Today

- Quiz 11
- Logic Programming Basics

Assignments

- HW 11 out
- Extra credit 1 out (due by end of semester)
- Proj Presentations next Thurs
Logic Programming (Overview)

Basic idea: Formal logic as a language syntax and computation model

- to specify computation problems as a set of facts and logical rules
- e.g., propositional logic ... \( p \land q \rightarrow r \)
- or using predicate logic ... \( p(x, y) \land q(x, z) \rightarrow r(x) \)

Some features of logic programming

- Extremely simple syntax
- (More) Declarative (what not how)
- Computation via “rules” and “constraints” (tricky at first)
  - Tends to feel more like specifying a solution than computing one
- Efficient at certain problems
- Usually “weak” (or agnostic) on data types

Examples of logic-programming in practice

- SQL (although not immediately obvious or designed this way)
- Datalog (more advanced version of SQL)
- Prolog (one of the first mainstraint LP languages)
- Answer Set Programming (we’ll dive into this a bit)
- Erlang (concurrent/parallel ... mixes logic and functional)
Where we are going ...

Exercise: Sketch out an algorithm that:

• takes a graph as input (with nodes and edges)
• assigns one of four colors to each node (“colors” the graph)
• however, no two adjacent nodes can have the same color

The “greedy” approach ...

• Assume colors are ordered (e.g., 0, 1, 2, 3) and nodes are ordered
• Iterate through each node
• Assign the next available color to the node
  – look at each of its adjacent node colors
  – pick the next available color

Using Answer-Set-Programming (in clingo)

% get the nodes
node(X) :- edge(X,\_).
node(X) :- edge(\_,X).

% guess a color for each node (better to use a choice rule)
color(X,r) | color(X,b) | color(X,g) | color(X,y) :- node(X).

% constrain adjacent colors
:- edge(X,Y), color(X,C), color(Y,C).

Note: This program can also find all possible colorings!
**Exercise:** Sketch an algorithm to solve [https://xkcd.com/287/](https://xkcd.com/287/) for any menu and total price

One solution using Answer-Set-Programming (in clingo)

```prolog
#const total = 1505. % target total price
amount(0..7). % amount(0). amount(1). etc.

% the appetizer menu
item_price(fruit,215). item_price(fries,275).
item_price(salad,335). item_price(wings,355).
item_price(stick,420). item_price(plate,580).

% grab the items
item(F) :- item_price(F,_).

% guess an amount to order ("choice" rule)
1 {order(F,N) : amount(N)} 1 :- item(F).

% calculate the item totals
total_order(F,N,T) :- order(F,N), item_price(F,A), T = N*A.

% constrain to be 1505 total
:- total != #sum { T,F : total_order(F,N,T) }.

#show total_order/3. % only show the total_order relation
```

Consider the following propositional statements.

$p$ ... proposition $p$ is true

$q \leftarrow p$ ... if $p$ is true, $q$ is true

Q: What do we know to be true as a result of these statements?

- We know that $p$ is true (given as a “fact”)
- We know that since $p$ is true, $q$ must also be true (inferred from the rule)

This result is called an “answer” of the “program”

- Referred to as an “answer set”: $\{p, q\}$ ... or a “stable model”

Using clingo (an ASP system), we write:

```
p. % p is a fact
q :- p. % same as p -> q
```

Which then gives us the answer set:

```
$ clingo prog.lp
Reading from prog.lp
Solving...
Answer: 1
p q
SATISFIABLE
```

- Models : 1
- Calls : 1
- Time : 0.020s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
- CPU Time : 0.000s
Basic terminology:

Given a rule

\[ r \leftarrow p \land q \]
\[ r : p, q. \]

- \( r \) is called the “**head**” of the rule.
- \( p \land q \) is called the “**body**” of the rule ... read “\( p \) and \( q \) are both true”

Informally, Answer Sets ...

1. Satisfy **all rules** of the program ...
   - if the answer includes the body it also includes the head
   - if the answer doesn’t include the body, the rule is trivially satisfied

2. Do not contain **contradictions** (e.g., \( p \) and \( \neg p \))

3. Are **minimal**
   - the **Rationality Principle**: “Believe nothing you are not forced to believe”
   - always forced to believe asserted facts
   - each answer set is as small as it can be
Q: What is the “answer” (answer set) of the following?

In propositional logic:

\[
\begin{align*}
q & \leftarrow p \\
r & \leftarrow p \\
s & \leftarrow q \land r \\
p & \\
\end{align*}
\]

In LP syntax:

\[
\begin{align*}
q & : - p. \\
r & : - p. \\
s & : - q, r. \\
p & \\
\end{align*}
\]

A: \{p, q, r, s\}

Q: What is the “answer” (answer set) of the following?

In propositional logic:

\[
\begin{align*}
q & \leftarrow p \\
r & \leftarrow q \\
t & \leftarrow r \land s \\
p & \\
t & \\
\end{align*}
\]

In LP syntax:

\[
\begin{align*}
q & : - p. \\
r & : - q. \\
t & : - r, s. \\
p & \\
t & \\
\end{align*}
\]

A: \{p, t, q, r\}

... note that s isn’t in the answer