Logic Programming (PLP 11, CTM 9.2, 9.4, 12.1-12.2)

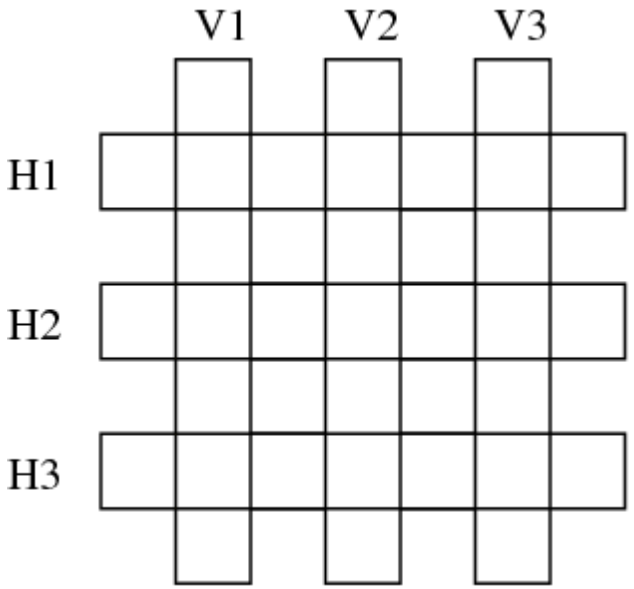
Constraint Satisfaction Problems, Natural Language Parsing (Definite Clause Grammars)

Carlos Varela
Rensselaer Polytechnic Institute

November 21, 2017

Constraint Satisfaction Example*

- Given six Italian words:
  - astante , astoria , baratto , cobalto , pistola , statale .
- They are to be arranged, crossword puzzle fashion, in the following grid:



Constraint Satisfaction Example(2)*

- The following knowledge base represents a lexicon containing these words:

```
word(astante,  a,s,t,a,n,t,e).
word(astoria,  a,s,t,o,r,i,a).
word(baratto,  b,a,r,a,t,t,o).
word(cobalto,  c,o,b,a,l,t,o).
word(pistola,  p,i,s,t,o,l,a).
word(statale,  s,t,a,t,a,l,e).
```

- Write a predicate `crossword/6` that tells us how to fill in the puzzle. The first three arguments should be the vertical words from left to right, and the last three arguments the horizontal words from top to bottom.

Constraint Satisfaction Example(3)*

- Try solving it yourself before looking at this solution!

```
crossword(V1,V2,V3,H1,H2,H3) :-
     word(V1,_,H1V1,_,H2V1,_,H3V1,_),
     word(V2,_,H1V2,_,H2V2,_,H3V2,_),
     word(V3,_,H1V3,_,H2V3,_,H3V3,_),
     word(H1,_,H1V1,_,H1V2,_,H1V3,_),
     word(H2,_,H2V1,_,H2V2,_,H2V3,_),
     word(H3,_,H3V1,_,H3V2,_,H3V3,_).
```

Generate and Test Example

- We can use the relational computation model to generate all digits:

```
fun {Digit}
   choice 0 [] 1 [] 2 [] 3 [] 4 [] 5 [] 6 [] 7 [] 8 [] 9 end
end
{Browse {Search.base.all Digit}}
% displays [0 1 2 3 4 5 6 7 8 9]
```

Finding digit pairs that add to 10

- Using generate and test to do combinatorial search:

```
fun {PairAdd10}
   D1 D2 in
   D1 = {Digit}          % generate
   D2 = {Digit}          % generate
   D1+D2 = 10            % test
   D1#D2
end
{Browse {Search.base.all PairAdd10}}
% displays [1#9 2#8 3#7 4#6 5#5 6#4 7#3 8#2 9#1]
```

Finding palindromes

- Find all four-digit palindromes that are products of two-digit numbers:

```
fun {Palindrome}
   X in
   X = (10*{Digit}+{Digit})*(10*{Digit}+{Digit})      % generate
   (X>=1000) = true                                   % test
   (X div 1000) mod 10 = (X div 1) mod 10             % test
   (X div 100) mod 10 = (X div 10) mod 10             % test
   X
end
{Browse {Search.base.all Palindrome}}                 % 118 solutions
```

Propagate and Search

- The *generate and test* programming pattern can be very inefficient (e.g., Palindrome program explores 10000 possibilities).

- An alternative is to use a *propagate and search* technique.

  Propagate and search filters possibilities during the generation process, to prevent combinatorial explosion when possible.

Propagate and Search

Propagate and search approach is based on three key ideas:

- *Keep partial information*, e.g., "in any solution, X is greater than 100".
- *Use local deduction*, e.g., combining "X is less than Y" and "X is greater than 100", we can deduce "Y is greater than 101" (assuming Y is an integer.)
- *Do controlled search*. When no more deductions can be done, then search. Divide original CSP problem P into two new problems: (P ^ C) and (P ^ ¬C) and where C is a new constraint. The solution to P is the union of the two new sub-problems. Choice of C can significantly affect search space.

Propagate and Search Example

- Find two digits that add to 10, multiply to more than 24:

```
D1::0#9          D2::0#9                  % initial constraints
{Browse D1}      {Browse D2}              % partial results
D1+D2 =: 10  % reduces search space from 100 to 81 possibilities
                 % D1 and D2 cannot be 0.
D1*D2 >=: 24   % reduces search space to 9 possibilities
                 % D1 and D2 must be between 4 and 6.
D1 <: D2         % reduces search space to 4 possibilities
                 % D1 must be 4 or 5 and D2 must be 5 or 6.
                 % It does not find unique solution D1=4 and D2=6
```

Propagate and Search Example(2)

- Find a rectangle whose perimeter is 20, whose area is greater than or equal to 24, and width less than height:

```
fun {Rectangle}
   W H in W::0#9   H::0#9
   W+H =: 10
   W*H >=: 24
   W <: H
   {FD.distribute naive rect(W H)}
   rect(W H)
end
{Browse {Search.base.all Rectangle}}
% displays [rect(4 6)]
```

Finding palindromes (revisited)

- Find all four-digit palindromes that are products of two-digit numbers:

```
fun {Palindrome}
   A B C X Y in
   A::1000#9999   B::0#99 C::0#99
   A =: B*C
   X::1#9   Y::0#9
   A =: X*1000+Y*100+Y*10+X
   {FD.distribute ff [X Y]}
   A
end
{Browse {Search.base.all Palindrome}}              % 36 solutions
```

Natural Language Parsing

(Example from "Learn Prolog Now!" Online Tutorial)

```
word(article,a).
word(article,every).
word(noun,criminal).
word(noun,'big kahuna burger').
word(verb,eats).
word(verb,likes).

sentence(Word1,Word2,Word3,Word4,Word5) :-
        word(article,Word1),
        word(noun,Word2),
        word(verb,Word3),
        word(article,Word4),
        word(noun,Word5).
```

Parsing natural language

- *Definite Clause Grammars (DCG)* are useful for natural language parsing.

- Prolog can load DCG rules and convert them automatically to Prolog parsing rules.

DCG Syntax

`-->`

DCG *operator*, e.g.,

```
sentence-->subject,verb,object.
```

Each goal is assumed to refer to the *head* of a DCG rule.

`{prolog_code}`

Include Prolog code in generated parser, e.g.,

```
subject-->modifier,noun,{write('subject')}.
```

`[terminal_symbol]`

Terminal symbols of the grammar, e.g.,

```
noun-->[cat].
```

Natural Language Parsing

(example rewritten using DCG)

```
sentence --> article, noun, verb, article, noun.

article --> [a] | [every].

noun --> [criminal] | ['big kahuna burger'].

verb --> [eats] | [likes].
```

Natural Language Parsing (2)

```
sentence(V) --> subject, verb(V), subject.

subject --> article, noun.

article --> [a] | [every].

noun --> [criminal] | ['big kahuna burger'].

verb(eats) --> [eats].
verb(likes) --> [likes].
```

Exercises

83. How would you translate DCG rules into Prolog rules?
84. PLP Exercise 11.8 (pg 571).
85. PLP Exercise 11.14 (pg 572).
86. CTM Exercise 12.6.2 (pg 774).