TDDC17

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

Limited Expressivity using Propositional Logic

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

Schemas:

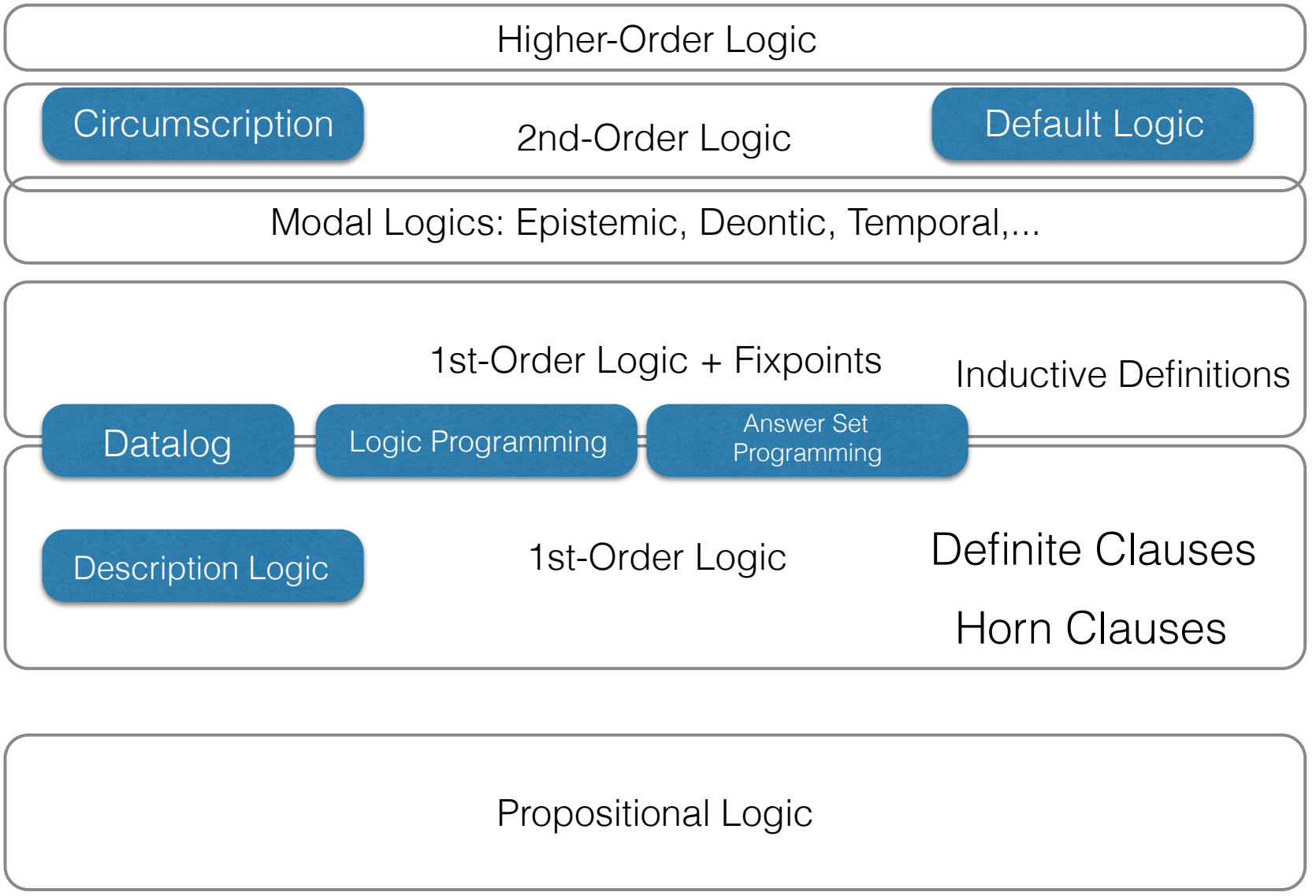
$(B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y}))$ Def. of breeze in pos [x,y]

$(S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y}))$ 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!

Spectrum of Logics and Languages



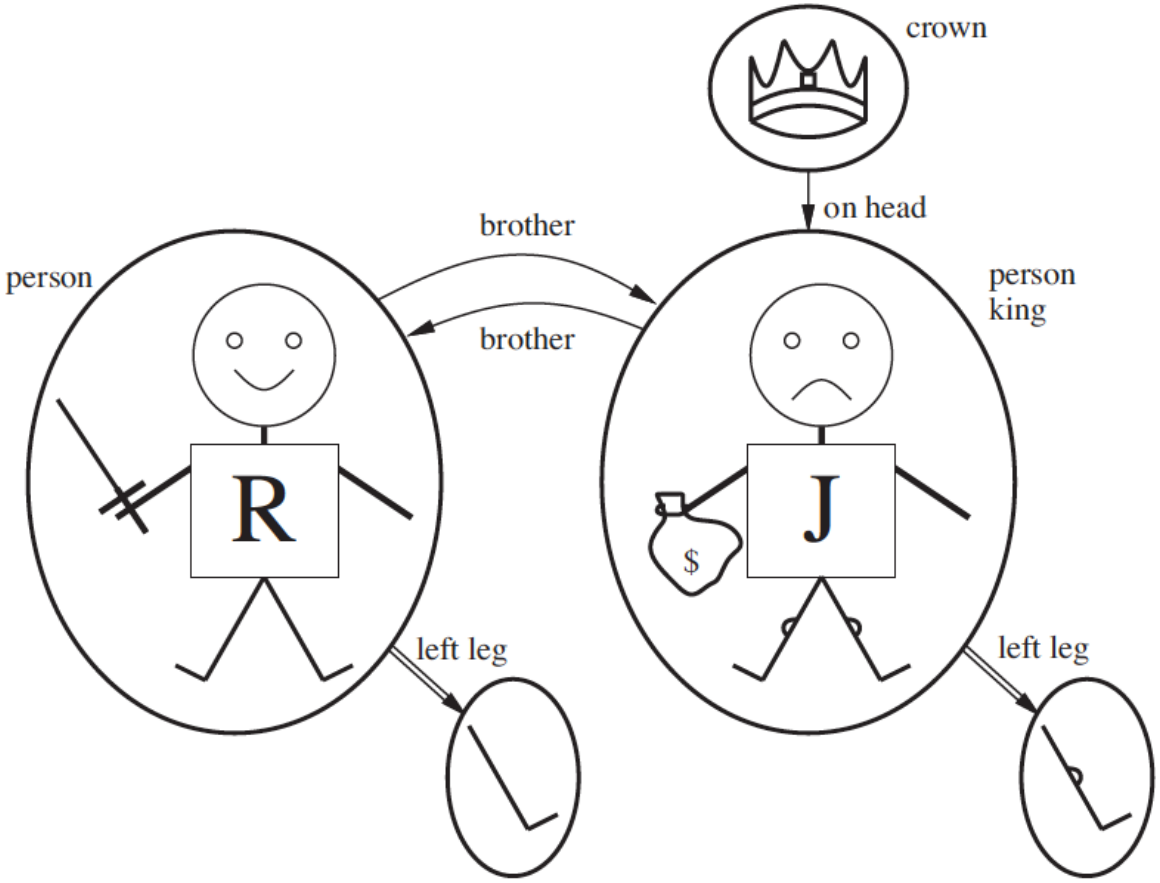
Ontological and Epistemological Commitments

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

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

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.

Example 2

$goto(X, Y) \wedge goto(Y, Z) \rightarrow in(Z)$

$goto(arad, sibiu) \wedge goto(sibiu, fagaras) \rightarrow in(fagaras)$

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.

Quantifiers

Example 4
Given the following signature Σ_1:

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

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

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

Existential formulas are disjunctions of grounded formulas

Example 5
Given the following signature Σ_1:

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

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

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

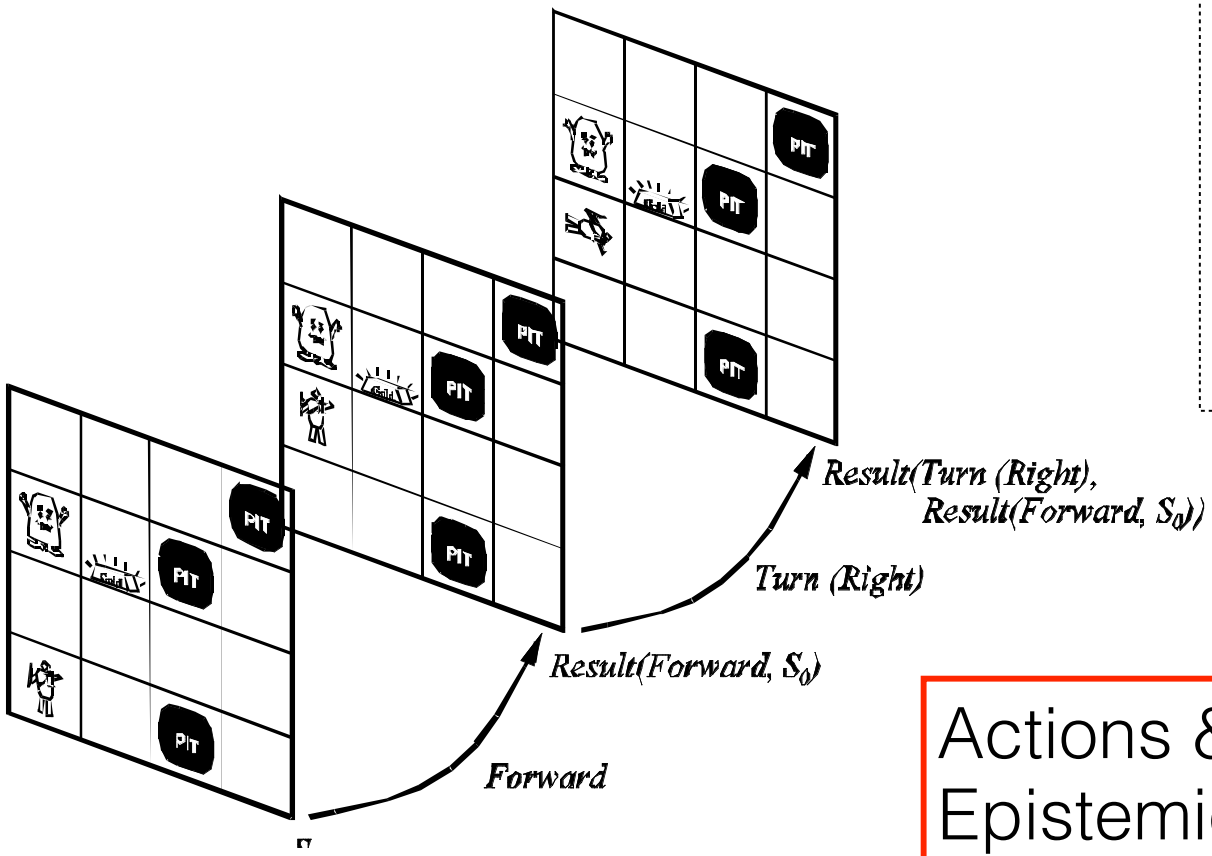
Universal formulas are conjunctions of grounded formulas

Reasoning about Action and Change
Nonmonotonic Reasoning

Action and Change

- One of the most important problems in KR is to be able to represent Action, dynamics and casuality.
  - Intelligent Agents: Observe, Plan and Act.
- Late 60's and 70's: difficulties in modeling
- Invention of Nonmonotonic Logics in early 70's
  - Modeling normative behavior or inertial assumptions about dynamic environments
- Great progress in modeling in 80's and 90's
- Trend toward pragmatic, scalable reasoning tools in 00's and 10's

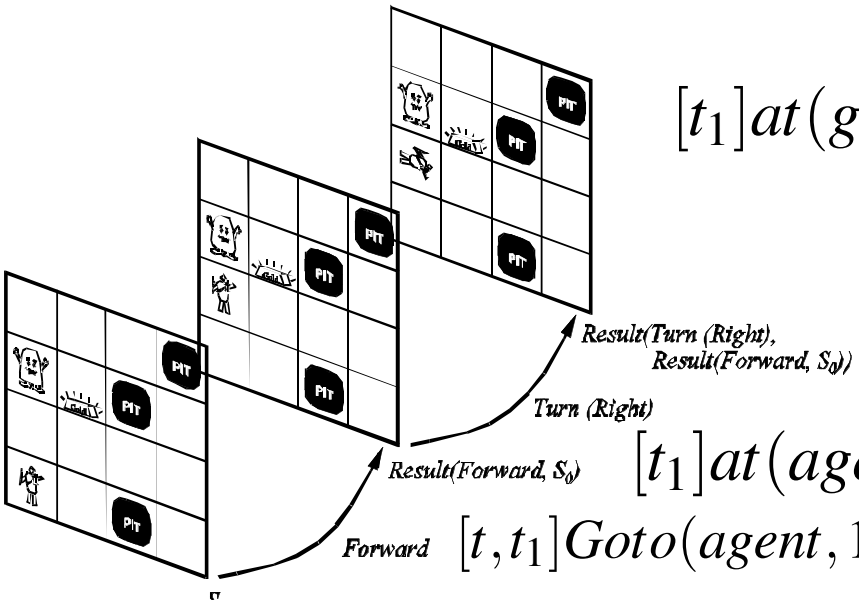
Reasoning about Action and Change



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

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

$[t]at(agent,1,1)$

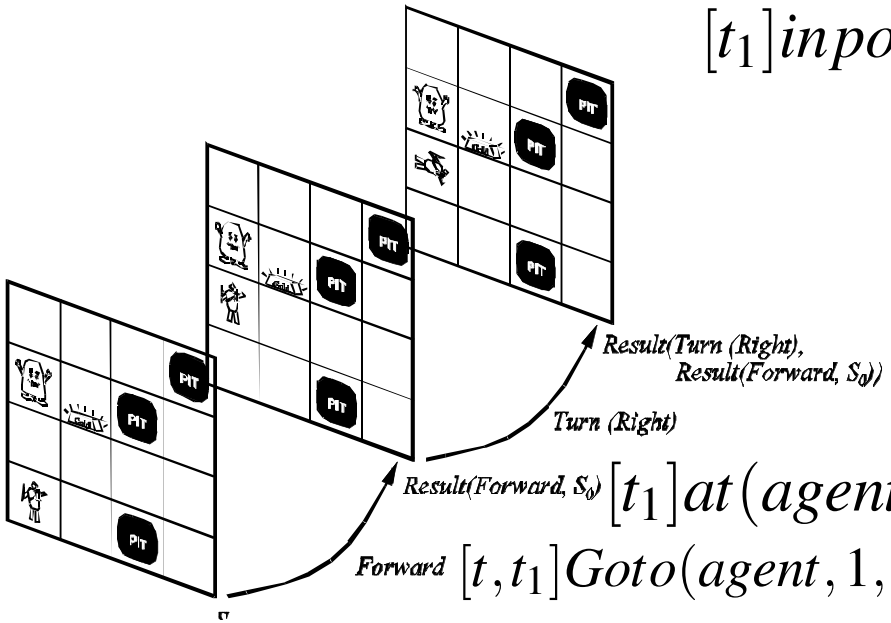
$[t]at(gold,2,3)$

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

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



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

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

The Ramification Problem

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

$[t]at(agent, 1, 1)$

$[t]at(gold, 2, 3)$

$[t]inpocket(agent, bubblegum)$

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?

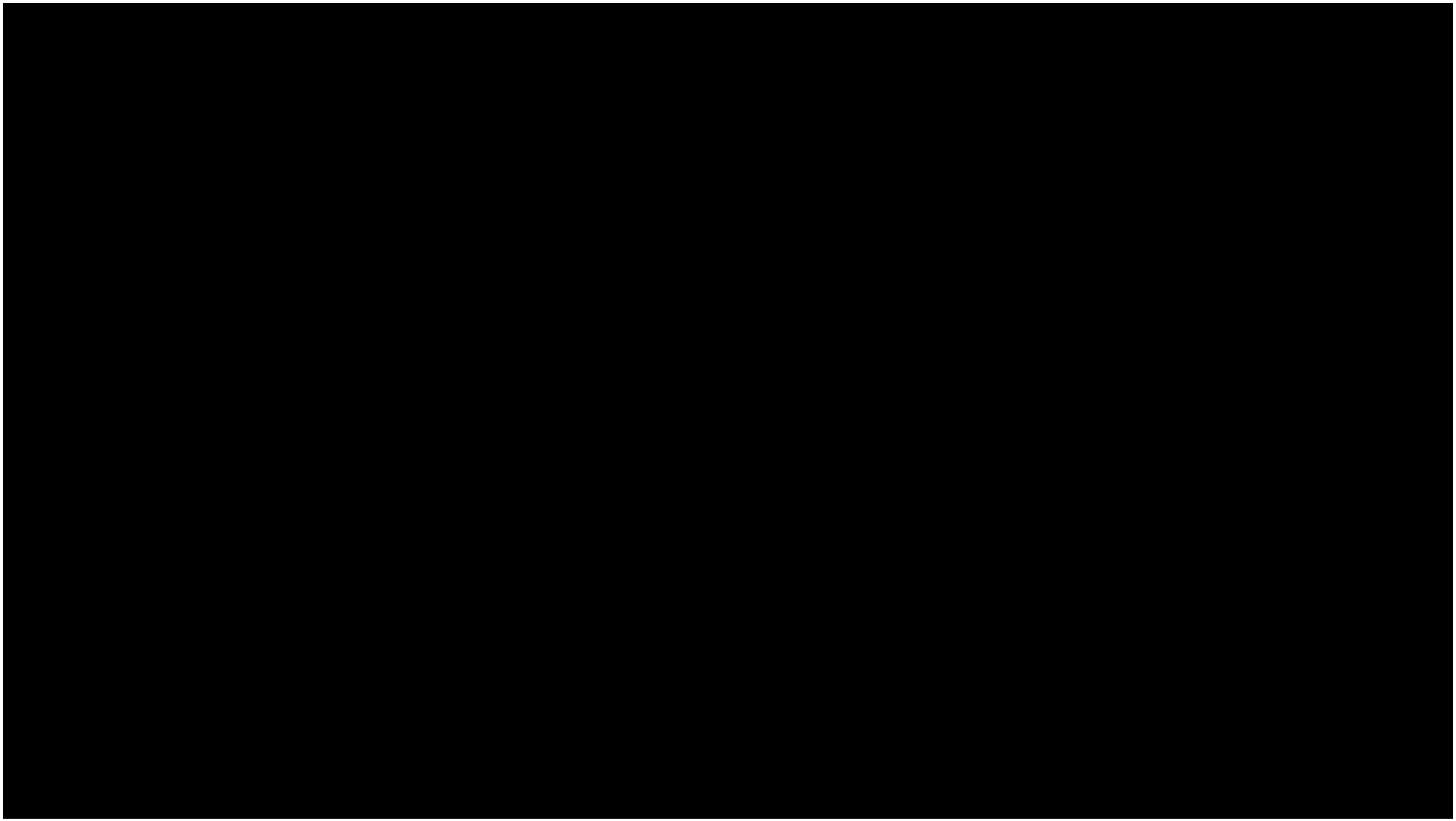
The Qualification Problem

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?

Some Popular Formalisms

- Situation Calculus — McCarthy (1959 - 1963)
  - Situation Calculus Revised — Reiter
- Fluent Calculus — Thielscher
- The A Family of Logics — Lifschitz, Gelfond, Baral
- Features and Fluents — Sandewall
- Temporal Action Logics (TAL) — Doherty

Multi-Agent Iphone Game



LINKÖPING UNIVERSITY

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

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

$Tiger(Fred)$ $Killer(Fred)$

$Tame(Fred)$

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()?

$\neg Ab(Ted)$

$\vdots$

$\neg Ab(Sally)$

Explicit Axiomatization of normal objects is not feasible

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

Technically: Minimize the extension of AB()

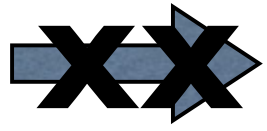
The only things abnormal are those explicitly stated to be abnormal in the theory.

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)

$\Gamma \models \alpha$  $\Gamma \cup \Delta \models \alpha$

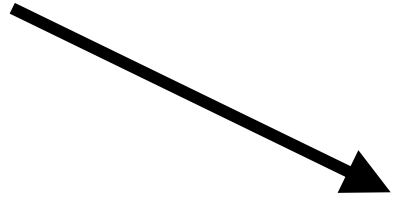
Nonmonotonic Logics do not have the property of monotonicity

Broad Set of Techniques In KR based on this basic idea

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

<u>Common Thread:</u>

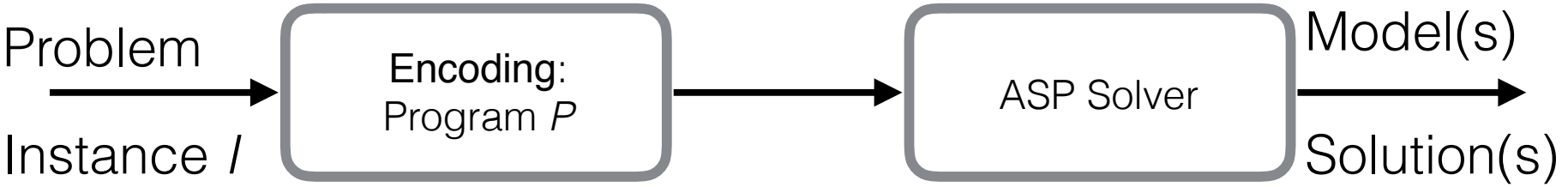
- Make assumptions about incomplete information
- Do this by referring to meta-theoretic concepts
- The entailment relation becomes nonmonotonic



- 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*.

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 "$\wedge$", for the "and" 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:

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

where the l's are literals in Σ.

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.*

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

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

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

$$l_0 \vee , \ldots, \vee l_i \leftarrow$$

Some Examples

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

Absence of truth for Ted being abnormal (default negation)

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

Tony is abnormal

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

Tame tigers are abnormal

Local Closed World Assumption (Predicate Completion):

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

Normally,individuals are not abnormal

Answer Sets: 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)

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

Answer Sets: Informal Example

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

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:

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

Answer Sets: Semantics

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

1. l **if** $l \in S$;
2. *not* l **if** $l \notin S$;
3. $l_1 \vee \ldots l_n$ **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):

It can include: ¬

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

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.

Example

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. $q(a) \leftarrow s(b)$

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 $l \in S$;
2. removing all other premises of the remaining rules containing *not*.

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 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 subsets of literals in heads of rules in a program can be potential answer sets for that program!

Note: For programs with no default negation (not), there is always a unique answer set if the program is consistent.

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

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:

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

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

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.

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

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

$\{\}$ 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!!

0,1 or more AnswerSets

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

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

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

0,1 or more AnswerSets

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

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!

The empty answer set

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

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 the body is false

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

Querying Programs

0,1 or more AnswerSets

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

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

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

0,1 or more AnswerSets

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

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!

The empty answer set

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

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 the body is false

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

Querying Programs

Computing Minimal Answer Sets

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

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: $T_\Pi(X) = X$

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!

Modelling Action and Change using Answer Sets

Back to Action and Change

Let's model using logic and Answer Set Programs!
(Note this a solution sketch)

$\mathscr{T}$ is the set of timepoints

$\mathscr{A}$ is the set of actions

$\mathscr{F}$ is the set of features

$\mathit{Dom}(\mathscr{F}_i)$ is a value domain for $\mathscr{F}_i$

Let's use $holds(t, f, v)$

to assert that a feature *f* has value *v* at time *t*

Let's use $occlude(t+1, f)$

to assert that a feature *f* is free to change value from *t* to *t*+1

Let's use $occurs(t, t+1, a)$

to assert that an action *a* occurs from *t* to *t*+1

Some axioms

For each $f \in \mathcal{F}, t \in \mathcal{T}$

$$\leftarrow holds(t, f, v), holds(t, f, v'), v \neq v'$$

A feature can only have one value at a timepoint

For each $f \in \mathcal{F}, t \in \mathcal{T}$

$$\bigvee_{v \in dom(f)} holds(t, f, v) \leftarrow$$

A feature must have at least one value at a timepoint

Some more axioms

For each $f \in \mathcal{F}, t \in \mathcal{T}$

$$\neg holds(t, f, false) \leftarrow holds(t, f, true)$$
$$\neg holds(t, f, true) \leftarrow holds(t, f, false)$$

$$holds(t, f, true) \leftarrow \neg holds(t, f, false)$$
$$holds(t, f, false) \leftarrow \neg holds(t, f, true)$$

Actions

Assume deterministic actions
negated atoms can be used

One such rule for each effect:

$$holds(t+1, f_0, v_0) \leftarrow holds(t, f_1, v_1), \ldots holds(t, f_n, v_n), occurs(t, t+1, a)$$

Action Effects

Action Preconditions

Action Occurrence

One such rule for each effect:

$$occlude(t+1, f_0) \leftarrow holds(t, f_1, v_1), \ldots holds(t, f_n, v_n), occurs(t, t+1, a)$$

If an action occurs and its preconditions hold then any feature in the action's effects is allowed to change value

Actions: An Example

Schemas that can be grounded:

$$\leftarrow holds(t, \mathsf{at}(agent, x_1, y_1), \mathsf{true}), occurs(t, t+1, \mathsf{goto}(agent, x_1, y_1, x_2, y_2))$$
$$holds(t+1, \mathsf{at}(x_2, y_2), \mathsf{true})$$

$$\leftarrow holds(t, \mathsf{at}(agent, x_1, y_1), \mathsf{true}), occurs(t, t+1, \mathsf{goto}(agent, x_1, y_1, x_2, y_2))$$
$$occlude(t+1, \mathsf{at}(x_2, y_2))$$

$$\leftarrow holds(t, \mathsf{at}(agent, x_1, y_1), \mathsf{true}), occurs(t, t+1, \mathsf{goto}(agent, x_1, y_1, x_2, y_2))$$
$$\neg holds(t+1, \mathsf{at}(x_1, y_1), \mathsf{true})$$

$$\leftarrow holds(t, \mathsf{at}(agent, x_1, y_1), \mathsf{true}), occurs(t, t+1, \mathsf{goto}(agent, x_1, y_1, x_2, y_2))$$
$$occlude(t+1, \mathsf{at}(x_1, y_1))$$

Actions: An Example

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

$\leftarrow holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$
$holds(1, \mathsf{at}(\mathsf{agent1}, 1, 2), \mathsf{true})$

$\leftarrow holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$
$occlude(1, \mathsf{at}(\mathsf{agent1}, 1, 2))$

$\leftarrow holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$
$\neg holds(1, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$\leftarrow holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$
$occlude(1, \mathsf{at}(\mathsf{agent1}, 1, 1))$

“Solving” the Frame Problem

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

Rule out spurious change:

$$holds(t+1, f, v) \leftarrow holds(t, f, v), not\ occlude(t+1, f)$$

$$\neg holds(t+1, f, v) \leftarrow \neg holds(t, f, v), not\ occlude(t+1, f)$$

The only features that are occluded are those that are occluded by an action effect axiom.
consequently, the *not occlude()* premises in the body of the rules above will only be true for features that are not occluded by an action at timepoint t+1..

Rule out spurious action occurrences:

$$\neg occurs(t, t+1, a) \leftarrow not\ occurs(t, t+1, a)$$

The only actions that occur are those that are explicitly stated to occur in the action theory

An Answer Set Program

An Action Theory

$holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

$occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\leftarrow not\ occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\quad \neg occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\leftarrow not\ occurs(1, 2, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\quad \neg occurs(1, 2, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\vdots$

Actions only occur when explicitly stated to occur

Continued…

An Answer Set Program cont'd

$holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

$occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\vdots$ Persistence rules

$\leftarrow holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{gold}, 2, 3), 1)$
$holds(1, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

$\leftarrow \neg holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{gold}, 2, 3), 1)$
$\neg holds(1, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

Similar rules for each grounded timepoint on the timeline

An Answer Set Program cont'd

$holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

$occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\vdots$

Persistence rules

$\leftarrow holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{agent1}, 1, 1), 1)$
$holds(1, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}),$

$\leftarrow \neg holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{agent1}, 1, 1), 1)$
$\neg holds(1, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$\leftarrow holds(1, \mathsf{at}(\mathsf{agent1}, 1, 2), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{agent1}, 1, 2), 2)$
$holds(2, \mathsf{at}(\mathsf{agent1}, 1, 2), \mathsf{true})$

$\leftarrow \neg holds(1, \mathsf{at}(\mathsf{agent1}, 1, 2), \mathsf{true}), not\ occlude(\mathsf{at}(\mathsf{agent1}, 1, 2), 2)$
$\neg holds(2, \mathsf{at}(\mathsf{agent1}, 1, 2), \mathsf{true})$

Similar rules for each grounded timepoint on the timeline

An Answer Set Program cont'd

$holds(0, \mathsf{at}(\mathsf{agent1}, 1, 1), \mathsf{true})$

$holds(0, \mathsf{at}(\mathsf{gold}, 2, 3), \mathsf{true})$

$occurs(0, 1, \mathsf{goto}(\mathsf{agent1}, 1, 1, 1, 2))$

$\vdots$

Additional Axioms

$\neg holds(t, at(o, x_1, y_1), \mathsf{true}) \leftarrow x \neq x_1, y \neq y_y, holds(t, at(o, x, y), \mathsf{true})$

objects can only be at one place at a time

$\vdots$

$\vdots$

Intended Answer Set

Feature	0	1	2	...
at(gold, 2, 3)	true	true	true	true
at(agent1, 1, 1)	true	false	false	false
at(agent1, 1, 2)	false	true	true	true

The frame problem solution:

Requires one additional occlude rule per action rule

Require 1-2 persistence rules per feature

Action rules do not have to state anything about what is not effected, only what is.

Conclusions

The frame problem solution:

Requires one additional occlude rule per action rule

Require 1-2 persistence rules per feature

Action rules do not have to state anything about what is not effected, only what is.

Similar solutions apply for the ramification and qualification problems.

Answer Set Programs provide a basis for “pragmatic” reasoning about action and change

Book

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

Downloadable System:

http://potassco.sourceforge.net/