Knowledge Representation and Logic Programming

Michael Gelfond

University of Texas at El Paso

mgelfond@cs.utep.edu

Plan of the talk

- Introduction.
- A-Prolog, syntax and semantics.
- Representing defaults.
- Incompleteness of information in databases.
- Inheritance reasoning.
- Reasoning with prioritized defaults.
- Agents is dynamic domains.

Introduction

• To make machines smart we need to teach them how to reason and how to learn.

Most teaching is done by instructions.

Most learning comes from instructions.

We need efficient means of communication!

• Languages differ according to the type of information their designers want to communicate to computers. Two basic types:

ALGORITHMIC languages describe sequences of actions for a computer to perform.

DECLARATIVE languages describe properties of objects in a given domain.

Logic based AI: the idea

- **Use declarative language to describe the domain.**
- **Express various tasks as queries to the resulting program.**
- **Use inference engine, i.e. a collection of reasoning algorithms, to answer these queries.**

Representing Knowledge: the basics

• Knowledge represented as a theory (a collection of statements) in a formal language with a consequence relation.

Such theory is often called KNOWLEDGE BASE (KB).

• Representation should be ELABORATION TOLERANT, i.e. small additions to the informal body of knowledge should correspond to small additions to the formal KB.

• Consequence relation should be NONMONOTONIC.

• Query answering systems should be able to answer queries to a knowledge base.

Choice of the language and of the consequence relation depends on the types of statements used in the informal description.

Statements to represent

• Defaults: A's are **normally** B's.

These statements are understood as communication agreements: given x of the type A one should conclude that x satisfies property B unless he has evidence to the contrary.

• Statements expressing lack of information: "Don't know if statement P is true or false."

• Naming assumptions: "There are no unnamed elements in the domain".

• Completeness of information: "Statements not entailed by the knowledge base are false".

Reasoning about defaults: the beginning

- **Extensions of Classical Logic (1980):**

Circumscription (McCarthy), Default Logic (Reiter), Nonmonotonic Logic (McDermott and Doyle), Autoepistemic Logic (Moore)

Complex languages and consequence relations, no inference engines, (ADA of AI)

- **Logic Programming with Negation as Failure**

Normally, comparatively simple to use, good inference engine, but

NO DECLARATIVE SEMANTICS, NOT ENOUGH EXPRESSIVE POWER

Declarative semantics for NOT

- **NOT is defined via Prolog interpreter (Colmerauer, 1974).**
- **First semantics (Clark, Reiter 1978). Great work, not fully adequate.**
- **Canonical model (Apt, Blair and Walker, 1985). Definitive semantics, but defined only for a special class of programs.**
- **Stable Model Semantics (Gelfond and Lifschitz, 1987), (based on work in nonmonotonic logic), Well-founded Semantics (Van Gelder, Ross, Schlipf)**

Extensions of the language

• Classical (explicit) negation (Gelfond and Lifschitz 1990, Pereira, 1993), disjunction (Minker and Lobo, Gelfond and Lifschitz, Przymusinski, etc).

• New knowledge representation language A-Prolog.

The language includes "classical" negation $\neg$ ($\neg p$ means that p is false), negation as failure *not* and "epistemic" *or*.

A-Prolog allows to represent incomplete information. Very expressive. Unlike Prolog, a query is answered YES, NO, or UNKNOWN.

A-Prolog

• A program of A-Prolog is a collection of rules of the form

$f_0\ or\ \ldots\ or\ f_i \leftarrow f_{i+1}, \ldots, f_m,\ not\ \ f_{m+1}, \ldots,\ not\ \ f_n$

where f's are literals, i.e., statements of the form $p(t)$, $\neg p(t)$. The rule says to a reasoner "If you believe in $f_{i+1} \ldots f_m$ and have no reason to believe in $f_{m+1} \ldots f_n$ then you must believe at least one of $f_0 \ldots f_i$".

• Intuitively, declarative program Π can be viewed as formal specification for sets of beliefs which can be held by a rational reasoner on the bases of Π.

• These beliefs are represented by sets of ground literals called answer sets of Π.

• Formula f is entailed by a program Π if f is satisfied by all answer sets of Π.

Answer sets for monotonic programs

Consider a program consisting of rules

$f_0 \; or \; \ldots \; or \; f_i \leftarrow f_{f+1}, \ldots, f_m$

A set S of formulae is closed under a rule if it satisfies its head or does not satisfy its body.

Answer set of a program Π not containing *not* is a minimal set S of ground literals satisfying the following two conditions:

- S is closed under the rules of Π.
- If S contains contrary literals then it is equal to the set of all ground literals (Lit).

Examples

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

  $A = \{\neg p(a)\}$

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

  $A = \{\neg p(a), p(b)\}$

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

  $A = \{\ \ \}$

- $p(a)\ or\ \ p(b).$

  $A1 = \{p(a)\}$ $A2 = \{p(b)\}$

Answer sets of arbitrary programs

Consider program Π^S obtained from Π by

(i) removing all rules containing $not \quad f$ such that S satisfies f

(ii) removing all other premisses containing not

- S is an answer set of Π iff S is an answer set of Π^S

Π	Π^S	$S = \{q(a), p(b)\}$
$p(a) \leftarrow \ not\ q(a)$		
$p(b) \leftarrow \ not\ q(b)$	$p(b) \leftarrow$	
$q(a) \leftarrow$	$q(a) \leftarrow$	

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

The consequence relation

- Program Π entails literal l ($\Pi \models l$) if l belongs to all answer sets of Π.
- If for some l, $\Pi \models l$ and $\Pi \models \neg l$ then Π is called inconsistent.
- Π's answer to l is

  YES if $\Pi \models l$,

  NO if $\Pi \models \neg l$,

  UNKNOWN otherwise.

The consequence relation is **nonmonotonic**, i.e., there are Π, Q, R, such that $\Pi \models Q$ but $\Pi \cup R \not\models Q$.

Examples

- $p(a) \leftarrow \ not\ p(b).$

  ? p(a). - YES

  ? p(b). - UNKNOWN

- $p(a) \leftarrow \ not\ p(b). \qquad \neg p(X) \leftarrow \ not\ p(X).$

  ? p(a). - YES

  ? p(b). - NO

- $p(a) \leftarrow \ not\ p(a).$

  NO ANSWER SET

- $p(a) \leftarrow \ not\ p(b). \qquad p(b) \leftarrow \ not\ p(a).$

  $A1 = \{p(a)\} \qquad A2 = \{p(b)\}$

- $p(a)\ or\ p(b). \qquad \neg p(X) \leftarrow \ not\ p(X).$

  $A1 = \{p(a), \neg p(b)\} \qquad A2 = \{p(b), \neg p(a)\}$

Simple Properties

• If a program Π has a consistent answer set then all its answer sets are consistent.

• For any answer set S of program Π of A-Prolog:

(a) For any rule

* $l_0 \leftarrow l_1, \ldots, l_m,\ not\ l_{m+1}, \ldots,\ not\ l_n$

from Π, if

$\{l_1, \ldots, l_m\} \subseteq S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$

then $l_0 \in S$.

(b) If S is a consistent answer set of Π and $l_0 \in S$ then there exist a rule * from Π such that

$\{l_1, \ldots, l_m\} \subseteq S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$.

Simple Properties (Continued)

Let R be a collection of rules of A-Prolog. R^+ is the result of replacing all occurrences of $\neg l$ in R by $neg(l)$.

• S is an answer set of Π iff S^+ is an answer set of Π^+ not containing literals of the form l, $neg(l)$.

• The consequence relation of A-Prolog is not cumulitive, i.e. there are Π and Q such that

$\Pi \models Q$, $\Pi \models R$ but $\Pi \cup Q \not\models R$.

Proof: (Van Gelder)

$$\begin{cases} p \leftarrow not\ q. \\ q \leftarrow not\ p. \\ r \leftarrow not\ r. \\ r \leftarrow not\ p. \end{cases}$$

What is next?

To use A-Prolog to represent knowledge we need

- better understand the consequence relation and the notion of argument in A-Prolog;
- develop mathematical tools to prove properties of programs of A-Prolog;
- develop methodology of representing knowledge;
- build implementations and develop methodology of their use.

In this talk I concentrate mainly on methodology.

Example: what is an orphan?

We assume that the machine already knows the notions of parent and age. (age(P,0) may indicate that P is not alive).

$$\begin{aligned} orphan(P) &\leftarrow child(P), \\ &\quad not\ has\_a\_parent(P). \end{aligned}$$

$$\begin{aligned} child(P) &\leftarrow age(P, A), \\ &\quad A < 21. \end{aligned}$$

$$\begin{aligned} has\_a\_parent(P) &\leftarrow parent(X, P), \\ &\quad alive(X). \end{aligned}$$

Defaults

Questions:

- What is a default?
- How to specify exceptions to defaults and priorities between them?
- What is the set of valid conclusions entailed by a default theory?
- How to arrive at these conclusions?

What is default?

There are different answers to this question:

• Statement of natural language of the form "Elements of a class C normally (regularly, as a rule) satisfy property P". This statement should be consistent with a statement "x is an exception to this rule. It belongs to C but does not satisfy P".

• Defeasible statements of some nonmonotonic logic, e.g. default rules or logic programming rules with negation as failure.

Usually, the latter is understood as a mathematical model of a former. However, there are many "formal" defaults which model no "informal" ones.

Example

Suppose we are given a list t of people

$in\_t(mike).$ $\quad$ $in\_t(john).$

and want to define the class of people not listed in t. This, of course, can be done by the rule

r1. $unlisted(X) \leftarrow not\ in\_t(X).$

The program entails $unlisted(mary)$.

This conclusion can be defeated by adding

$in\_t(mary)$.

but cannot be defeated by adding

$\neg unlisted(mary)$.

The latter attempt will (justifiably) lead to contradiction.

Representing defaults

- "**a's are normally b's** " translated as

$$r: \quad b(X) \leftarrow a(X),\ not\ \ ab(r,X),\ not\ \ \neg b(X) \qquad (1)$$

An expression $not\ \ \neg b(X)$ is called a guard.

- "**c's are exceptional a's. They do not satisfy b**" translated as

$$\neg b(X) \leftarrow c(X) \qquad (2)$$

- "**c's are exceptional a's. They may or may not satisfy b**" translated as

$$ab(r,X) \leftarrow\ \ not\ \ \neg c(X) \qquad (3)$$

Exceptions of this sort are called *weak exceptions*.

Representing defaults : an example

"Normally, parents care about their children. John is an exception to this rule. He does not care about his."

parent(john,sam). parent(mary,sam).

Default is expressed by the rule:

$$cares(X,Y) \leftarrow parent(X,Y), \; not \; \neg cares(X,Y).$$

John's feelings are described as

$$\neg cares(john,X) \leftarrow parent(john,X).$$

?cares(mary,sam). YES

?cares(john,sam). NO

?cares(sam,sam). UNKNOWN

Defaults (continued)

"Normally, students are afraid of math. Math students are not. Those in CS may or may not be afraid."

$in(john, english).$ $in(mary, cs).$ $in(pat, math).$

$$\neg in(S, D1) \leftarrow in(S, D2), D1 \neq D2.$$

$$afraid(S, math) \leftarrow student(S), not\ ab(S), not\ \neg afraid(S, math).$$

$$ab(S) \leftarrow student(S), not\ \neg in(S, cs).$$

$$\neg afraid(S, math) \leftarrow student(S), in(S, math).$$

? $afraid(john, math).$ YES

? $afraid(mary, math).$ UNKNOWN

? $afraid(pat, math).$ NO

Example: Null values in databases

In databases incomplete information is normally represented by NULL values. Problems:

- How to define answer to a query?

Relational algebra operations do not extend to relations with null values.

Translation to predicate calculus can't deal with the Closed World Assumption: "Normally, if things are not in the tables they are false."

- How to compute the answers?

Proposed Solution

- Use A-Prolog to

  (a) rewrite database tables

  (b) express the closed world assumption.

- Use answer set semantics to define answer to a query.

If Π is a program representing database tables and CWA then answer to a query $Q(X)$ is
$\{t : \Pi \models Q(t)\}$.

- Use SMODELS like systems to answer the query.

Example: Complete database

The following table represents COMPLETE info about the summer schedule of a department:

Professor	Course
mike	pascal
john	c

Program representing this info:

1. $t(mike, pascal)$.

2. $t(john, c)$.

3. $\neg t(X, Y) \leftarrow \ not \ \ t(X, Y)$.

? $t(mike, c)$. NO

Incomplete Database 1

Professor	Course
mike	pascal
john	c
staff	lisp

STAFF is a null value which stands for an unknown professor (possibly different from Mike and John).

How to represent this information?

1. $t(mike, pascal).\quad t(john, c).\quad t(staff, lisp).$

2a. $\neg t(X, Y) \leftarrow\ not\ \ t(X, Y),\ not\ \ ab(X, Y).$

2b. $ab(X, Y) \leftarrow t(staff, Y).$

? $t(mike, c).\ NO,\quad ?t(mike, lisp).\ UNKNOWN$

Incomplete Database 2

Professor	Course
mike	pascal
john	c
{mike,john}	prolog
staff	lisp

where {mike,john} represents the second type of nulls – "value unknown but one of the finite set of values".

1. $t(mike, pascal).\quad t(john, c).\quad t(staff, lisp).$

2. $t(mike, prolog)\ or\ t(john, prolog).$

3a. $\neg t(X, Y) \leftarrow\ not\ \ t(X, Y),\ not\ \ ab(X, Y).$

3b. $ab(X, Y) \leftarrow t(staff, Y).$

$?t(mike, c).\ NO,\quad ?t(mike, prolog).\quad UNKNOWN$

$?\ t(mike, prolog) \wedge t(john, prolog).\quad NO$

Inheritance hierarchy (closed nets)

We identify a net N with a collection of literals of the form $subclass(c_1, c_2)$, $default(d, c, p, +)$, and $default(d, c, p, -)$ corresponding to N's links.

An is-net N informally specifies a function which takes as an input collections of literals formed by predicate symbol is and computes all possible conclusions about relations is and has which a rational agent can obtain from this net. If the membership relation is defined by the net is complete the net is called *closed*.

Defining the membership relation

For closed nets *is* is defined as:

$$
\begin{array}{lcl}
is(O,C) & \leftarrow & subclass(C0,C), \\
 & & is(O,C0). \\
\neg is(O,C) & \leftarrow & not\ is(O,C). \\
is\_subclass(C1,C2) & \leftarrow & subclass(C1,C2). \\
is\_subclass(C1,C2) & \leftarrow & subclass(C1,C3), \\
 & & is\_subclass(C3,C2).
\end{array}
$$

To implement the inheritance principle, *more specific information is more important than less specific one*, we need relations

exception(e,d,+) - positive default d is not applicable to objects of class e.

exceptional(x,d,+) - positive default d is not applicable to object x.

Similarly for negative defaults.

Defaults with exceptions

$$
\left.
\begin{array}{rl}
has(x,p) \leftarrow & default(d,c,p,+), \\
 & is(x,c),\ not\ \neg has(x,p), \\
 & not\ exceptional(x,d,+). \\
\neg has(x,p) \leftarrow & default(d,c,p,-), \\
 & is(x,c),\ not\ has(x,p), \\
 & not\ exceptional(x,d,-). \\
exception(e,d_1,+) \leftarrow & default(d_1,c,p,+), \\
 & default(d_2,e,p,-), \\
 & not\ subclass(c,e). \\
exception(e,d_1,-) \leftarrow & default(d_1,c,p,-), \\
 & default(d_2,e,p,+), \\
 & not\ is\_subclass(c,e). \\
exceptional(x,d,s) \leftarrow & exception(e,d,s), \\
 & is(x,e).
\end{array}
\right\} \pi
$$

Open Nets

Now let us assume that the net is open, i.e. there is no CWA for the membership relation is. The modified definition for is looks as follows:

$$
\begin{aligned}
is(O, C2) \quad &\leftarrow \quad is(O, C1), \\
&\qquad\quad is\_subclass(C1, C2). \\
\neg is(O, C1) \ &\leftarrow \ \neg is(O, C2), \\
&\qquad\quad is\_subclass(C1, C2). \\
\neg is(X, C1) \ &\leftarrow \ is(X, C2), \\
&\qquad\quad not\ is\_subclass(C1, C2), \\
&\qquad\quad not\ is\_subclass(C2, C1).
\end{aligned}
$$

The last rule says that the classes not belonging to the same path are disjoint.

Open Nets (continued)

Consider an open net

Intuitively, the correct answer to a query p(x) is UNKNOWN. The π's answer to it however is YES. The problem can be corrected by replacing the rule for *exceptional* in π by the rule

$$\left.\begin{array}{ll} exceptional(x,d,s) \leftarrow & exception(e,d,s), \\ & not\ \neg is(x,e). \end{array}\right\} R$$

which says "do not apply d to x which *may belong* to exceptional class".

Reasoning with prioritized defaults

We need to reify defaults. Let σ contain names of objects, functions, and relations.

Literals of $\mathcal{L}(\sigma)$ are:

1. literals of σ
2. $rule(r,\ l_0,\ \Gamma)$
3. $default(d,\ l_0,\ \Gamma,\ \Delta)$
4. $exception(d,\ \Gamma,\ \Delta)$
5. $prefer(d_1, d_2)$
6. $conflict(d_1, d_2)$

Γ, Δ - sets of literals; $not\ \Delta = \{\ not\ l : l \in \Delta\}$.

Sets of ground literals of $\mathcal{L}(\sigma)$ are called **domain descriptions**.

Informal Semantics

- $rule(r,\ l,\ \Gamma)$

  $l \leftarrow \Gamma$

- $default(d,\ l,\ \Gamma,\ \Delta)$

  If X satisfies Γ and $not\ \Delta$ then it satisfies l.

- $exception(d,\ \Gamma,\ \Delta)$

  Default d is not applicable to object X

  which satisfies Γ and $not\ \Delta$.

- $prefer(d_1, d_2)$

  If defaults d_1, d_2 are in conflict with each other

  and d_1 is applicable to X then d_2 is not.

- $conflict(d_1, d_2)$

  Defaults d_1 and d_2 have conflicting conclusions.

Query Language

Relations $holds(l)$ and $holds\_by\_default(l)$ are defined on literals of $\mathcal{L}(\sigma)$.

The query language associated with domain descriptions of $\mathcal{L}(\sigma)$ consist of ground atoms of the form

$holds\_by\_default(l)$

$holds(l)$

and their negations.

Main Question: What is the entailment relation $\mathcal{D} \models q$?

Axiomatic Approach: Find a set of axioms $\mathcal{P}_\sigma$ capturing general properties of defaults.

$\mathcal{D} \models q$ if q belongs to every answer set of the program

$\mathcal{P}_\sigma(\mathcal{D}) = \mathcal{P}_\sigma \cup \{holds(l) \mid l \in fact(\mathcal{D})\} \cup laws(\mathcal{D})$.

Non-defeasible Inference:

$$holds(L) \leftarrow rule(R, L, Body), \quad (4)$$
$$hold(Body).$$

$$hold([\,]). \quad (5)$$

$$hold([H|T]) \leftarrow holds(H), \quad (6)$$
$$hold(T).$$

The first axiom defines the relation *holds* which is satisfied by a $\mathcal{L}(\sigma)$ literal l iff l is non-defeasibly true in the domain description $\mathcal{D}$. The next two axioms define the same relation on the lists of literals in $\mathcal{L}(\sigma)$, i.e., $hold([l_1, \ldots, l_n])$ iff all l's from the list are true in $\mathcal{D}$.

Defeasible Inference:

$$holds\_by\_default(L) \leftarrow holds(L). \quad (7)$$

$$holds\_by\_default(L) \leftarrow rule(R, L, \Gamma), \quad (8)$$
$$hold\_by\_default(\Gamma).$$

$$holds\_by\_default(L) \leftarrow default(D, L, \Gamma, \Delta), \quad (9)$$
$$hold\_by\_default(\Gamma),$$
$$fail\_by\_default(\Delta),$$
$$not\ defeated(D),$$
$$not\ holds\_by\_default(\neg L).$$

$$hold\_by\_default([\,]). \tag{10}$$

$$hold\_by\_default([H|T]) \leftarrow holds\_by\_default(H),\ hold\_by\_default(T). \tag{11}$$

$$fail\_by\_default([\,]). \tag{12}$$

$$fail\_by\_default([H|T]) \leftarrow not\ holds\_by\_default(H),\ fail\_by\_default(T). \tag{13}$$

Defeating defaults:

$$defeated(D) \leftarrow default(D, L, \Gamma, \Delta), \quad (14)$$
$$holds(\neg L).$$

$$defeated(D) \leftarrow default(D, L, \Gamma, \Delta), \quad (15)$$
$$default(D_1, L_1, \Gamma_1, \Delta_1),$$
$$holds(conflict(D_1, D)),$$
$$holds\_by\_default(prefer(D1, D)),$$
$$hold\_by\_default(\Gamma_1),$$
$$fail\_by\_default(\Delta_1),$$
$$not\ defeated(D_1).$$

Defeating defaults (continued)

$$defeated(D) \leftarrow \tag{16}$$
$$exception(D, Positive, Negative),$$
$$hold\_by\_default(Positive),$$
$$fail\_by\_default(Negative).$$

Preference

$$\neg holds(prefer(D_1, D_2)) \leftarrow \tag{17}$$
$$holds(prefer(D_2, D_1)),$$
$$D_1 \neq D_2.$$

Axioms for conflict

$$holds(conflict(d_1, d_2)) \tag{18}$$

for any two defaults with contrary literals in their heads and for any two defaults whose heads are of the form $prefer(d_i, d_j)$ and $prefer(d_j, d_i)$ respectively.

$$\neg holds(conflict(D, D)) \tag{19}$$

$$holds(conflict(D_1, D_2)) \leftarrow holds(conflict(D_2, D_1)). \tag{20}$$

Example: defaults

Consider two legal default rules:

• Normally, a person who cannot be shown to be a minor has the capacity to perform legal acts.

• In order to exercise the right to vote the person has to demonstrate that he is not a minor.

The defaults can be represented as

1. $default(d_1(x), has\_legal\_capacity(x), [\,], [minor(x)])$.

2. $default(d_2(x), has\_right\_to\_vote(x), [\neg minor(x)], [\,])$.

Let $\mathcal{D}$ be (1), (2) and $\neg minor(mike)$.

$\mathcal{D} \models has\_right\_to\_vote(mike)$.

$\mathcal{D} \not\models has\_right\_to\_vote(mary)$.

$\mathcal{D} \models has\_legal\_capacity(mary)$.

Example: Priorities

The **signature** of our domain contains the following relations:

$possession$ - "Mike is in possession of the ship"

$perfected$ - "Mike's ownership of the ship is perfected"

$filed$ - "Mike filed the financial statement about possession of the ship".

$federal(D)$ - "D is a federal law"

$state(D)$ - "D is a state law"

$more\_recent(D_1, D_2)$ - "Law D_1 is more recent than D_2"

Domain Description

Facts about Mike's ownership:

$possession.$

$\neg filed.$

Legal laws about perfecting the ownership:

$default(d_1,\ perfected,\ [possession], [\,]).$

$default(d_2,\ \neg perfected,\ [\neg filed], [\,]).$

and their status:

$more\_recent(d_1, d_2).$

$federal(d_2).$

$state(d_1).$

Legal principles for resolving conflicts

$default(d_3(D_1, D_2),\ prefer(D_1, D_2),$

$[more\_recent(D_1, D_2)], [\,]).$

$default(d_4(D_1, D_2),\ prefer(D_1, D_2),$

$[federal(D_1), state(D_2)], [\,]).$

Complete knowledge of laws status

$default(d_5(D_1, D_2), \neg more\_recent(D_1, D_2), [\,], [\,]).$

$default(d_6(D), \neg federal(D), [\,], [\,]).$

$default(d_7(D), \neg state(D), [\,], [\,]).$

The resulting program $\mathcal{P}(\mathcal{D}_0)$ has answer sets A_1 and A_2:

(i) $holds\_by\_default(perfected) \in A_1$

(ii) $\neg holds\_by\_default(perfected) \in A_2$

To resolve the ambiguity we add

$prefer(d4(D_1, D_2),\ d3(D_2, D_1))$.

Now A_2 is the only answer set of the program $\mathcal{P}(\mathcal{D}_1)$.

Changing the mode of reasoning

The theory $\mathcal{P}$ considered so far allows alternative sets of beliefs about the world. This is a powerful mode of reasoning which includes reasoning by cases. The alternative may be to construct $\mathcal{P}_c$ which blocks application of both defaults. This can be done by the axiom

$$
\begin{aligned}
defeated(D) \leftarrow\ & default(D, L, Body), \\
& default(D_1, L_1, Body_1), \\
& holds(conflict(D_1, D)), \\
& not\ holds\_by\_default(prefer(D1, D)), \\
& not\ holds\_by\_default(prefer(D, D1)), \\
& hold\_by\_default(Body), \\
& hold\_by\_default(Body_1).
\end{aligned}
\tag{21}
$$

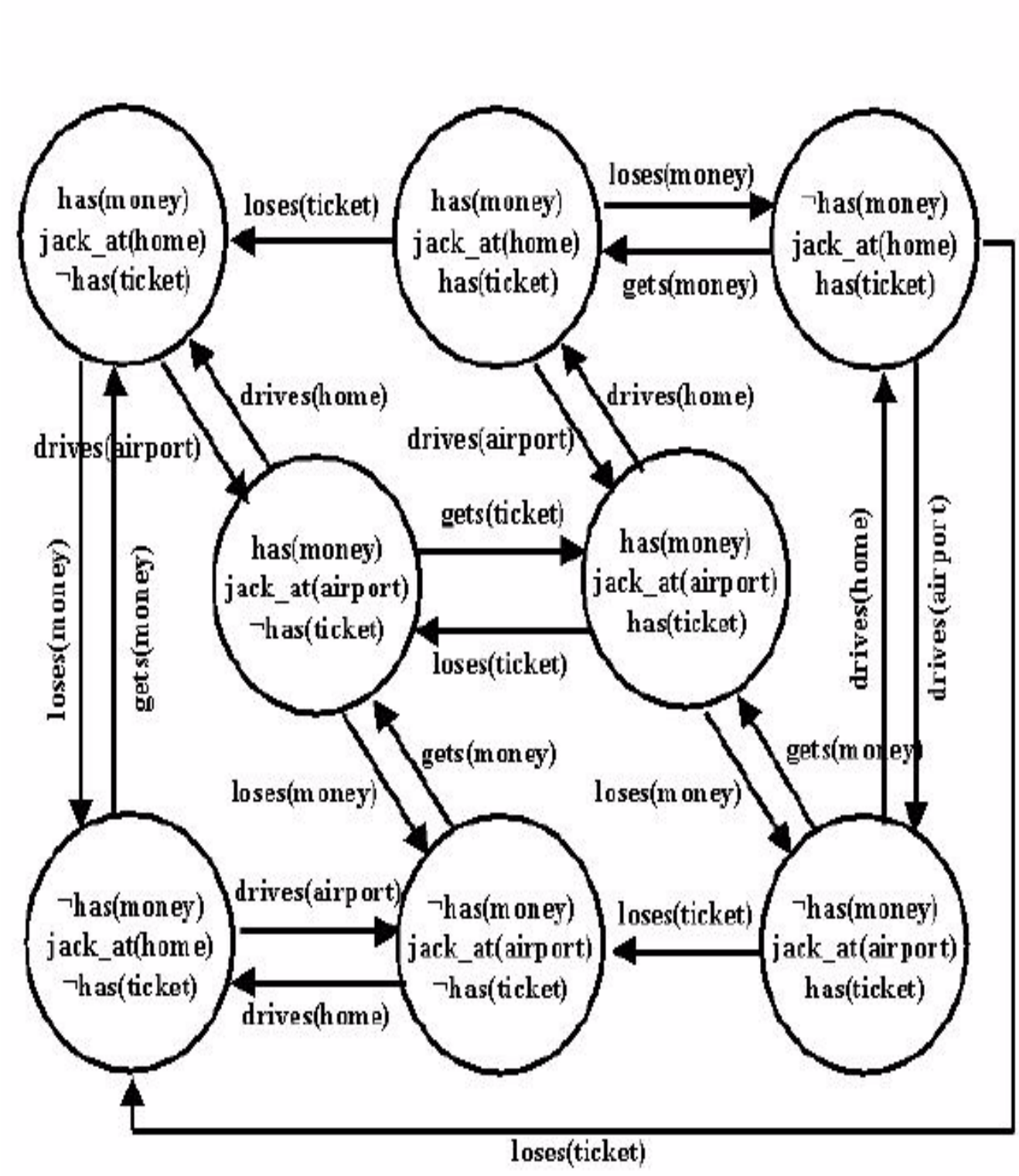
Example: acting agents

- **Problem: how to design software components of agents capable of reasoning, planning and acting in a changing environment.**
- **Simplifying Assumptions:**

1. The environment can be viewed as a transition diagram whose states are sets of fluents and whose arcs are labeled by actions.

2. The agent is capable of making correct observations, performing actions, and remembering the domain history.

The diagram


has(money)
jack_at(home)
¬has(ticket)
loses(ticket)
has(money)
jack_at(home)
has(ticket)
loses(money)
gets(money)
¬has(money)
jack_at(home)
has(ticket)
drives(home)
drives(airport)
drives(home)
drives(airport)
loses(money)
gets(money)
has(money)
jack_at(airport)
¬has(ticket)
gets(ticket)
loses(ticket)
has(money)
jack_at(airport)
has(ticket)
drives(home)
drives(airport)
gets(money)
loses(money)
gets(money)
loses(money)
¬has(money)
jack_at(home)
¬has(ticket)
drives(airport)
drives(home)
¬has(money)
jack_at(airport)
¬has(ticket)
loses(ticket)
¬has(money)
jack_at(airport)
has(ticket)
loses(ticket)

Difficulties

For any state σ and any action A we need to define possible successor states. To do that we need to

- **Understand causal reasoning in the presence of complex interrelations between fluents.**
- **Represent the inertia axiom: "Normally, actions do not change values of fluents."**

Recent developments in a subfield of AI, called "reasoning about actions", substantially clarified the causality question. (McCain, Turner, Lifschitz, Baral, Gelfond, Thielscher, Sandwall)

Describing the diagram: the language

- **Fluents $F_1, F_2, \ldots$ and actions $A_1, A_2, \ldots$;**
- **Causal Laws**

(a) causes$(A, F, [P_1, \ldots, P_n])$ - action A causes fluent literal F to be true after A is executed in any state which satisfies $P_1, \ldots, P_n$.

(b) caused$(F, [P_1, \ldots, P_n])$ - fluent literal F is true because fluent literals $P_1, \ldots, P_n$ are true.

- **Executability Conditions**

(c) impossible$(A, [P_1, \ldots, P_n])$ - action A cannot occur in a state satisfying $P_1, \ldots, P_n$.

Describing the diagram (continued)

The discovery: there is a program P of A-Prolog such that a state S' is a successor state of a state S w.r.t. action A if S' is an answer set of the program $P \cup S$.

Computing successor states is reduced to computing answer sets!

Describing history

- Time points: $1, 2, \ldots$
- Axioms:

  (a) **observed**(**F**, **T**) - fluent literal F is observed

  to be true at the moment T;

  (b) **occurs**(**A**, **T**).

- A set of axioms defines a collection of paths in the diagram which can be interpreted as possible histories of the domain. If our knowledge is complete and actions are non-deterministic there is only one such path.

Describing Queries

T is a moment in the past.

- **? holds_at(F, T) - did F hold at moment T?**
- **? holds_at(F) - does F currently hold?**
- **? holds_after(F, α, T) - would F be true if α were executed at T?**
- **? holds_after(F, α) - will F be true after the execution of α in the current situation?**
- **? holds_after(F, X) - find plan X to achieve F.**

The consequence relation

A - description of actions, Γ - set of axioms, Q - query

$$\Gamma \models_A Q$$

Problems of reasoning, planning, explaining observations, etc. can be reduced to computing this consequence relation.

This, in turn, can be reduced to answering queries in A-Prolog!

Architecture for Intelligent Agents

The agent memory contains action description A, the history Γ, and the set of goals $\mathcal{G}$. It executes the loop:

1. observe (expand axioms by new observations)
2. select_goal(G);
3. plan, i.e. find $a_1, \ldots, a_n$ such that

$$\Gamma \models_A holds\_after(G, [a_1, \ldots, a_n]);$$

4. $execute(a_1)$;

At stages (1) and (4) the axioms Γ are expanded by new information about actual occurrences of events (both, caused by and observed by the agent), and observations of fluents.

Example: Jack's trip to the airport

Types :

$location(home).$ $location(airport).$

$object(money).$ $object(ticket).$

Fluents :

$jack\_at(L) \leftarrow location(L).$

$has(O) \leftarrow object(O).$

Actions :

$drive(L) \leftarrow location(L).$

$get(O) \leftarrow object(O).$

$lose(O) \leftarrow object(O).$

Trip to the airport: Laws and History

Causal Laws :

$$\left.\begin{array}{l} impossible(drive(L), jack\_at(L)). \\ causes(drive(L), jack\_at(L), none). \\ impossible(get(ticket), \neg has(money)). \\ impossible(get(ticket), \neg jack\_at(airport)). \\ impossible(get(O), has(O)). \\ causes(get(O), has(O), none). \\ causes(lose(O), \neg has(O), none). \\ caused(\neg jack\_at(L1), jack\_at(L2)). \text{if} L1 \neq L2 \end{array}\right\} A$$

Axioms :

$$\left.\begin{array}{l} observed(has(money), 0). \\ observed(\neg has(ticket), 0). \\ observed(jack\_at(home), 0). \end{array}\right\} \Gamma_0$$

Jack in action

1. observes Γ_0 and selects the goal has(ticket);
2. solves $\Gamma_0 \models_A holds\_after(has(ticket), X)$,

   $X = [drive(airport), get(ticket)]$;
3. executes the first action,

   $\Gamma_1 = \Gamma_0 \cup \{occurs(drive(airport), t_0)\}$;
4. observes $\neg has(money)$,

   $\Gamma_2 = \Gamma_1 \cup \{holds\_at(\neg has(money), t_1)\}$;
5. solves $\Gamma_2 \models_A holds\_after(has(ticket), X)$

   $X = [get(money), get(ticket)]$.

Modeling Jack's behavior

Jack's behavior was modeled by a computer program which solves the corresponding equations by calling S-models or DLV.

The idea: Use A-Prolog to represent a diagram (DIA) and a class of possible future paths of a given length which may contain a plan (CM).

Observation: Given an action description A, complete history H with a current moment n, maximum length of a plan, l, and a goal G:

$a_1, \ldots, a_k$ is a plan for G iff

$occurs(a_1, n)$,

...,

$occurs(a_k, n + k)$

belongs to an answer set of A + H + DIA + CM + G.

Domain independent axioms (DIA)

$$
\begin{aligned}
holds(F, T1) \quad &\leftarrow next(T, T1), fluent(F), action(A), \\
&\quad causes(A, F, P), \\
&\quad holds(P, T), \\
&\quad occurs(A, T).
\end{aligned}
$$

$$
holds(none, T) \leftarrow time(T).
$$

$$
\begin{aligned}
holds(F, T) \quad &\leftarrow time(T), fluent(F), \\
&\quad caused(F, G), \\
&\quad holds(G, T).
\end{aligned}
$$

$$
\begin{aligned}
holds(F, T) \quad &\leftarrow time(T), fluent(F), \\
&\quad observed(F, T).
\end{aligned}
$$

DIA continued

$$
\begin{aligned}
holds(F, T1) \leftarrow\ & next(T, T1), fluent(F), \\
& holds(F, T), \\
& not\ ab(F, T).
\end{aligned}
$$

$$
\begin{aligned}
ab(\overline{F}, T) \leftarrow\ & time(T), fluent(F), action(A), \\
& causes(A, F, P), \\
& holds(P, T), \\
& occurs(A, T).
\end{aligned}
$$

$$
\begin{aligned}
ab(\overline{F}, T) \leftarrow\ & next(T, T1), fluent(F), \\
& caused(F, G), \\
& holds(G, T1).
\end{aligned}
$$

Describing possible futures (CM)

The following rule can be used to look for sequential plans.

$$occurs(A_1, T) \text{ or...or } occurs(A_n, T) \leftarrow$$
$$time(T), current(Tc), T \geq Tc,$$
$$agent\_action(A_1),$$
$$...$$
$$agent\_action(A_n),$$
$$not\ goal(T).$$

Parallel plans can be found using

$$occurs(A, T) \text{ or } occurs(\neg A, T) \leftarrow$$
$$time(T), current(Tc), T \geq Tc,$$
$$agent\_action(A),$$
$$not\ goal(T).$$

Control module (continued)

We also need constraints:

$$\leftarrow\ time(T), occurs(lose(money), T),$$
$$\quad occurs(get(ticket), T).$$

$$\leftarrow\ time(T), fluent(F), action(A),$$
$$\quad impossible(A, F), holds(F, T), occurs(A, T).$$

Control strategy can be much more sophisticated. For instance, to say that we should try action $a2$ only if $a1$ is impossible we can introduce a new, complex action, a and add rules

$$occurs(a1, T) \leftarrow\ occurs(a, T),$$
$$\quad not\ impossible(a1, T).$$

$$occurs(a2, T) \leftarrow\ occurs(a, T),$$
$$\quad impossible(a1, T).$$

Have declarative specification of control!

Modeling the trip (continued)

Select goal:

$$goal(T) \leftarrow time(T),$$
$$holds(has(ticket), T).$$

A program $A + \Gamma_0 + DIA + CM + goal(lasttime)$ together with a parameter lasttime $= 3$ is given as input to SMODELS. It returns several plans, the shortest being

$occurs(drive(airport), 0), occurs(get(ticket), 1).$

Perform the first action, i.e. add

$occurs(drive(airport), 0).$

to the history. Observe, that money is lost

$occurs(lost(money), 0).$

Verify the rest of the plan. Need to replan. Call SMODELS again with history Γ_1, etc.

Explaining observations

On the previous slide we recorded the loss of money by

2. $occurs(lost(money), 0)$.

which can be viewed as an explanation of the original observation

1. $observed(\neg has(money))$.

Notice that if (1) is used in history Γ_1 instead of (2), the resulting program will have no model. If this is detected by SMODELS, the explanation (2) of (1) can be found by examining answer sets of the program

$A + \Gamma_1 + DIA +$ the rule

$$occurs(A, T) \text{ or } occurs(\neg A, T) \leftarrow time(T), current(Tc), T < Tc, exogenous\_action(A), not\ goal(T).$$

Current work

- **Execution time of S-models and DLV significantly depends on the form of their input. Finding representation of domain knowledge which improves efficiency of planning is in some ways similar to the work on finding proper data structures in procedural languages. This is one of our priorities now.**
- **Larger applications - the shuttle control.**
- **We still need better understanding of default reasoning, e.g. how to conveniently represent prioritized defaults.**

HUGE PROGRESS BUT MANY UNANSWERED QUESTIONS!