Lecture Two: Answer Set Programming - Definitions

301315 Knowledge Representation and Reasoning

Logic Based Approach to AI - The Languages

- Algorithmic - describe sequences of actions for a computer to perform
- Declarative - describe properties of objects and relations between them
- Logic-based approach to AI proposes to:
  - use a declarative language to describe the domain
  - express various tasks (like planning or explanations of unexpected observations) as queries to the resulting program
  - use an inference engine (a collection of reasoning algorithms) to answer these queries

Logic Based Approach to AI - Declarative Programs

```
father(john,sam).
mother(alice,sam).
gender(john,male).
gender(sam,male).
gender(alice,female).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
child(X,Y) :- parent(Y,X).
```

Feed the program to an inference engine and ask it questions (queries): Is Sam the child of John? Who are Sam's parents?

```
? child(sam,john).
? parent(X,sam).
```

Note: This is similar to how we check to see what a human knows. Does it know that Sam is John's son?

Logic Based Approach to AI - Features of Logic Programming

- Knowledge is represented in a precise mathematical language
- Search problems are expressed as queries
- An inference engine is used to answer queries
- The program is *elaboration tolerant*, that is, if small changes in specifications do not cause global problems change

Defeasible Reasoning - Definition

Definition

A reasoning process is *defeasible* when the corresponding argument is rationally compelling but not deductively valid.

Example

1. Normally, birds can fly
2. Normally, when the grass is wet in the morning, it rained last night
3. Normally, a full-time student is unemployed

Defeasible Reasoning - Commonsense Reasoning

A number of nonmonotonic logics were proposed in 1980, which started a new era of AI foundation research.

- John McCarthy developed *circumscription* (1980)
- Drew McDermott and Jon Doyle developed *nonmonotonic logics* (1980)
- Ray Reiter developed *default logic* (1980)
- Robert Moore developed *autoepistemic logic* which served as a starting point for the development of ASP (1980)

The Birth of ASP

ASP is a kind of nonmonotonic reasoning.

Example

```
fly(X) :- bird(X), not emu(X).
bird(tweety).
```

We will conclude "`fly(tweety)`". But if we add new information into the program:

```
fly(X) :- bird(X), not emu(X).
bird(tweety).
emu(tweety).
```

We will not be able to conclude that "`fly(tweety)`".

The Birth of ASP

The ASP approach to AI

- it is declarative - the programmer only deals with what to do, does not deal with how to do;
- it separates knowledge representation and algorithm, allowing the same knowledge base to be used for a variety of reasoning tasks;
- it is state-of-the-art and is explored by a lively community of researchers around the world;
- it has applications in diverse domains;
- it is elegant.

The Birth of ASP

ASP have been applied in the following AI areas

- planning
- data and ontology query answering
- multiagent systems
- knowledge system update and revision
- diagnostics
- semantic web
- Robotics

The Birth of ASP

ASP have also been applied in other areas, such as

- bioinformatics
- software engineering
- automated product configuration
- decision support systems

Syntax - ASP Building Blocks

Signature

$\Downarrow$

Terms

$\Downarrow$

Atoms

Connectives

$\neg$	classical negation
not	default negation
$\leftarrow$	if
or	disjunctive or

Atoms plus connectives allow us to construct rules.

Syntax - The Signature

- The building blocks of ASP are
  - objects $\mathcal{O}$
  - functions $\mathcal{F}$
  - predicates (i.e., relations) $\mathcal{P}$
  - variables $\mathcal{V}$
- This is known as program **signature** $\Sigma = \{\mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{V}\}$.
- Functions and predicates have an arity associated with them.
- **arity** - a non-negative integer indicating the number of parameters.
- Whenever necessary, we assume that our signatures contain standard names for non-negative integers, functions, and relations of arithmetic (e.g., $+$, $*$, $\leq$).

Syntax - The Signature

10 min classroom exercise

Example

What is the signature Σ of the following program?

```
father(john,sam).
mother(alice,sam).
gender(john,male).
gender(sam,male).
gender(alice,female).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
child(X,Y) :- parent(Y,X).
```

Syntax - Terms

- **terms**:
  - Variables and object constants are terms.
  - If `t1`, $\cdots$, `tn` are terms and `f` is a function symbol of arity `n`, then `f(t1,`$\cdots$`,tn)` is also a term.
- **ground terms** are terms containing no symbols for arithmetic functions and no variables.
- Examples from our program:
  - john, sam, and alice are ground terms;
  - X and Y are terms that are variables;
  - father (X,Y) is not a term.
- If a program contains natural numbers and arithmetic functions, then both $2+3$ and 5 are terms; 5 is a ground term while $2+3$ is not.

Syntax - Atoms and Literals

- An **atom** is an expression of the form $p(t1, \cdots, tn)$ where p is a predicate symbol of arity n and $t1, \cdots, tn$ are terms.
- If the signature is sorted, these terms should correspond to the sorts assigned to the parameters of p.
- If p has arity 0 then parentheses are omitted.
- A **literal** is an atom or its negation.

Syntax - Atoms and Literals

Example

- $father(john, sam)$ is an atoms of signature Σ.
- $father(john, X)$ is an atom of signature Σ.
- $father(john, sam)$ and $\neg father(john, sam)$ are literals of Σ.

A **program** Π of ASP consists of a signature Σ and a collection of **rules** of this form:

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

where each l is a literal of Σ.

Syntax - Rules and Programs

- Symbol *not* is a new logical connective called default negation; *not* l is often read as "*it is not believed that* l *is true.*"
- **disjunction** l_1 *or* l_2 is often read as "l_1 *is believed to be true or* l_2 *is believed to be true.*"

Example

$$unemployed(X) \ or \ partTime\_work(X) \leftarrow fullTime\_student(X)$$

All full-time student are either unemployed or doing part-time job.

Syntax - Grounding

The set of ground instantiations of rules of program Π is called the **grounding** of Π.

Example

Given program Π with the signature Σ where

$$\begin{aligned}
&\mathcal{O} = \{a, b\} \\
&\mathcal{F} = \emptyset \\
&\mathcal{P} = \{p, q\} \\
&\mathcal{V} = \{X\}
\end{aligned}$$

and program $\Pi = \{p(X) \leftarrow q(X)\}$

The grounding of Π is a program Π' consists of the following rules:

$$\begin{aligned}
&p(a) \leftarrow q(a) \\
&p(b) \leftarrow q(b)
\end{aligned}$$

Two Case Studies

Example

Let us consider the following ASP program Π_1:

$$\begin{array}{l} fly(X) \leftarrow bird(X), not\ ab(X). \\ ab(X) \leftarrow penguin(X). \\ bird(X) \leftarrow penguin(X). \\ bird(tweety). \\ penguin(skippy). \end{array}$$

- What is the signature Σ of Π_1?
- What facts can be derived from Π_1?

Two Case Studies

Example

Consider the following ASP program Π_2:

$$
\begin{aligned}
&ancestor(X, Y) \leftarrow parent(X, Y).\\
&ancestor(X, Y) \leftarrow parent(X, Z), ancestor(Z, Y).\\
&parent(alice, bob).\\
&parent(bob, carol).\\
&parent(dean, eric).\\
&parent(bob, dean).
\end{aligned}
$$

- Understand Π_2 with transitive rules.
- Find all objects matching X and Y for $ancestor(X, Y)$ to be derived from Π_2.

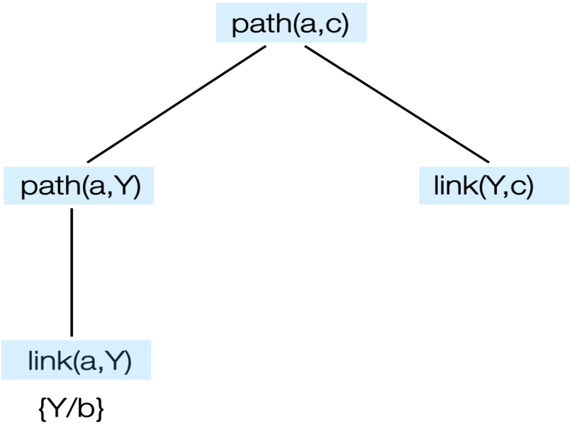
Derivation Tree

Derivation Tree, sometimes, also called *proof tree*, is a useful tool to represent the derivation process of a logic program.
Consider the following program Π^* as follows:

$$
\begin{aligned}
&path(X, Z) \leftarrow link(X, Z).\\
&path(X, Z) \leftarrow path(X, Y), link(Y, Z).\\
&link(a, b).\\
&link(b, c).
\end{aligned}
$$

Then we know that the fact $path(a, c)$ would be derived from Π^*.
This derivation can be actually represented by the following derivation tree.

Derivation Tree



Derivation Tree

Note

It should be noted that such derivation tree would not work if a logic program containing rules with default negation *not*.

Tutorial and Lab Exercises

1. Execute three programs discussed in sections Two Case Studies and Derivation Tree of this lecture using *clingo* on your personal computer/laptop.
2. For program Π_2 in section Two Case Studies, establish the complete derivation tree for the fact $ancestor(alice, eric)$.