the head and the body can be empty.

A rule with the empty head is often called a *constraint* or *denial*.

A rule with the empty body is called a *fact* and written as

$$l_0 \mathit{ or } \ldots \mathit{ or } l_i.$$

A *program* Π (sometimes called a knowledge base) is a pair $\langle \Sigma, R \rangle$ where Σ is a signature and R is a set of rules over Σ.

The Syntax of ASP

$\Sigma_2 = \langle \{john, mary, ted\}, \{car(\cdot)\}, \{married(\cdot,\cdot), \cdot \neq \cdot\}, \{X, Y\} \rangle$

Example

$married(john, mary).$
$married(X, Y) \leftarrow married(Y, X).$
$\neg married(mary, ted) \leftarrow married(mary, john).$
$\leftarrow married(X, X).$

The Syntax of ASP

In common use, people often denote a program by its rules and leave the signature implicitly defined.

Example

Given "program":

$$p(a) \leftarrow q(b).$$
$$r(t(c,d)) \leftarrow p(a).$$

Inferred signature: $\langle \{a, b, c, d\}, \{t\}, \{p, q, r\}, \{\} \rangle$.

Grounding

Object, function and predicate symbols of Σ are denoted by identifiers starting with small letters.

Variables are identifiers starting with the capital letters.

A *ground instantiation* (or *grounding*) of a rule is obtained by replacing every variable by a ground term of Σ. All occurrences of a variable are replaced by the same term.

A rule with variables is viewed as the set of all of its ground instantiations.

This means that it is enough to define the semantics of ground programs.

Grounding

$\Sigma_2 = \langle \{john, mary, ted\}, \{car(\cdot)\}, \{married(\cdot,\cdot), \cdot \neq \cdot\}, \{X, Y\} \rangle$

Examples

- $\neg married(X, ted) \leftarrow married(X, john).$
  is grounded as:
  $\neg married(john, ted) \leftarrow married(john, john).$
  $\neg married(mary, ted) \leftarrow married(mary, john).$
  $\neg married(ted, ted) \leftarrow married(ted, john).$
  $\neg married(car(john), ted) \leftarrow married(car(john), john).$
  $\neg married(car(car(john)), ted) \leftarrow$
  $married(car(car(john)), john).$
  ...
- $married(X, Y) \leftarrow married(Y, X).$ is grounded as:
  $married(john, mary) \leftarrow married(mary, john).$ etc.

Informal Semantics of ASP

- Ground program Π can be viewed as a specification for the sets of beliefs, called *answer sets* or *stable models*, to be held by a rational reasoner associated with Π.
- Such sets are represented by collection of ground literals.
- The informal semantics of a program is obtained by:
  - Complying with informal reading

    "*if you believe* $l_{i+1}, \ldots, l_m$ *and has no reason to believe* $l_{m+1}, \ldots, l_n$*, then believe one of* $l_0, \ldots, l_i$*.*"
  - Satisfy the *rationality principle*, which says:

    "*Believe nothing you are not forced to believe*"

Informal Semantics of ASP

Example

Program:

$$\Pi_0 \left\{ \begin{array}{lcl} p(a) & \leftarrow & not\ q(a). \\ p(b) & \leftarrow & not\ q(b). \\ q(a). & & \end{array} \right.$$

has one answer set $S_0 = \{q(a), p(b)\}$.

Meaning of S_0: $q(a)$ is true in S_0 while, say, $q(b)$ is unknown.

Informal Semantics of ASP

Example

Program $\Pi_1 = \Pi_0 \cup \{\neg q(b) \leftarrow \ not\ q(b).\}$, i.e.

$$\Pi_1 \begin{cases} p(a) & \leftarrow & not\ q(a). \\ p(b) & \leftarrow & not\ q(b). \\ q(a). & & \\ \neg q(b) & \leftarrow & not\ q(b). \end{cases}$$

has one answer set $S_1 = \{q(a), \neg q(b), p(b)\}$.
$q(b)$ is false in S_1 while $p(a)$ is still unknown.

Informal Semantics of ASP

Example

Program

$$\Pi_2 \begin{cases} p(a) & \leftarrow \quad not\ q(a). \\ p(b) & \leftarrow \quad p(b). \end{cases}$$

has one answer set $S_2 = \{p(a)\}$.

$p(b)$ is unknown because of the rationality principle "*Believe nothing you are not forced to believe*".

More Notation and Terminology

Given a rule r:

$$l_0 \; or \; \ldots \; or \; l_i \leftarrow l_{i+1}, \ldots, l_m, \; not \; l_{m+1}, \ldots, \; not \; l_n.$$

we define:

$$head(r) = \{l_0, \ldots, l_i\}$$

$$pos(r) = \{l_{i+1}, \ldots, l_m\}$$

$$neg(r) = \{l_{m+1}, \ldots, l_n\}$$

Formal Semantics: Satisfaction of Rules

A ground set S of literals *satisfies* a rule r (S *is closed under* r) if one of the following conditions hold:

$$pos(r) \not\subseteq S$$

$$neg(r) \cap S \neq \emptyset$$

$$head(r) \cap S \neq \emptyset$$

Example

Rule r is $p(a) \leftarrow q(b), \neg t(c).$ $\quad S = \{\neg p(a), q(b), \neg t(c)\}$

Let us check if S satisfies r.

$$pos(r) = \{q(b), \neg t(c)\} \quad neg(r) = \{\ \} \quad head(r) = \{p(a)\}$$

Check the conditions:

$$\{q(b), \neg t(c)\} \subset \{\neg p(a), q(b), \neg t(c)\},$$

$$\emptyset \cap \{\neg p(a), q(b), \neg t(c)\} = \emptyset, \text{and}$$

$$\{p(a)\} \cap \{\neg p(a), q(b), \neg t(c)\} = \emptyset,$$

every satisfiability condition fails. Therefore, S does *not* satisfy r.

Satisfaction of Rules: More Examples

A ground set S of literals *satisfies* a rule r (*is closed under* r) if one of the following conditions hold:

$$pos(r) \not\subseteq S$$

$$neg(r) \cap S \neq \emptyset$$

$$head(r) \cap S \neq \emptyset$$

Rule r is $p(a) \leftarrow q(b), \neg t(c)$.

$S_1 = \{\ \}$
$S_2 = \{p(a)\}$
$S_3 = \{p(b)\}$
$S_4 = \{t(c)\}$
$S_5 = \{p(a), q(b), \neg t(c)\}$

Defining Answer Sets – Part 1

Let program Π consist of rules of the form (*not*-free rules):

$$l_0 \; or \; \ldots \; or \; l_i \leftarrow l_{i+1}, \ldots, l_m.$$

Definition

An *answer set* of Π is a consistent set S of ground literals such that:

- S is closed under the rules of Π, and
- S is minimal i.e. no proper subset of S satisfies the rules of Π.

Note: a program may have multiple answer sets, representing alternative and equally possible sets of beliefs.
A program may also have no answer sets, meaning that it is impossible to form a consistent set of beliefs that satisfies all rules.

Examples

- $p(a) \leftarrow \neg p(b).$ $\qquad \neg p(a).$

  $A = \{\neg p(a)\}$
- $p(b) \leftarrow \neg p(a).$ $\qquad \neg p(a).$

  $A = \{\neg p(a), p(b)\}$
- $p(b) \leftarrow \neg p(a).$ $\qquad p(b) \leftarrow p(a).$

  $A = \{\quad\}$
- $p(a)\ or\ \ p(b).$

  $A_1 = \{p(a)\}$ $\qquad A_2 = \{p(b)\}$
- $p(a)\ or\ \ p(b).$ $\qquad \leftarrow p(a).$

  $A = \{p(b)\}$
- $p(a).$ $\qquad \leftarrow p(a).$

  NO ANSWER SET

Defining Answer Sets – Part 2

Let Π be an arbitrary program and S be a consistent set of ground literals.

Definition

The reduct of Π with respect to S, denoted by Π^S, is the program obtained from Π by:

1. Removing all rules containing *not* l such that $l \in S$;
2. Removing all other premises containing "*not*".

Example

$S = \{q(a), p(b)\}$

Π	Π^S
$p(a) \leftarrow$ *not* $q(a)$.	
$p(b) \leftarrow$ *not* $q(b)$.	$p(b)$.
$q(a)$.	$q(a)$.

Reduct: Examples

Recall:

1. Removing all rules containing *not* l such that $l \in S$;
2. Removing all other premises containing "*not*".

$S = \{q(a)\}$

Π	Π^S
$p(a) \leftarrow$ *not* $q(a)$.	
$q(a) \leftarrow \neg r(c)$, *not* $p(a)$.	$q(a) \leftarrow \neg r(c)$.
$p(b) \leftarrow p(b)$.	$p(b) \leftarrow p(b)$.

$S = \{p(a), p(b)\}$

Π	Π^S
$r(a) \leftarrow$ *not* $q(a)$.	$r(a)$.
$q(a) \leftarrow \neg r(c)$, *not* $p(a)$.	
$p(b) \leftarrow p(b)$.	$p(b) \leftarrow p(b)$.

Defining Answer Sets – Part 2

Let Π be an arbitrary program and S be a consistent set of ground literals.

Definition

S is an *answer set* of Π if-and-only-if S is an answer set of Π^S.

Example

$S = \{q(a), p(b)\}$

Π	Π^S
$p(a) \leftarrow \ not\ q(a).$	
$p(b) \leftarrow \ not\ q(b).$	$p(b).$
$q(a).$	$q(a).$

$\{q(a), p(b)\}$ is an answer set of Π.

Answer Sets: Examples

Recall: S is an *answer set* of Π if-and-only-if S is an answer set of Π^S.

$S = \{q(a)\}$

Π	Π^S	Answer Set of Π^S
$p(a) \leftarrow \ not\ q(a).$		
$q(a) \leftarrow \neg r(c),\ not\ p(a).$	$q(a) \leftarrow \neg r(c).$	$\{\}$
$p(b) \leftarrow p(b).$	$p(b) \leftarrow p(b).$	

$S = \{p(a), p(b)\}$

Π	Π^S	Answer Set of Π^S
$r(a) \leftarrow \ not\ q(a).$	$r(a).$	
$q(a) \leftarrow \neg r(c),\ not\ p(a).$		$\{r(a)\}$
$p(b) \leftarrow p(b).$	$p(b) \leftarrow p(b).$	

Answer Sets: Examples

Recall: S is an *answer set* of Π if-and-only-if S is an answer set of Π^S.

$S = \{p(a)\}$

Π	Π^S	Answer Set of Π^S
$p(a) \leftarrow \ not\ q(a).$	$p(a).$	
$q(a) \leftarrow \neg r(c),\ not\ p(a).$		$\{p(a)\}$
$p(b) \leftarrow p(b).$	$p(b) \leftarrow p(b).$	

Examples

- $\Pi_0 = \{p(a) \leftarrow \ not\ p(a).\}$
  NO ANSWER SET

- $\Pi_1 = \{p(a) \leftarrow \ not\ p(b). \qquad p(b) \leftarrow \ not\ p(a).\}$
  $A_1 = \{p(a)\} \qquad A_2 = \{p(b)\}$

- $\Pi_2 = \Pi_1 \cup \{\leftarrow p(b).\}$
  $A = \{p(a)\}$

- $\Pi_3 = \Pi_2 \cup \{\neg p(a).\}$
  NO ANSWER SET

Examples

$married(john, mary).$
$married(X, Y) \leftarrow married(Y, X).$
$\leftarrow married(X, X).$

Signature?

Examples

$married(john, mary).$
$married(X, Y) \leftarrow married(Y, X).$
$\leftarrow married(X, X).$

Signature: $\langle \{john, mary, \}, \{\}, \{married\}, \{X, Y\} \rangle$

Grounding?

– Try on your own –

Examples

$married(john, mary).$
$married(X, Y) \leftarrow married(Y, X).$
$\leftarrow married(X, X).$

Signature: $\langle \{john, mary\}, \{\}, \{married\}, \{X, Y\} \rangle$

Grounding:
$married(john, mary).$
$married(john, mary) \leftarrow married(mary, john).$
$married(mary, john) \leftarrow married(john, mary).$
$married(john, john) \leftarrow married(john, john).$
$married(mary, mary) \leftarrow married(mary, mary).$
$\leftarrow married(john, john).$
$\leftarrow married(mary, mary).$

Answer set(s)? – Try on your own –

Examples

Signature: $\langle \{john, mary\}, \{\}, \{married\}, \{X, Y\} \rangle$

Program:

$married(john, mary).$
$married(john, mary) \leftarrow married(mary, john).$
$married(mary, john) \leftarrow married(john, mary).$
$\leftarrow married(john, john).$
$\leftarrow married(mary, mary).$

Answer set(s):

Fixpoint construction of S. Steps:

1. $S_1 = \{\}$
2. Which rules are satisfied by S_1?
   $S_2 = \{married(john, mary)\} \cup S_1$
3. $S_3 = \{married(mary, john)\} \cup S_2$

No other rule can be satisfied. We have reached a *fixpoint*.
Therefore, $S = S_4$ is an answer set of the program.

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, X) \leftarrow \ not\ married(john, X).$

Signature?

– Try on your own –

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, X) \leftarrow \ not\ married(john, X).$

Signature: $\langle \{john, mary, ted\}, \{\}, \{married\}, \{X\} \rangle$

Grounding?

– Try on your own –

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, X) \leftarrow \ not\ married(john, X).$

Signature: $\langle \{john, mary, ted\}, \{\}, \{married\}, \{X, X\} \rangle$

Grounding:
$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow \ not\ married(john, john).$
$\neg married(john, mary) \leftarrow \ not\ married(john, mary).$
$\neg married(john, ted) \leftarrow \ not\ married(john, ted).$

Answer set(s)? We must guess a possible answer set and use the reduct.

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow \ not\ married(john, john).$
$\neg married(john, mary) \leftarrow \ not\ married(john, mary).$
$\neg married(john, ted) \leftarrow \ not\ married(john, ted).$

Answer set(s): we must guess a possible answer set and use the reduct.
Lets try:
$S = \{married(john, mary), \neg married(mary, ted), \neg married(john, john),$
$\neg married(john, mary), \neg married(john, ted)\}$

Reduct:

– Try on your own –

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow \ not\ married(john, john).$
$\neg married(john, mary) \leftarrow \ not\ married(john, mary).$
$\neg married(john, ted) \leftarrow \ not\ married(john, ted).$

Answer set(s): we must guess a possible answer set and use the reduct.
Lets try:
$S = \{married(john, mary), \neg married(mary, ted), \neg married(john, john),$
$\neg married(john, ted)\}$

Reduct: Step 1: Remove all rules containing $not\ l$ such that $l \in S$
$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow \ not\ married(john, john).$
~~$\neg married(john, mary) \leftarrow \ not\ married(john, mary).$~~
$\neg married(john, ted) \leftarrow \ not\ married(john, ted).$

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow not\ married(john, john).$
$\neg married(john, mary) \leftarrow not\ married(john, mary).$
$\neg married(john, ted) \leftarrow not\ married(john, ted).$

Answer set(s): we must guess a possible answer set and use the reduct.
Lets try:
$S = \{married(john, mary), \neg married(mary, ted), \neg married(john, john),$
$\neg married(john, ted)\}$

Reduct: Step 2: Remove all other premises containing "*not*".
$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow$ ~~$not\ married(john, john)$~~.
~~$\neg married(john, mary) \leftarrow not\ married(john, mary).$~~
$\neg married(john, ted) \leftarrow$ ~~$not\ married(john, ted)$~~.

Examples

$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john) \leftarrow \ not\ married(john, john).$
$\neg married(john, mary) \leftarrow \ not\ married(john, mary).$
$\neg married(john, ted) \leftarrow \ not\ married(john, ted).$

Answer set(s): we must guess a possible answer set and use the reduct.
Lets try:
$S = \{married(john, mary), \neg married(mary, ted), \neg married(john, john),$
$\neg married(john, ted)\}$

Reduct:
$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john).$
$\neg married(john, ted).$ (This rule is redundant)

Examples

Answer set(s): we must guess a possible answer set and use the reduct.
Lets try:
$S = \{married(john, mary), \neg married(mary, ted), \neg married(john, john),$
$\neg married(john, ted)\}$

Reduct:
$married(john, mary).$
$\neg married(mary, ted).$
$\neg married(john, john).$
$\neg married(john, ted).$

Is S an answer set of the reduct? *Yes.*

Therefore, S *is an answer set of the original program*.

Computing Answer Sets

Answer sets can be computed by means of a solver such as CLINGO.

Download clingo 4.4.0 from

`http://sourceforge.net/projects/potassco/files/clingo/4.4.0/`

Compute one answer set with:

clingo *filename*

Compute *n* answer sets with (use 0 to compute all answer sets):

clingo *n filename*

Syntax changes:

- $\leftarrow$ is written `:-`
- $\neg$ is written `-`
- *or* is not supported (a different solver is needed)

Computing Answer Sets: Example

Given file "p1.asp":

$p(a) :- \mathit{not}\ q(a).$
$q(a) :- \ \ -r(c),\ \mathit{not}\ p(a).$
$p(b) :- p(b).$

Executing "CLINGO p1.asp" produces:

```
clingo version 4.4.0
Reading from p1.asp
p1.asp:2:9-13: warning: atom is undefined:
  r(c)

Solving...
Answer: 1
p(a)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.003s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Action Languages

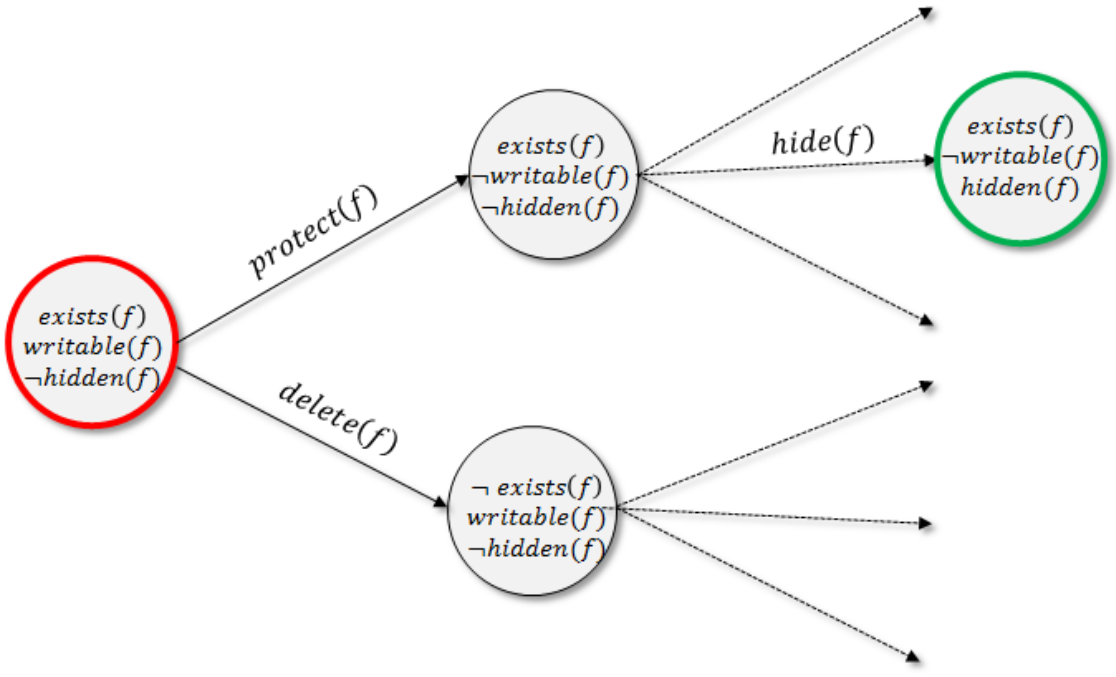
Required Reading

- Action Languages
  - http://ssdi.di.fct.unl.pt/rcr/geral/biblio/assets/gelfond98action.pdf

Types of Search: Forward Search

Given: (1) initial/starting state, (2) desired solutions (states), (3) operators.

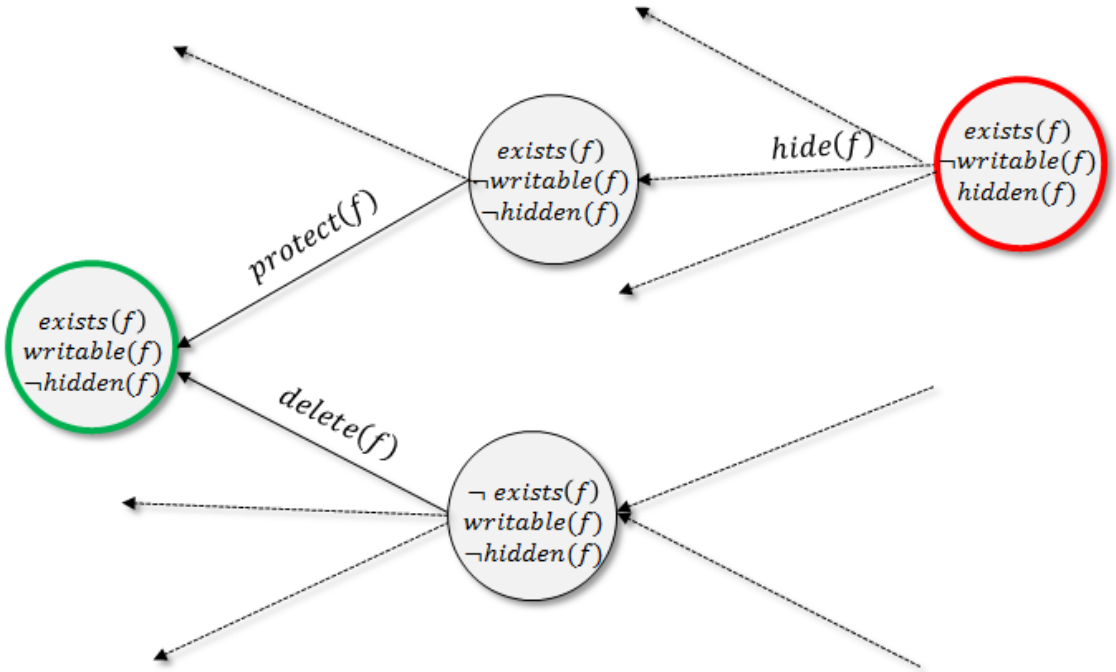
Forward search: start from the initial state and search the graph until a solution state has been found.



Types of Search: Backward Search

Given: (1) initial/starting state, (2) desired solutions (states), (3) operators.

Backward search: start from the solution states and search the graph until the initial state has been found.



Note: *the bottom node is never visited*.

What Makes a Search Algorithm Successful

- Finding every solution (soundness/completeness).
- Finding an answer in a reasonable amount of time and memory (performance).

Performance is often improved by *heuristics*:

- "Rules of thumb" that *usually* achieve their goal.

Soundness and Completeness

- Soundness: if the algorithm outputs X, then X is an answer to the problem.
- Completeness: if X is an answer to the problem, then the algorithm outputs X.

General Theory of Action and Change

Goal: decouple representation of a domain from reasoning about it.

Overview of the theory:

- World=a dynamic system (states + transitions)
- States are changed by actions

Action languages describe such systems.

Action languages enable:

- Concise and mathematically accurate descriptions of states and transitions
- Separation of concerns between system description and reasoning algorithm(s)

Some Terminology

Action signature Σ: $\langle \mathcal{F}, \mathcal{A} \rangle$

- $\mathcal{F}$: set of *fluents* (symbols)
- $\mathcal{A}$: set of *elementary actions* (symbols)
- $\mathcal{F}$, $\mathcal{A}$ are disjoint and non-empty

Fluent literal: a fluent $f \in \mathcal{F}$ or its negation $\neg f$

A set S of fluents is *complete* if for any fluent f either f or $\neg f$ is in S.

Set S of fluents is *consistent* if, for every fluent f, $\{f, \neg f\} \not\subseteq S$.

A set $\{\alpha_0, \ldots, \alpha_n\}$ of elementary actions is called a *(compound) action*. It is interpreted as a set of elementary actions performed *simultaneously*.

Markovian System Models

A dynamic system is modeled by the *transition diagram*.

Transition diagram: directed graph

- whose nodes correspond to physically possible states of the system
- whose arcs are labeled by (compound) actions and correspond to transitions.

A transition $\langle \sigma_0, a, \sigma_1 \rangle$ indicates that:

- action a is executable in state σ_0
- as a result of such execution, the system *may* move in state σ_1.

Note: in a transition diagram, the next state is determined by the previous state and by the actions executed.
Models with this property are called **Markovian**.
In other words: the effect of action depends only on the state in which it was executed. How this state was reached is irrelevant.

Problem: Representing Transition Diagrams

- Due to the size of the diagram, the problem of finding its concise and mathematically accurate description is not trivial and has been a subject of research for about 30 years
- Its solution requires good understanding of the nature of causal effects of actions in the presence of complex interrelations between fluents.
- Additional level of complexity is added by the need to specify what is not changed by actions. As noticed by McCarthy, the latter, known as the 'frame problem', can be reduced to representation of Inertia Axiom:

Things normally stay as they are.

Action Language *AL*

Action languages: formal models of parts of natural language that are used for describing the behavior of dynamic systems.

Action signature Σ: $\langle \mathcal{F}, \mathcal{A} \rangle$, defined as before

The statements of the language:

- *Dynamic causal law*: a **causes** l **if** p
- *State Constraint*: l **if** p
- *Executability condition*: a **impossible_if** p

where: l (*head* of the statement) is a fluent literal, a is an action, and p (*body* of the statement) is a set of fluent literals.
Dynamic causal laws and state constraints are *causal laws*.

System Descriptions

System (or action) description of AL: an action signature Σ and a collection, $\mathcal{D}$, of statements of AL formed from Σ.

Simplification:
Actions occurring in dynamic laws are elementary.

Action Language AL: Example

Action signature:

$$\Sigma = \langle \{f, g, h\}, \{a_1, a_2\} \rangle$$

Action description:

$$w: \ f \text{ if } g, \neg h$$

$$d: \ a_1 \text{ causes } g \text{ if } \neg h$$

$$i: \ a_1 \text{ impossible\_if } g, h$$

System States

A system description $\mathcal{D}$ serves as a specification of the transition diagram T defining all possible trajectories of the corresponding dynamic system.

A set S of fluent literals is *closed under a state constraint* w if $head(w)$ holds in S whenever $body(w)$ is satisfied by S.

A set S of fluent literals is *closed under a set*, Z, *of state constraints* if it is closed under each element of Z.

State σ of T: a complete and consistent set of fluent literals that is closed under the state constraints of T.

The System States: Examples

$\Sigma = \langle \{f, g, h\}, \{a_1, a_2\} \rangle$
w : f if $g, \neg h$
d : a_1 causes g if $\neg h$
i : a_1 impossible_if g, h

A state or not a state?

- $\{f, g\}$
- $\{f, g, \neg f\}$
- $\{f, g, h\}$
- $\{f, g, \neg h\}$

Recall:

- Consistent
- Complete
- Closed under the state constraints

Transition Relation – the Idea

To complete the definition of T we need to define the transition relation $\langle \sigma_0, a, \sigma_1 \rangle$.

For every state σ_0 and action a, we must specify the collection of their possible successor states.

Transition Relation

The set $Cn_Z(S)$ of *consequences of* S *under* Z is the smallest set of fluent literals that contains S and is closed under Z.

The set $E(a, \sigma)$ of *direct effects* of a in σ under AD is the set:

$E(a, \sigma) = \{head(d) \mid d \in AD \wedge d \text{ is dyn. law} \wedge body(d) \subseteq \sigma \wedge action(d) \subseteq a\}$

Action a is *non-executable* in σ if there exists an executability condition i such that:

- $body(i) \subseteq \sigma$
- $action(i) \subseteq a$

If that is not the case, a is *executable* in σ.

Transition Relation: Examples

The set $Cn_Z(S)$ of *consequences of S under Z* is the smallest set of fluent literals that contains S and is closed under Z.

$$Z_a = \left\{ \begin{array}{l} f \text{ if } g. \end{array} \right\} \quad S_1^a = \{f\} \quad Cn_{Z_a}(S_1^a) = \{f\}$$
$$Z_a = \left\{ \begin{array}{l} f \text{ if } g. \end{array} \right\} \quad S_2^a = \{\} \quad Cn_{Z_a}(S_2^a) = \{\}$$

$$Z_b = \left\{ \begin{array}{l} f \text{ if } g. \\ h \text{ if } f, g. \end{array} \right\} \quad S_1^b = \{g\} \quad Cn_{Z_b}(S_1^b) = \{f, g, h\}$$
$$Z_b = \left\{ \begin{array}{l} f \text{ if } g. \\ h \text{ if } f, g. \end{array} \right\} \quad S_2^b = \{f\} \quad Cn_{Z_b}(S_2^b) = \{f, g, h\}???$$

$$Z_c = \left\{ \begin{array}{l} f \text{ if } f. \end{array} \right\} \quad S_1^c = \{f\} \quad Cn_{Z_c}(S_1^c) = \{f\}$$
$$Z_c = \left\{ \begin{array}{l} f \text{ if } f. \end{array} \right\} \quad S_2^c = \{\} \quad Cn_{Z_c}(S_2^c) = \{f\}???$$

Transition Relation

$\langle \sigma_0, a, \sigma_1 \rangle$ is a transition in the transition diagram described by AD if:

- σ_0, σ_1 are states
- a is executable in σ_0
- $\sigma_1 = Cn_Z(E(a, \sigma_0) \cup (\sigma_0 \cap \sigma_1))$

Transition Relation: Example

$\Sigma = \langle \{f, g, h\}, \{a_1, a_2\} \rangle$
w : f if $g, \neg h$
d : a_1 causes g if $\neg h$
i : a_1 impossible_if g, h

A transition or not a transition?

- $\langle \{f, \neg g, h\}, \{a_1\}, \{f, g, h\} \rangle$
- $\langle \{f, \neg g, h\}, \{a_1\}, \{f, \neg g, h\} \rangle$
- $\langle \{f, \neg g, h\}, \{a_1, a_2\}, \{f, \neg g, h\} \rangle$
- $\langle \{f, \neg g, \neg h\}, \{a_1\}, \{f, g, \neg h\} \rangle$
- $\langle \{f, \neg g, \neg h\}, \{a_1\}, \{f, \neg g, \neg h\} \rangle$
- $\langle \{f, g, h\}, \{a_1\}, \{f, g, h\} \rangle$

Recall:

- σ_0, σ_1 are states
- a is executable in σ_0
- $\sigma_1 = Cn_Z(E(a, \sigma_0) \cup (\sigma_0 \cap \sigma_1))$

Yale Shooting Problem in AL

- Objects:
  - fred: a turkey
  - *g*: a gun
- Properties:
  - alive(T): T is alive
  - loaded(G): G is loaded
- Actions:
  - load(G): load G
  - shoot(G,T): shoot T using G
- Initial state: fred alive; gun unloaded
- Goal: fred not alive

In-Class Exercise

Try to come up on your own with an action description for YSP.

Yale Shooting Problem in AL

$$\Sigma = \langle \{alive(fred), loaded(g)\}, \{load(g), shoot(g,t)\} \rangle$$

// Shoot gun g at fred
// preconditions: g is loaded; fred is alive
// postconditions: g is not loaded; fred is not alive
$shoot(g, fred)$ causes $\neg loaded(g)$ if $loaded(g), alive(fred)$.
$shoot(g, fred)$ causes $\neg alive(fred)$ if $loaded(g), alive(fred)$.

Note the use of compound fluent literals such as $alive(fred)$.
Contrast with, e.g., $fred\_alive$.
Does the definition of action signature impose any requirements?

Finally: does the first dynamic law make sense???

Yale Shooting Problem in AL

$$\Sigma = \langle \{alive(fred), loaded(g)\}, \{load(g), shoot(g, t)\} \rangle$$

// Shoot gun g at fred
// preconditions: g is loaded; fred is alive
// postconditions: g is not loaded; fred is not alive
$shoot(g, fred)$ causes $\neg loaded(g)$ if $loaded(g)$.
$shoot(g, fred)$ causes $\neg alive(fred)$ if $loaded(g)$.

$shoot(g, fred)$ impossible_if $\neg loaded(g)$.

Observation: the relationship between $shoot(g, fred)$ and $\neg loaded(g)$ is better rendered as an executability condition.

Yale Shooting Problem in AL

$$\Sigma = \langle \{alive(fred), loaded(g)\}, \{load(g), shoot(g,t)\} \rangle$$

// Load gun g
// preconditions: g is not loaded
// postconditions: g is loaded
load(g) causes *loaded*(g) if $\{\}$.
load(g) impossible_if *loaded*(g).

Are there any other ways of formalizing action *load*(g)?

Yale Shooting Problem in AL

$$\Sigma = \langle \{alive(fred), loaded(g)\}, \{load(g), shoot(g,t)\} \rangle$$

// Load gun g
// preconditions: g is not loaded
// postconditions: g is loaded
$load(g)$ causes $loaded(g)$ if $\neg loaded(g)$.

Observation: it is sometimes difficult to decide between alternative formalizations when the problem is simple.

We did not specify initial state and goal state. Why?

Yale Shooting Problem in AL

Recall:

- σ_0, σ_1 are states (consistent, complete, closed under the state constraints)
- a is executable in σ_0
- $\sigma_1 = Cn_Z(E(a, \sigma_0) \cup (\sigma_0 \cap \sigma_1))$
- $\Sigma = \langle \{alive(fred), loaded(g)\}, \{load(g), shoot(g, t)\} \rangle$
  $shoot(g, fred)$ causes $\neg loaded(g)$ if $loaded(g)$.
  $shoot(g, fred)$ causes $\neg alive(fred)$ if $loaded(g)$.
  $shoot(g, fred)$ impossible_if $\neg loaded(g)$.
  $load(g)$ causes $loaded(g)$ if $\{\}$.
  $load(g)$ impossible_if $loaded(g)$.

Transitions??

$\langle \{alive(fred)\}, \{load(g)\}, \{alive(fred), loaded(g)\} \rangle$
$\langle \{\neg loaded(g), alive(fred)\}, \{load(g)\}, \{alive(fred), loaded(g)\} \rangle$
$\langle \{loaded(g), alive(fred)\}, \{load(g)\}, \{loaded(g), alive(fred)\} \rangle$
$\langle \{loaded(g), alive(fred)\}, \{shoot(g, fred)\}, \{\neg loaded(g), \neg alive(fred)\} \rangle$
$\langle \{loaded(g), \neg alive(fred)\}, \{shoot(g, fred)\}, \{\neg loaded(g), \neg alive(fred)\} \rangle$

Example: Causal Dependencies

Problem:
Consider a briefcase with two clasps. We have actions that unfasten the clasps, one at a time. If both clasps are unfastened the briefcase is open.
Create a (simple) model of this domain.
Solution:
We start with describing the action signature:

- Fluents: *open*, *fastened*(1), *fastened*(2)
- Actions: *unfasten*(1), *unfasten*(2)

In-Class Exercise

Try to come up on your own with an action description for this problem.

Example: Causal Dependencies

Action description

(We use variable C for clasps.)

Dynamic causal law:
$unfasten(C)$ causes $\neg fastened(C)$.

Shorthand for: $unfasten(1)$ causes $\neg fastened(1)$.
$unfasten(2)$ causes $\neg fastened(2)$.

Static causal law:
$open$ if $\neg fastened(1), \neg fastened(2)$.

Executability condition:
$unfasten(C)$ impossible_if $\neg fastened(C)$.

Example: Causal Dependencies

Recall:

- σ_0, σ_1 are states (consistent, complete, closed under the state constraints)
- a is executable in σ_0
- $\sigma_1 = Cn_Z(E(a, \sigma_0) \cup (\sigma_0 \cap \sigma_1))$
- $unfasten(C)$ causes $\neg fastened(C)$.
  $open$ if $\neg fastened(1), \neg fastened(2)$.
  $unfasten(C)$ impossible_if $\neg fastened(C)$.

Transitions?? ($fast'd$ is used as abbreviation of $fastened$)

$\langle \{fast'd(1), \neg fast'd(2)\}, \{unfasten(1)\}, \{fast'd(1), \neg fast'd(2)\} \rangle$
$\langle \{fast'd(1), \neg fast'd(2), \neg open\}, \{unfasten(1)\}, \{fast'd(1), \neg fast'd(2), \neg open\} \rangle$
$\langle \{fast'd(1), \neg fast'd(2), \neg open\}, \{unfasten(1)\}, \{\neg fast'd(1), \neg fast'd(2), \neg open\} \rangle$
$\langle \{fast'd(1), \neg fast'd(2), \neg open\}, \{unfasten(1)\}, \{\neg fast'd(1), \neg fast'd(2), open\} \rangle$
$\langle \{fast'd(1), \neg fast'd(2), \neg open\}, \{unfasten(2)\}, \{fast'd(1), \neg fast'd(2), \neg open\} \rangle$
$\langle \{\neg fast'd(1), \neg fast'd(2), \neg open\}, \{\}, \{\neg fast'd(1), \neg fast'd(2), open\} \rangle$

Advanced Privacy Examples and Time-based Reasoning

Advanced Privacy Challenges

Often, privacy issues that are considered are static in nature. However, even more issues can arise by reasoning about dynamic knowledge.

Time-Based Reasoning: Example

- Ashley Madison's account deletion service is exactly $19
- Even after account removal, user's transactions remain in the payment database
- *"They don't show up in the account list, but these were obviously former active users. Why else would they have spent $19 on Ashley Madison's website?"*

(http://www.wired.com/2015/08/ashley-madison-hack-exposes-wait-lousy-business/)

Time-Based Reasoning: Example

- "They don’t show up in the account list, but these were obviously former active users. Why else would they have spent $19 on Ashley Madison’s website?"
- How can we formalize this reasoning?

Problem Formalization

- Lists
  - List of user accounts
    Fluent `is_active(<user>)`
  - List of transactions, with username and amount
    Fluent `has_paid(<user>,<amount>)`

- Actions:
  - Users can:
    - Create a guest account
      `create_account(<user>)`
    - Buy credits in increments of $5
      `buy_credits(<user>,<amount in $5 increments>)`
    - Delete guest account at a cost of $19
      `delete_account(<user>)`

Action Description (Sketch)

- Causal laws describing the effects of the relevant actions
  - `create_account(u)`
    - causes the activation of the account, e.g.
      - `is_active(`*`u`*`)`
  - `delete_account(`*`u`*`)`
    - causes the removal of an account and a $19 transaction, e.g.
      - `has_paid(`*`u,19`*`)`
  - `buy_credits(u,`*`amt`*`)`
    - causes a transaction for *amt*, e.g.
      - `has_paid(`*`u,amt`*`)`

Formalization of Account Removal

Achieved by blocking the application of the inertia axioms:

```
% delete_accout(U) blocks inertia for is_active(U)
ab(is_active(U),T+1) :- occurs(delete_account(U),T).


% Modified inertia axiom (similar for negated fluents)
holds(F,T+1) :- holds(F,T), not ¬holds(F,T+1), not
                ab(U,T+1).
¬holds(F,T+1) :- ¬holds(F,T), not holds(F,T+1), not
                 ab(U,T+1).
```

Problem Formalization: Evidence

- Observations about fluents are encoded by statements:
  - `obs(<fluent>,{true|false},<step>)`
    - "The fluent is observed to be true (resp., false) at that step"
- Detecting unexpected evidence:
  - Reality Check Axioms

    ```
    :- not ¬holds(F,T), obs(F,false,T).
    :- not holds(F,T), obs(F,true,T).
    ```
  - The ASP program becomes inconsistent if unexpected observations are added

Unexpected Evidence: Example

- It is known that the account list was initially empty and there were no transactions.
- Now, a $19 transaction is found for user u (say, at time step 2)

```
% no holds(is_active(<u>),0) statements
% no holds(has_paid(<u>,<a>),0) statements

% observation
obs(has_paid(u,19),true,2).
```

- The observation triggers the Reality Check Axioms:
  - By inertia, no is_active(.) and has_paid(.) statements
  - The observation contradicts this expectation

Problem Formalization: Explaining Unexpected Observations

- Diagnostic reasoning
  - Any action may have occurred in the past

```
{ occurs(A,T) : action(A) } :- step(T).
```

  - Generates sequences of actions that:
    - Provide a cause for the observations
    - Eliminate the inconsistency triggered by the Reality Check Axioms

Explaining Evidence: Example

- [...] Now, a $19 transaction is found for user u (say, at time step 2)

```
obs(has_paid(u,19),true,2).

{ occurs(A,T) : action(A) } :- step(T).
```

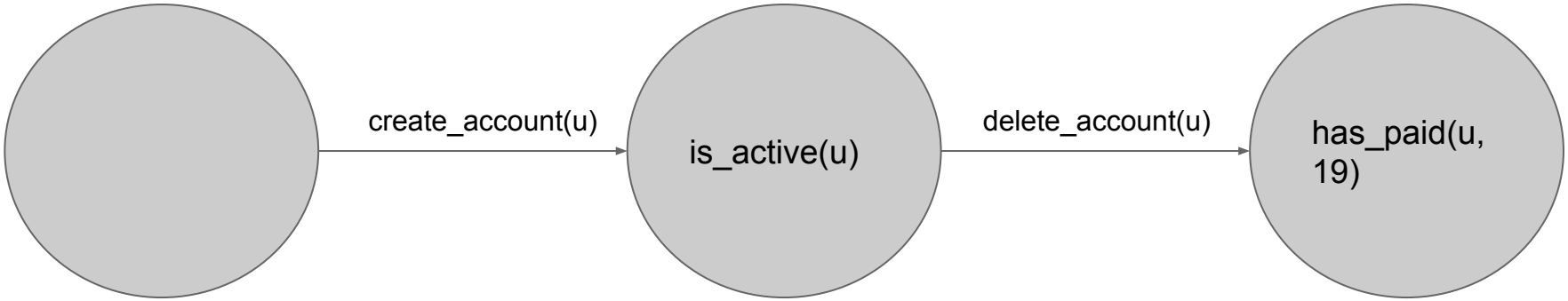
- An answer set exists that contains the sequence:
  `occurs(create_account(u),0), occurs(delete_account(u),1)`
- That is, the user must have created an account and then deleted it

  Observation explained and user caught!
- Given the above action description, it is not difficult to show that this is the only answer set

Sample Transition Diagram

```
obs(has_paid(u,19),true,2).

occurs(create_account(u),0).
occurs(delete_account(u),1).
```



Related Topic:
Law Informatics

Law Informatics

- Addresses a related set of problems
- Often tackled with logic-based approaches

Law Informatics

- LegalRuleML is a logic based language applied to the legal domain.
  - The paper talks about why, how, and when LegalRuleML is well-suited for modelling norms
- LegalRuleML is used to
  - Model legal rules, definitions, the effects of applying the rules
  - Identify exceptions to the rules, conflicts between rules, means to resolve them
  - Interpret ambiguous rules

https://www.researchgate.net/publication/277498922_LegalRuleML_Design_Principles_and_Foundations

Required Reading

- LegalRuleML: Design Principles and Foundations
  - https://www.researchgate.net/publication/277498922_LegalRuleML_Design_Principles_and_Foundations

Law Informatics

- LegalRuleML is also time-based because "legal texts are often amended as a society or judicial system evolves"
- Not only offers modelling of rules, but also reasoning with them "by filling requirements in the legal domain"