TDDC17

First-Order Logic
Nonmonotonic Reasoning
Answer Set Programming
Reasoning about Action and Change

Patrick Doherty
Dept of Computer and Information Science
Artificial Intelligence and Integrated Computer Systems Division

LINKÖPING UNIVERSITY

Limited Expressivity using Propositional Logic

Physics of the Wumpus World:
Modeling is difficult with Propositional Logic

Schemas:

$(B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y}))$ Def. of breeze in pos [x,y]

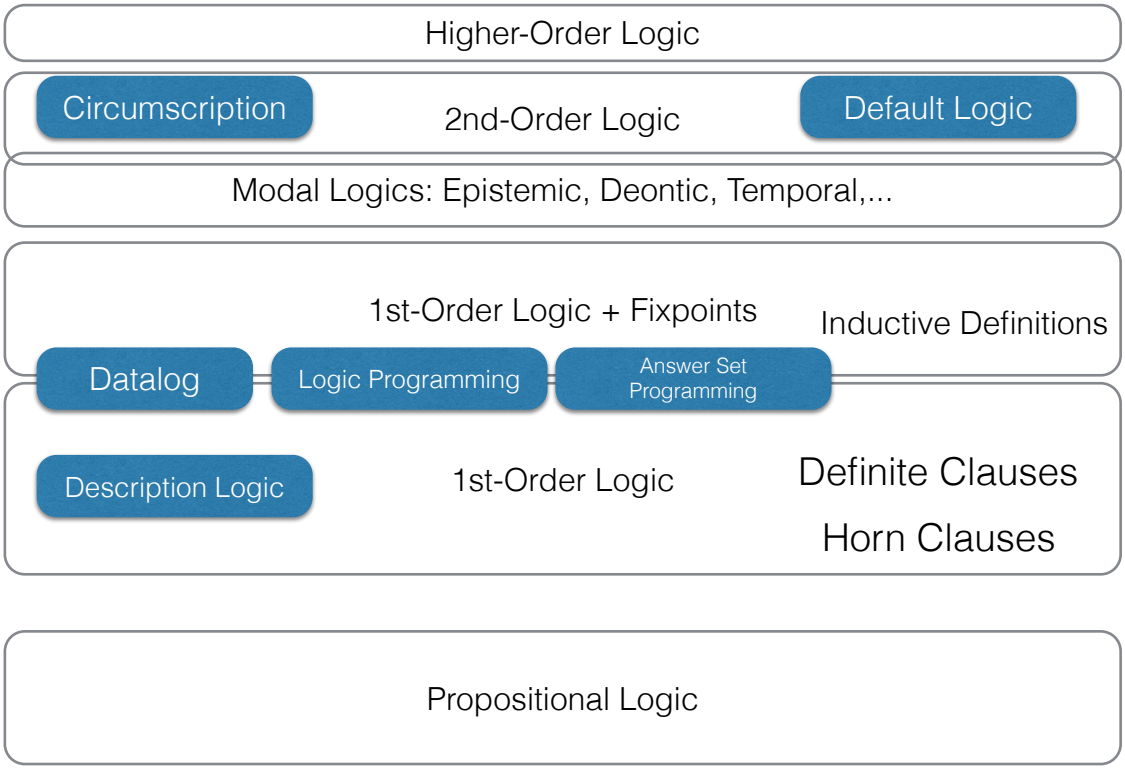
$(S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}))$ Def. of stench in pos [x,y]

$(W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}))$ There is at least one wumpus!

…, etc. There is only one wumpus!

LINKÖPING UNIVERSITY

Spectrum of Logics and Languages Studied and USED in KR



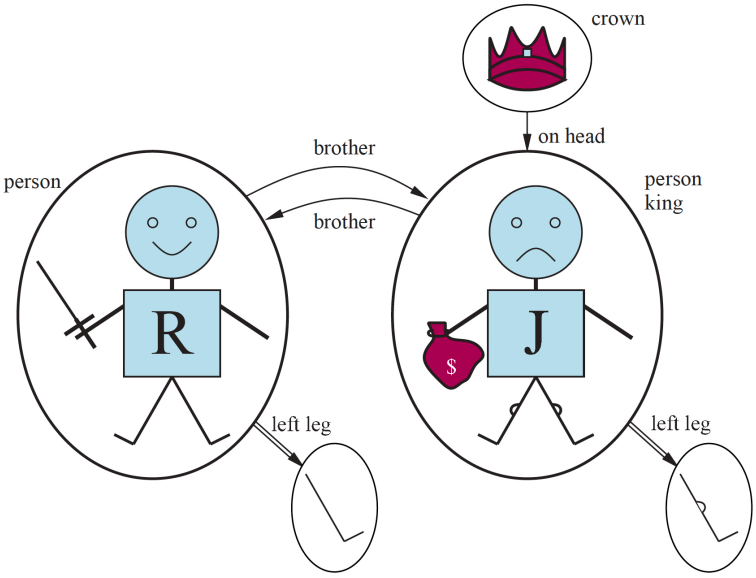
Ontological and Epistemological Commitments

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional Logic	facts	true/false/unknown
First-order Logic	facts, objects, relations	true/false/unknown
Temporal Logic	facts, objects, relations, times	true/false/unknown
Probability Theory	facts	degree of belief in [0,1]
Fuzzy Logic	facts with degree of truth in [0,1]	known interval value

Ontological Commitment - *what is assumed about the nature of reality*

Epistemological Commitment - *what is assumed about knowledge with respect to facts*

Ontological Commitments of First-order Logic



Facts
Objects
Relations

- Model with:

- 5 *objects*
- 2 *binary relations*
  - *brother*
  - *on-head*
- 3 unary relations
  - *person*
  - *crown*
  - *king*
- 1 *unary function*
  - *left-leg()*

The Syntax of First-Order Logic

Different applications have different first-order languages, but certain components are common to all languages

Propositional Connectives: $\wedge, \vee, \rightarrow, \neg, \leftrightarrow$

Propositional Constants: $\bot, \top$

Quantifiers:

$\forall$ (for all, the universal quantifier)

$\exists$ (there exists, the existential quantifier)

Punctuation: ")", "(", ",",...

Variables: $v_1, v_2, \ldots$ (which we write informally as $x, y, z, \ldots$)

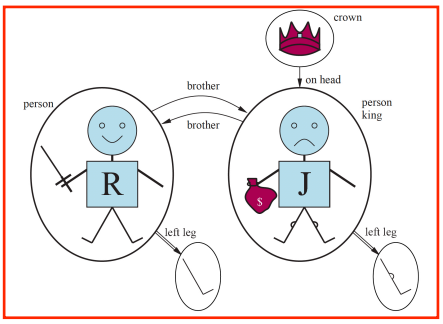
First-Order Language

Definition 1. A *first-order language*, $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$, is determined by specifying:

(1) A finite or countable set $\mathbf{R}$ of *relation symbols*, or *predicate symbols*, each of which has a positive integer associated with it denoting its arity.
(2) A finite or countable set $\mathbf{F}$ of *function symbols*, each of which has a positive integer associated with it denoting its arity.
(3) A finite or countable set $\mathbf{C}$ of *constant symbols*.

Example 1. Let $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ be the first-order language where,

(1) $\mathbf{R} = \{\langle Brother, 2\rangle, \langle OnHead, 2\rangle, \langle Person, 1\rangle, \langle Crown, 1\rangle, \langle King, 1\rangle\}$.
(2) $\mathbf{F} = \{\langle leftleg, 1\rangle\}$.
(3) $\mathbf{C} = \{John, Richard, Crown\}$.



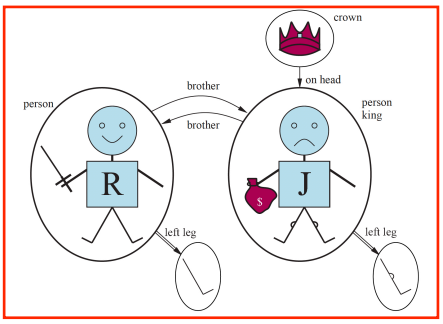
First-Order Language

Terms name individuals

Definition 2. The family of terms of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ is the smallest set meeting the conditions:

(1) Any variable is a term of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.
(2) Any constant symbol (member of $\mathbf{C}$) is a term of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.
(3) If f is an n-place function symbol (member of $\mathbf{F}$) and $t_1, \ldots, t_n$ are terms of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$, then $f(t_1, \ldots, t_n)$ is a term of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.

Examples: $Richard$, $John$, $leftleg(x)$, $leftleg(John)$, $Crown$



First-Order Language

Definition 3. An *atomic formula* of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ is any string of the form $R(t_1, \ldots, t_n)$ where $R \in \mathbf{R}$ is an n-place relation symbol and $t_1, \ldots, t_n$ are terms of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$; also $\bot$ and $\top$ are taken to be atomic formulas of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.

Examples:

$$Brother(Richard, John), King(John), OnHead(Crown, x)$$

Definition 4. The family of formulas of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ is the smallest set meeting the following conditions:

(1) Any atomic formula of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ is a formula of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.
(2) If A is a formula of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$, so is $\neg A$.
(3) For a binary connective $\circ$, if A and B are formulas of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$, so is $(A \circ B)$.
(4) If A is a formula of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$ and x is a variable, then $(\forall x)A$ and $(\exists x)A$ are formulas of $L(\mathbf{R}, \mathbf{F}, \mathbf{C})$.

Examples:

$$(\forall x)(\forall y)Brother(x, y) \rightarrow Brother(y, x)$$

$$Brother(Richard, John) \wedge King(John)$$

$$(\exists x)King(x) \wedge OnHead(Crown, x)$$

Restricted use of a First-Order Language

We will use a function free first-order language and assume a one-to-one mapping between constants and objects. Additionally, we will assume our object domain is finite.

Definition 1 A *signature* is a three tuple $\Sigma = \langle \mathcal{O}, \mathcal{P}, \mathcal{V} \rangle$ of (disjoint) sets, where:

- $\mathcal{O}$ is a set of object constants,
- $\mathcal{P}$ is a set of predicate constants, and
- $\mathcal{V}$ is a set of variables.

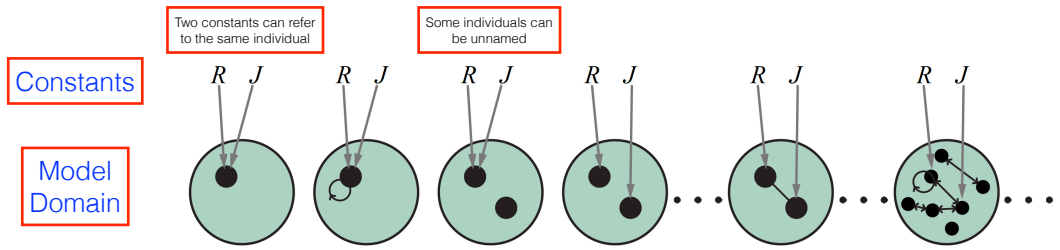
Example 1

- $\mathcal{O} = \{arad, sibiu, fagaras, bucharest\}$
- $\mathcal{P} = \{in, goto\}$
- $\mathcal{V} = \{X, Y\}$

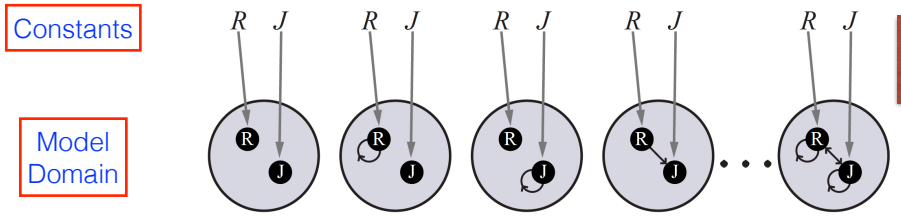
Logic Programming Convention:
- Predicates are lower case
- Variables are upper case
- Object constants are lower case

Database Semantics

Assume a language with 2 constant symbols, R and J
And one binary relation symbol



1st-order logic semantics: Some members of the set of all models



Database semantics: Some members of the set of all models

Database semantics is also used in logic programming

Restricted 1st-Order Language *cont*

Definition 2 *Terms* (over signature Σ) are defined as follows:

- Variables and object constants are terms.

Definition 3 An *atomic statement*, or simply an *atom*, is an expression of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms. Additionally, $\top$, $\bot$, representing $True$ and $False$, respectively, are also atomic statements.

Other statements in the language can be built from atomic statements using the logical connectives.

Definition 4 A *grounded atomic statement*, or simply a *grounded atom*, is an expression of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol of arity n, and $t_1, \ldots, t_n$ are terms that do not contain variables.

A grounded statement is built up from grounded atomic statements using the logical connectives.

Atomic Statements
$goto(X, Y)$
$goto(arad, Y)$
$in(fagaras)$

Grounded Atomic Statements
$goto(sibiu, fagaras)$
$in(fagaras)$

Statement: $goto(X, sibiu) \wedge goto(sibiu, Z) \rightarrow in(Z)$
Grounded Statement: $goto(arad, sibiu) \wedge goto(sibiu, fagaras) \rightarrow in(fagaras)$

Grounded Instantiations = Propositional Logic

Given a signature Σ_1, and set of formulas Δ, we view each formula as the set of its grounded instantiations.

Example 3
Given the following signature Σ_1:

- $\mathcal{O} = \{a, b\}$
- $\mathcal{P} = \{p, q\}$
- $\mathcal{V} = \{X\}$

and the the theory $\Delta = \{q(X) \rightarrow p(X)\}$, the grounding of Δ is:

$q(a) \rightarrow p(a)$

$q(b) \rightarrow p(b)$

Consequently, our theories are propositional where each grounded atomic statement can be viewed as a propositional variable, and each statement as a propositional formula, where the standard propositional semantics applies.

What about Quantifiers?

Example 4
Given the following signature Σ_1:

Existential formulas are disjunctions of grounded formulas

- $\mathcal{O} = \{a, b\}$
- $\mathcal{P} = \{p, q\}$
- $\mathcal{V} = \{X\}$

and the the theory $\Delta = \{\exists X(q(X) \rightarrow p(X))\}$, the grounding of Δ is:

$(q(a) \rightarrow p(a)) \vee (q(b) \rightarrow p(b))$

Example 5
Given the following signature Σ_1:

Universal formulas are conjunctions of grounded formulas

- $\mathcal{O} = \{a, b\}$
- $\mathcal{P} = \{p\}$
- $\mathcal{V} = \{X, Y\}$

and the the theory $\Delta = \{\forall X \forall Y\, p(X, Y)\}$, the grounding of Δ is:

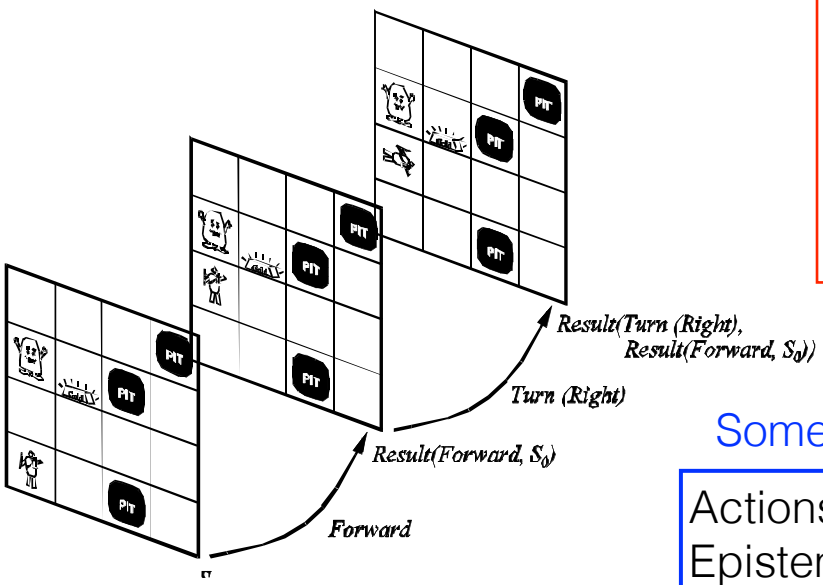
$p(a, b) \wedge p(b, a) \wedge p(a, a) \wedge p(b, b)$

Reasoning about Action and Change
Nonmonotonic Reasoning
Common Sense Reasoning

Action, Change and Common Sense

- One of the most important problems in KR is to be able to represent action, dynamics, time, space, belief, knowledge and causality.
  - *Intelligent Agents*: Observe, Plan and Act.
  - Agents with general intelligence operate in *Common Sense Informatic situations*
- Late 60's and 70's: difficulties in modeling and understanding
- Invention of *Nonmonotonic Logics* in early 70's
  - Used for modeling *normative* common sense behavior
  - Used for modeling inertial and other assumptions about dynamic environments
- Great progress in modeling in 80's and 90's
- Trend toward pragmatic, scalable reasoning tools after 2000
- Common sense reasoning and understanding of the world is still one of the "*Holy Grails*" of Artificial Intelligence.

Reasoning about Action and Change

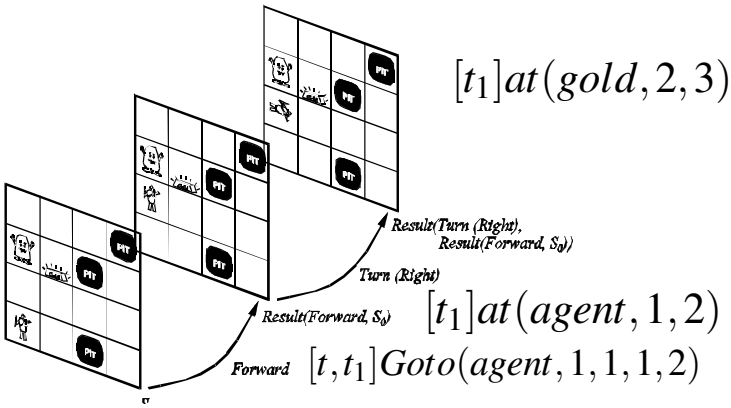


How do we represent and develop efficient inference mechanisms for dynamic behaviors of agents in incompletely specified environments?

Some modelling challenges

Actions & Effects
Epistemics & Causality
Temporal & Spatial Reasoning
Sensing & Observation
Planning & Plan Execution
Prediction & Explanation

Some Representation Problems



$[t_1]at(gold,2,3)$??

Is the gold still at 2,3 after the agent moves to 1,2 ?

Are the pits in the same place after the agent moves to 1,2 ?

$[t_1]at(agent,1,2)$

$[t,t_1]Goto(agent,1,1,1,2)$

The Frame Problem

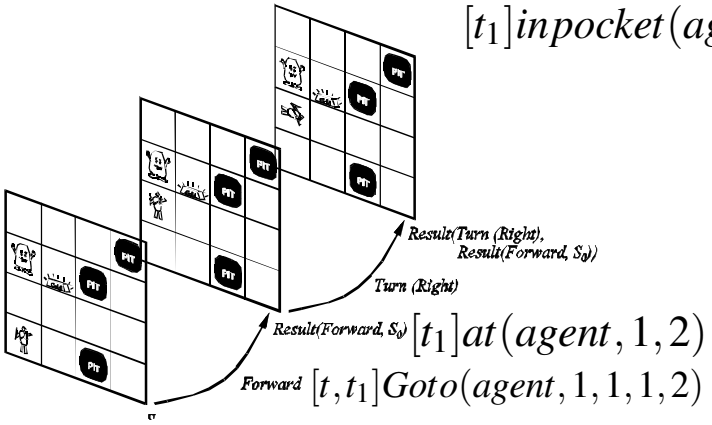
$[1]at(agent,1,1)$
$[1]at(gold,2,3)$

The world tends to remain inert. Most actions are local and do not disturb the larger frame. Most features in the world do not change.

How can this rule of thumb be represented succinctly in logic?

Some More Problems

$[t_1]inpocket(agent, bubblegum)$??



The Ramification Problem

There are many ramifications to an action, causal dependencies that become true when an action is executed

$[1]at(agent,1,1)$
$[1]at(gold,2,3)$
$[1]inpocket(agent, bubblegum)$

How can one succinctly represent these ramified effects without making action specifications overly detailed?

Some More Problems

Goto Action

$$[at(agent,x,y) \wedge adjacent(x,y,x_1,y_1) \wedge Safe(x_1,y_1)] \rightarrow [at(agent,x_1,y_1) \wedge \neg at(agent,x,y)]$$

But what if ….
The Wumpus breaks the rules and goes on a rampage?
The gold falls off the shelf and hits the agent in the head knocking it unconscious?
A chunk of ice falls from the sky blocking movement?

The Qualification Problem

All actions have exceptions (possible infinite). How can we succinctly represent that actions work most of the time but there are exceptions, and we can add them to the theory in a manner that does not force us to go into the action rules and change them all the time?

Application: Multi-Agent Iphone Game

Uses TAL & SAT solving algorithms

The basic problem

Universal Rules: $(\forall x) Tiger(x) \rightarrow Killer(x)$

Assume UNA, DC

$Tiger(Ted)$ $Killer(Ted)$

$Tiger(Tony)$

Always Exceptions to the case

$(\forall x) Tiger(x) \wedge x \neq Tony \rightarrow Killer(x)$

2021-08-22 $Tiger(Fred)$ $Killer(Fred)$

2021-08-24 $Tame(Fred)$

Retract based on new evidence

$(\forall x) Tiger(x) \wedge x \neq Tony \wedge x \neq Fred \rightarrow Killer(x)$

Do not want to continually update the rule!

A Solution?

Normative Rules: $(\forall x)Tiger(x) \wedge \neg Ab(x) \rightarrow Killer(x)$

Normally Tigers are Killers

But how do we axiomatize $Ab()$? Ab*normal*

$\neg Ab(Ted)$

:

:

:

$\neg Ab(Sally)$

Explicit Axiomatization of normal objects is not feasible

We would like to represent a rule of thumb:
Unless told otherwise, an object is normal

The only things abnormal are those explicitly stated to be abnormal in the theory.

Technically: Minimize the extension of $Ab()$
But let the logical formalism do this naturally

Monotonicity

Classical Logic:

IF $\Gamma \models \alpha$ THEN $\Gamma \cup \Delta \models \alpha$

Practical Reasoning is often not as conservative. We often "jump" to conclusions or assume something is true if there is no reason to believe otherwise. (*ceteris paribus* in law,… *normality*)

This is often part of what we call Commonsense reasoning

$\Gamma \models \alpha$ ✗✗ $\Gamma \cup \Delta \models \alpha$ (New Information)

Nonmonotonic Logics do not have the property of monotonicity in general.

Broad Set of Techniques In KR and CS based on this basic idea

Open & Closed World Assumptions
Predicate Completion
Negation as failure to prove
Circumscription
Default Logic
Answer Set Programming

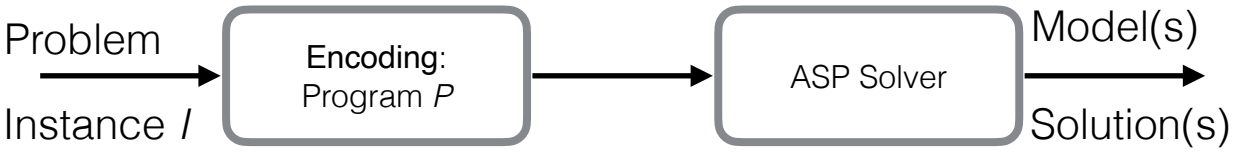
Common Thread:

- Make assumptions about incomplete information
- Do this by referring to meta-theoretic concepts
- The entailment relation generally becomes nonmonotonic

- If an atom is not in a database, assume it is false
- The objects or tuples that can be shown to satisfy a relation are the only ones that do
- If I can't prove a literal is true, assume it is false
- The sufficient conditions for a predicate in a theory are also the necessary conditions
- If a formula is consistent with a theory then assume it is true
- If there is absence of truth for an atom, assume it is false

Answer Set Programming

Answer Set Programming Paradigm



1. **Encode** *I* as a (nonmonotonic) logic program *P*, such that solutions for *I* are represented as models of *P*.
2. **Compute** some model *M* of *P*, using an ASP solver
3. **Extract** a solution for *I* from *M*.

ASP Solver: https://potassco.org/clingo/run/

Answer Set Program Syntax

We use use the restricted first-order language defined previously with the following additional definitions:

We will use the symbol "," instead of "∧", for the "and" connective.
We will use the symbol "*or*" instead of "∨", for the "or" connective.

Definition 5 A *literal* is an atomic formula p, or its negation $\neg p$. An *extended literal* is a literal or *not* l, where l is a literal.

Definition 6 A program Π consists of a signature Σ and a collection of rules of the form:

Postcondition/head — Precondition/body

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

where the l's are literals in Σ.

Two types of negation:
- Classical negation: ¬
- Default negation: *not*

Logic programming conventions
- A program is a set of rules
- Precondition/body: conjunction
- Postcondition/head: disjunction

Answer Set Program Syntax

The left-hand side of a rule is called its *head*.
The right-hand side of a rule is called its *body*.

$$\langle Head \rangle \leftarrow \langle Body \rangle$$

A rule with an *empty head* is called a *constraint*:

lhs is an empty disjunction = false

$$\leftarrow l_{i+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$$

A rule with an *empty body* is called a *fact*:

$$l_0\ or, \ldots, or\ l_i \leftarrow$$

rhs is an empty conjunction = true

Facts are also written as:

$$l_0\ or, \ldots, or\ l_i\ .$$

Some Examples

$killer(X) \leftarrow not\ ab(X), tiger(X)\ .$

Tigers are normally Killers

$tiger(ted)\ .$
$killer(ted) \leftarrow not\ ab(ted), tiger(ted)\ .$

Absence of truth for Ted being abnormal (default negation)

Since there is no evidence for ted being abnormal, the AS program infers: $killer(ted)$

$tiger(tony)\ .$
$ab(tony)\ .$
$killer(tony) \leftarrow not\ ab(tony), tiger(tony)\ .$

Tony is abnormal

Since there is evidence for Tony being abnormal, the AS program does not infer: $killer(tony)$.

$tiger(fred)\ .$
$tame(fred)\ .$
$ab(fred) \leftarrow tame(fred)\ .$
$killer(fred) \leftarrow not\ ab(fred), tiger(fred)\ .$

Tame tigers are abnormal

Since there is evidence for fred being abnormal, the AS program does not infer: $killer(fred)$.

Some more examples

Local Closed World Assumption (Predicate Completion of $ab()$):

A general rule of thumb!

$$\neg ab(X) \leftarrow not\ ab(X)$$

Normally,individuals are not abnormal

$$\neg ab(X) \leftarrow not\ ab(X)\,.$$
$$killer(X) \leftarrow \neg ab(X), tiger(X)\,.$$

Program P

Suppose (add to P): $tiger(ted)$ We can infer: $killer(ted)$

Suppose (add to P): $ab(X) \leftarrow tame(X)$

Suppose (Add to P): $tame(ted)$

We can no longer infer: $killer(ted)$
(Nor can we infer: $\neg killer(ted)$)

Nonmonotonic behaviour

Answer Set Programs: Informal Semantics

View a program Π as a specification for answer sets (models) of the program. The following principles apply when forming such sets:

(1) Satisfy the rules of Π. In other words, believe in the head of a rule if you believe in its body. (Supportedness)

(2) Do not believe in contradictions. (The set must be consistent).

(3) Adhere to the general principle:
"Believe nothing you are not forced to believe". (Minimization)

$$tiger(ted)\,.$$
$$killer(ted) \leftarrow not\ ab(ted), tiger(ted)\,.$$

Answer Sets: Informal Example

$$tiger(ted)\,.$$
$$killer(ted) \leftarrow not\ ab(ted), tiger(ted)\,.$$

Program P

Believe in *tiger(ted)* because it is a fact.

There is no reason to believe in $ab(ted)$ nor $\neg ab(ted)$

Since there is absence of truth for $ab(ted)$, $not\ ab(ted)$ is true.

Believe in killer(ted) because it is supported by the rule.

Do not believe anything else.

There is one answer set for this program:

$$\{tiger(ted), killer(ted)\}$$

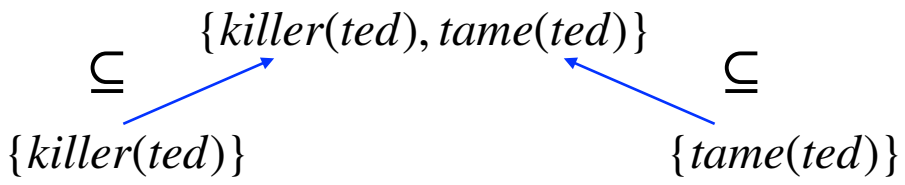
Epistemic Disjunction: Informal Example

$$tiger(ted)\,.$$
$$killer(ted)\ or\ tame(ted) \leftarrow tiger(ted)\,.$$

Program P

Believe in *tiger(ted)* because it is a fact.

There are three possible models for $killer(ted)\ or\ tame(ted)$



Believe nothing you are not forced to believe.

There are two minimal answer sets for this program:

$$\{tiger(ted), killer(ted)\} \qquad \{tiger(ted), tame(ted)\}$$

Answer Sets: Semantics

Stopped here!

Definition 7 [Satisfiability]
A set of (ground) literals S satisfies:

1. l **if** $l \in S$;
2. *not* l **if** $l \notin S$;
3. l_1 *or* $\ldots$ *or* l_n **if** *for some* $1 \leq i \leq n, l_i \in S$;
4. a set of (ground) extended literals **if** S satisfies every element of this set;
5. rule r **if** , whenever S satisfies r's body, it satisfies r's head.

Definition 8 [ASP Entailment]
A program Π *entails* a literal l ($\Pi \models l$) if l belongs to all answer sets of Π.

Answer Sets: Semantics

The first part of the definition is for programs without default negation (*not*)

Definition 9 [Answer Sets, Part I]
Let Π be a program not containing default negation (i.e., consisting of rules of the form):

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

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

1. S satisfies the rules of Π and
2. S is minimal (i.e., there is no proper subset of S that satisfies the rules of Π.

They can include extended literals with ¬

Examples

1. $r(b)$.
2. $q(b) \leftarrow p(a)$.
3. $p(a) \leftarrow r(b)$.
4. $q(a) \leftarrow s(a)$.

Unique answer set:

$$\{r(b), p(a), q(b)\}$$

1. $\neg s(b)$.
2. $q(b) \leftarrow q(a)$.
3. $p(a) \leftarrow r(b)$.
4. $q(a) \leftarrow \neg s(b)$.

Unique answer set:

$$\{\neg s(b), q(a), q(b)\}$$

Note that there are many other logical models for these programs using classical semantics!

Answer Sets: Semantics

The second part of the definition explains how to remove default negation so the first part can be applied

Definition 10 [Answer Sets, Part II]
Let Π be an arbitrary program and S be a set of ground literals. By Π^S we denote the program obtained from Π by

1. removing all rules containing *not* l such that $l \in S$; — *not* l is false
2. removing all other premises of the remaining rules containing *not*. — *not* l is true

S is an answer set of Π if S is an answer set of Π^S.
We refer to Π^S as the *reduct* of Π with respect to S.

A Reduct Example

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

rule	Π	Π^S
r_1	$p(a) \leftarrow not\ q(a).$	(deleted)
r_2	$p(b) \leftarrow not\ q(b).$	$p(b).$
r_3	$q(a).$	$q(a).$

(1) Remove r_1 from Π because it has $not\ q(a)$ in its premise, whereas $q(a) \in S$.

(2) Remove the premise $not\, q(b)$ of r_2.

S is an answer set of Π^S and, hence of Π.

A Reduct Example

Possible ASETs: Powerset of literals in rule heads:

$\{\}$ $\{tiger(ted)\}$ $\{killer(ted)\}$ $\{tiger(ted), killer(ted)\}$

$S = \{tiger(ted), killer(ted)\}$

rule	Π	Π^S
r_1	tiger(ted).	tiger(ted).
r_2	$killer(ted) \leftarrow tiger(ted), not\ ab(ted).$	$killer(ted) \leftarrow tiger(ted).$

S is an answer set of Π^S and, hence of Π.

$S = \{tiger(ted)\}$ is not an answer set of Π^S and, hence not of Π.

$S = \{killer(ted)\}$ is not an answer set of Π^S and, hence not of Π.

$S = \{\}$ is not an answer set of Π^S and, hence not of Π.

Reduct Example

$S = \{tiger(ted), tame(ted), ab(ted)\}$

rule	Π_1	Π_1^S
r_1	tiger(ted).	tiger(ted).
r_2	$killer(ted) \leftarrow tiger(ted), not\ ab(ted).$	(deleted).
r_3	tame(ted).	tame(ted).
r_4	$ab(ted) \leftarrow tame(ted).$	$ab(ted) \leftarrow tame(ted).$

S is an answer set of Π^S and, hence of Π.

Nonmontonic Entailment relation

Note that $\Pi \subset \Pi_1$ and $\Pi \models killer(ted)$ but $\Pi_1 \not\models killer(ted)$

Extending a theory may lead to retraction of beliefs!!

Programs with several Answer Sets

$$paint(red) \leftarrow not\ paint(blue)\,.$$
$$paint(blue) \leftarrow not\ paint(red)\,.$$

Candidates: $\{\}, \{paint(red)\}, \{paint(blue)\}, \{paint(red), paint(blue)\}$

Two Answer Sets

$S = \{paint(red)\}$

rule	Π	Π^S
r_1	$paint(red) \leftarrow not\ paint(blue).$	$paint(red).$
r_2	$paint(blue) \leftarrow not\ paint(red).$	delete

$S = \{paint(blue)\}$

rule	Π	Π^S
r_1	$paint(red) \leftarrow not\ paint(blue).$	delete
r_2	$paint(blue) \leftarrow not\ paint(red).$	$paint(blue).$

Programs with no Answer Sets

$$paint(red) \leftarrow not\ paint(red)\,.$$

Candidates: $\{\}, \{paint(red)\}$

$\{\}$ does not satisfy the the program's rule

$\{paint(red)\}$ $paint(red)$ is not supported by any rule in the program

The program has no answer sets!

Please note: No answer sets is not the same as the empty answer set: $\{\}$

Programs with empty Answer Sets

$$p(b) \leftarrow \neg p(a)\,.$$

Candidates: $\{\}, \{p(b)\}$

$\{p(b)\}$ Not an answer set because there are no facts supporting the rule so nothing needs to be in the answer set

$\{\}$ Satisfies the rule because it does not satisfy its body
It is an answer set.

Note: A query about $p(b)$ would return *unknown*

Querying Programs

Definition 11
The answer set to a ground conjunctive query, $l_1 \wedge \ldots \wedge l_n$, where $n \geq 1$, is

- yes, *if* $\Pi \models \{l_1, \ldots, l_n\}$,
- no, *if there is* i *such that* $\Pi \models \overline{l_i}$,
- unknown *otherwise.*

The answer set to a ground disjunctive query, $l_1 \vee \ldots \vee l_n$, where $n \geq 1$, is

- yes, *if there is* i *such that* $\Pi \models l_i$,
- no, *if* $\Pi \models \{\overline{l_1}, \ldots, \overline{l_n}\}$,
- unknown *otherwise.*

An answer to a query $q(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ is the list of variables occuring in q, is a sequence of ground terms $t_1, \ldots, t_n$ such that $\Pi \models q(t_1, \ldots, t_n)$.

A Useful Proposition

Proposition 1 Let S be an answer set of a (ground) ASP program Π.

1. S satisfies every rule $r \in \Pi$.
2. If literal $l \in S$ then there is a rule r from Π such that the body of r is satisfied by S and l is the only literal in the head of r satisfied by S. In other words, the rule r supports S.

This implies that only minimal, consistent subsets of literals in heads of rules in a program can be potential answer sets for that program!

Note: For positive programs with no default/classical negation, there is always a unique answer set if the program is definite (one literal in the head).

Note: For programs with default negation (not), there may be 0, 1 or more answer sets (including an empty answer set) if the program is consistent.

Computing Minimal Answer Sets

In the following, assume that rules do not contain classical negation and have a single positive literal in the head. The body of a rule can contain default negated positive literals. Let's call programs with such rules, normal programs.

A reduct Φ^S of a normal program Π and a set of ground literals S is a definite or positive program.

Definition 24. [Consequence operator T_Π]
The smallest model, $Cn(\Pi)$, of a positive program Π can be computed via its associated *consequence operator* T_Π. For a set of atoms X we define,

$$T_\Pi X = \{head(r) \mid r \in \Pi \text{ and } body(r) \subseteq X\}.$$

Iterated applications of T_Π are written as T_Π^j for $j \geq 0$, where

$$T_\Pi^0 X = X$$
$$T_\Pi^i X = T_\Pi T_\Pi^{i-1} X \text{ for } i \geq 1.$$

For any positive program Π, we have $Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i \emptyset$.
Since T_Π is monotonic, $Cn(\Pi)$ is the smallest fixpoint of T_Π.

Each iteration fires rules whose body is satisfied

Example

$$\Pi = \{\underline{a \leftarrow}, \underline{b \leftarrow a}, \underline{c \leftarrow a, b}, \underline{e \leftarrow f}\}$$

$$T_\Pi^0 \emptyset = \emptyset$$
$$T_\Pi^1 \emptyset = \{a\} \quad = T_\Pi T_\Pi^0 \emptyset = T_\Pi \emptyset$$
$$T_\Pi^2 \emptyset = \{a, b\} \quad = T_\Pi T_\Pi^1 \emptyset = T_\Pi \{a\}$$
$$T_\Pi^3 \emptyset = \{a, b, c\} = T_\Pi T_\Pi^2 \emptyset = T_\Pi \{a, b\}$$
$$T_\Pi^4 \emptyset = \{a, b, c\} = T_\Pi T_\Pi^3 \emptyset = T_\Pi \{a, b, c\}$$

Fixpoint: $T_\Pi(X) = X$

This is simply forward chaining of rules described differently in terms of Fixpoints

Inference with Forward Chaining using definite clauses

function PL-FC-ENTAILS?(KB, q) **returns** *true* or *false*
  inputs: KB, the knowledge base, a set of propositional definite clauses
    q, the query, a proposition symbol
  count ← a table, where *count*[c] is the number of symbols in c's premise
  inferred ← a table, where *inferred*[s] is initially *false* for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while *agenda* is not empty **do**
    p ← POP(*agenda*)
    if $p = q$ **then return** *true*
    if *inferred*[p] = *false* **then**
      inferred[p] ← *true*
      for each clause c in KB where p is in c.PREMISE **do**
        decrement *count*[c]
        if *count*[c] = 0 **then** add c.CONCLUSION to *agenda*
  return *false*

Figure 7.12 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

R&N, 4th edition: p.231

R&N, 4th global edition: p.249

Computing Answer Sets for Normal Programs

Given a program Π:

(1) Compute the possible answer sets for Π:

  (a) Powerset 2^{Π} of all atoms in the heads of rules in Π.

(2) For each $S \in 2^{\Pi}$:

  (a) Compute the reduct Π^S of Π.

  (b) If $Cn(\Pi^S) = S$ then S is an answer set for Π.

  (c) If $Cn(\Pi^S) \neq S$ then S is not an answer set for Π.

Note: There are more efficient ways to do this!

Action and Change: Frame Problem

Frame Problem:
Features in the world tend to stay the same unless effected by some actions

occlude : *allow to change*
$occlude(Time, Object, Feature)$
$holds(Time, Object, Feature)$
$occurs(Time, Agent, Action)$

Wumpus World: Move from cell [1,1] to [1,2] from time 1 to time 2:

$$holds(2,agent,at(1,2)) \leftarrow holds(1,agent,at(1,1)), occurs(1,agent,goto(1,1,1,2)).$$
$$\neg holds(2,agent,at(1,1)) \leftarrow holds(1,agent,at(1,1)), occurs(1,agent,goto(1,1,1,2)).$$

$$occlude(2,agent,at(1,2)) \leftarrow holds(1,agent,at(1,1)), occurs(1,agent,goto(1,1,1,2)).$$
$$occlude(2,agent,at(1,1)) \leftarrow holds(1,agent,at(1,1)), occurs(1,agent,goto(1,1,1,2)).$$

$$holds(1,wumpus,at(1,3)).$$
$$holds(1,gold,at(2,3)).$$

Using ← for :-

Inertia Assumption: Frame Axioms

$$holds(T+1,Object,Feature) \leftarrow not\ occlude(T+1,Object,Feature), holds(T,Object,Feature)$$
$$\neg holds(T+1,Object,Feature) \leftarrow not\ occlude(T+1,Object,Feature), \neg holds(T,Object,Feature)$$

Commonsense rule of thumb:
Things tend to stay the same

Encoding a planning problem using Answer sets

```
#include <incmode>.

#program base.
% Define
location(table).
location(X) :- block(X).
holds(F,0) :- init(F).

#program step(t).
% Generate
{ move(X,Y,t) : block(X), location(Y), X != Y } = 1.
% Test
:- move(X,Y,t), holds(on(A,X),t-1).
:- move(X,Y,t), holds(on(B,Y),t-1), B != X, Y != table.
% Define
moved(X,t) :- move(X,Y,t).
holds(on(X,Y),t) :- move(X,Y,t).    Action law
holds(on(X,Z),t) :- holds(on(X,Z),t-1), not moved(X,t).

% continued ..
```

Generate moves, allow 1
At each timepoints

Can't move a block
that is under a block

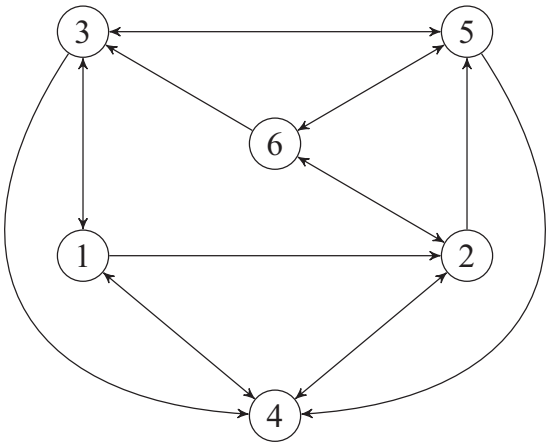
Only 1 block on top
Of another at a time

```
#program check(t).
% Test
:- query(t), goal(F), not holds(F,t).

% Display
#show move/3.

#program base.
% Sussman Anomaly
%
block(b0).
block(b1).
block(b2).
%
% initial state:
%
%  2
%  0 1
% -------
%
init(on(b1,table)).
init(on(b2,b0)).
init(on(b0,table)).
%
% goal state:
%
%  2
%  1
%  0
% -------
%
goal(on(b1,b0)).
goal(on(b2,b1)).
goal(on(b0,table)).
```

Frame problem: inertia assumption

```
Answer: 1
move(b2,table,1) move(b1,b0,2) move(b2,b1,3)
SATISFIABLE
```

Graph-Coloring



Directed graph with 6 nodes
And 17 edges

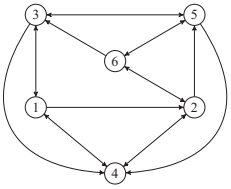
Two connected nodes
can not have the same color

Assume 3 colors

LINKÖPING UNIVERSITY

Encoding the N-Color Problem using Answer sets

```
% Default
#const n = 3.

% Generate
{ color(X,1..n) } = 1 :- node(X).
% Test
:- edge(X,Y), color(X,C), color(Y,C).

% Nodes

node(1..6).

% (Directed) Edges

edge(1,(2;3;4)).  edge(2,(4;5;6)).  edge(3,(1;4;5)).
edge(4,(1;2)).    edge(5,(3;4;6)).  edge(6,(2;3;5)).

% Display
#show color/2.
```



There are 3 colors

If X is a node, it has a unique color

Constraint: 2 connected nodes can not have the same color

There are 6 nodes named 1 to 6

edge(1, (2;3;4)) is an abbreviation for:
edge(1,2), edge(1,3),edge(1,4)

Just display the *color* relation in the output

One answer set (there are more):

```
Answer: 1
color(2,2) color(1,3) color(3,2) color(4,1) color(5,3) color(6,1)
```

LINKÖPING UNIVERSITY

Book

Answer Set Solving in Practice,
M. Gebser, R Kaminski, B. Kaufmann, T. Schaub
Morgan & Claypool Publishers, 2013

Downloadable Answer Set Solver System:

http://potassco.sourceforge.net/

On-Line ASP Solver: https://potassco.org/clingo/run/

Sample questions and answers
(LISAM- course documents): answer_set_problems.pdf