A generate-and-test procedure has two parts:

1. A **generator** which can generate a number of possible solutions.
2. A **tester** which succeeds iff the generated result is an acceptable solution.

When the tester fails, the generator will backtrack and generate a new possible solution.

### Generate & Test – Integer Division

We can define integer arithmetic (inefficiently) in Prolog:

```prolog
% Integer generator.
is_int(0).
is_int(X) :- is_int(Y), X is Y+1.

% Result = N1 / N2.
divide(N1, N2, Result) :-
is_int(Result),
P1 is Result*N2,
P2 is (Result+1)*N2,
P1 =< N1, P2 > N1, !.
```

```prolog
| ?- divide(6,2,R).  
  R = 3 |
```
is_int(0).

is_int(X) :- is_int(Y), X is Y+1.

divide(N1, N2, Result) :-
    is_int(Result),
    P1 is Result*N2, P2 is (Result+1)*N2,
    P1 =< N1, P2 > N1, !.

<table>
<thead>
<tr>
<th>divide(6,2,R) --- N1=6, N2=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Res</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

This is a part of a program to play Tic-Tac-Toe (Naughts and Crosses).

Two players take turns to put down X and O on a 3x3 board. Whoever gets a line of 3 (horizontal, vertical, or diagonal) markers has won.

We'll look at the predicate forced_move which answers the question:

- Am I (the naught-person) forced to put a marker at a particular position?

The program tries to find a line with two crosses.

It only makes sense to find one forced move, hence the cut.
aline(L) is a generator – it generates all possible lines(L).
threatening(L,B,Sq) is a tester – it succeeds if Sq is a threatened square in line L of board B.

forced_move(Board, Sq) :-
  aline(Line),
  threatening(Line, Board, Sq), !.

?- forced_move(b(x,x,o,x,o,x,x),4). yes

A square is empty if it is an uninstantiated variable.
arg(N,S,V) returns the N:th element of a structure S.

empty(Sq, Board) :-
  arg(Sq,Board,Val), var(Val).
cross(Sq, Board) :-
  arg(Sq,Board,Val), nonvar(Val), Val=x.
nought(Sq, Board) :-
  arg(Sq,Board,Val), nonvar(Val), Val=o.
Generate & Test – Arbitrage

From the Online Webster’s: 

**arbitrage**

*simultaneous purchase and sale of the same or equivalent security in order to profit from price discrepancies*

?- arbitrage.

dollar dmark yen 1.03751
yen dollar dmark 1.03751
dmark yen dollar 1.03751

exchange(pound, dollar, london, 1.550).
exchange(pound, dollar, new_york, 1.555).
exchange(pound, dollar, tokyo, 1.559).
exchange(pound, yen, london, 153.97).
exchange(pound, yen, new_york, 154.05).
exchange(pound, yen, tokyo, 154.3).
exchange(pound, dmark, london, 2.4075).
exchange(pound, dmark, new_york, 2.44).
exchange(pound, dmark, tokyo, 2.408).
exchange(dollar, yen, london, 98.3).
exchange(dollar, yen, new_york, 98.35).
exchange(dollar, yen, tokyo, 98.25).
exchange(dollar, dmark, london, 1.537).
exchange(dollar, dmark, new_york, 1.58).
exchange(dollar, dmark, tokyo, 1.57).
exchange(yen, dmark, london, 0.015635).
exchange(yen, dmark, new_york, 0.0155).
exchange(yen, dmark, tokyo, 0.0158).

Generate & Test – Arbitrage...

arbitrage :-
profit3(From, Via, To, Profit), % Gen
Profit > 1.03, % Test
write(From), write(' '), write(Via), write(' '), write(To), write(' '),
write(Profit), nl, fail.

arbitrage.

% Find three currencies, and the profit:
profit3(From, Via, To, Profit) :-
best_rate(From, Via, P1, R1),
best_rate(Via, To, P2, R2),
best_rate(To, From, P3, R3),
Profit is R1 * R2 * R3.

Generate & Test – Arbitrage...

% We can convert back and forth
% between currencies:
rate(From, To, P, R) :-
exchange(From, To, P, R).
rate(From, To, P, R) :-
exchange(To, From, P, S), R is 1/S.

% Find the best place to convert
% between currencies From & To:
best_rate(From, To, Place, Rate) :-
rate(From, To, Place, Rate),
not((rate(From, To, P1, R1), R1>Rate)).
Stable Marriages

Suppose there are $N$ men and $N$ women who want to get married to each other.

Each man (woman) has a list of all the women (men) in his (her) preferred order. The problem is to find a set of marriages that is stable.

A set of marriages is unstable if two people who are not married both prefer each other to their spouses. If $A$ and $B$ are men and $X$ and $Y$ women, the pair of marriages $A-Y$ and $B-X$ is unstable if

- $A$ prefers $X$ to $Y$, and
- $X$ prefers $A$ to $B$.

Write a program which takes a set of people and their preferences as input, and produces a set of stable marriages as output.

Input Format: _______________
prefer(avraham, man,
[chana,tamar,zvia,ruth,sarah]).

men([avraham,binyamin,chai, david,elazar]).

women([zvia, chana, ruth, sarah, tamar]).

The first rule, says that avraham is a man and that he prefers chana to tamar, tamar to zvia, zvia to ruth, and ruth to sarah.
Stable Marriages... 

• gen generates all possible sets of marriages, unstable tests if they are stable.

go :-
   men(ML), women(WL),
   gen(ML, WL, [], L),
   unstable(L),
   show(L), fail.

go.

?- men(ML), women(WL), gen(ML,WL, [],L).
L = [m(elazar,tamar),m(david,sarah),
m(chaim,ruth),m(binyamin,chana),
m(avraham,zvia)] ? ;

......

Stable Marriages — Generate

```prolog
prefer(avraham, man, [chana, tamar, zvia, ruth, sarah]).
prefer(binyamin, man, [zvia, chana, ruth, sarah, tamar]).
prefer(chaim, man, [chana, ruth, tamar, sarah, zvia]).
prefer(david, man, [zvia, ruth, chana, sarah, tamar]).
prefer(elazar, man, [tamar, ruth, chana, zvia, sarah]).
prefer(zvia, woman, [elazar, avraham, david, binyamin, chaim]).
prefer(chana, woman, [david, elazar, binyamin, avraham, chaim]).
prefer(ruth, woman, [avraham, david, binyamin, chaim, elazar]).
prefer(sarah, woman, [chaim, binyamin, david, avraham, elazar]).
prefer(tamar, woman, [david, binyamin, chaim, elazar, avraham]).
```

Stable Marriages — Test

```prolog
\% A prefers B to C.
pref(A, B, C) :-
   prefer(A, _, L),
   append(_, [B|S], L), !,
   member(C, S), !.

\% unstables
unstable(L) :-
   append(_, [A|R], L),
   member(B, R),
   (is_unstable(A,B);
    is_unstable(B,A)).

\% is_unstable
is_unstable(m(A,Y), m(B,X)) :-
   pref(A, X, Y),
   pref(X, A, B).
```

Stable Marriages — Generate

```prolog
gen([A|M1], W, In, Out) :-
   delete(B, W, W1),
   gen(M1, W1, [m(A,B)|In], Out).

gen([],[],L,L).

delete(A, [A|L], L).
delete(A, [X|L], [X|L1]) :-
   delete(A, L, L1).
```
“Helder, a poor scientist, was in love with the daughter of an admiral. One day, a general captured the girl. Helder rode to the general’s barrack and killed the general. The girl was grateful and fell in love with Helder. The admiral was so happy to have his daughter back he gave Helder half of all his boats.”

- Who is the father of the girl?
- Who is rich?
- Who loves who?
- Who is poor?
- Who captured who?
- Who killed who?

:- op(500, xfy, 'is_').
:- op(500, yfx, 'loves').
:- op(500, yfx, 'kills').
:- op(500, yfx, 'to').
:- op(500, yfx, 'captures').
:- op(500, yfx, 'rides_to').
:- op(500, yfx, 'gives').
:- op(500, yfx, 'is_father_of').
:- op(800, yfx, 'and').

X and Y :- X, Y.
helder is poor.
helder is scientist.
admiral is happy.
admiral is father of girl.
helder loves girl.
girl loves helder.
general captures girl.
helder kills general.
admiral gives half boats to helder.

\% Who loves who?
?- Z loves Y, write(Z), write(' loves '), write(Y), nl, fail.
    helder loves girl
    girl loves helder

\% Who captures who?
?- Z captures Y.
    Z = general
    Y = girl

\% Who kills who?
?- Z kills Y.
    Z = helder
    Y = general

\% Who loves who’s daughter?
?- Z loves G and F is father of G.
    Z = helder
    G = girl
    F = admiral
The Crewes, Dews, Grandes, and Lands of Bower Street each have a front-yard tree: Catalpa, Dogwood, Gingko, Larch.

- The Grandes’ tree and the Catalpa are on the same side of the street.
- The Crewes live across the street from the Larch.
- The Larch is across the street from the Dews’ house.
- No tree starts with the same letter as its owner’s name.
- Who owns which tree?

% Let’s assume that the Larch is on the % north side of the street.
northside('Larch').

% The Crewes live across the street from % the Larch. The Larch is across the % street from the Dews’ house.
southside('Crewes').
southside('Dews').

% The Grandes’ tree and the ’Catalpa’ % are on the same side of the street.
northside('Catalpa') :-
    northside('Grandes').

Grandes owns the Larch
Crewes owns the Dogwood
Dews owns the Ginko
Lands owns the Catalpa
% If Grandes have a 'Larch', then they must live on the north side.
northside('Grandes') :-  
    have('Grandes', 'Larch').

% Grandes have a 'Larch', if noone else does.
have('Grandes','Larch') :-
    not own('Crewes','Larch'),
    not own('Dews','Larch'),
    not own('Lands','Larch')

% then the Dews' and Crews' will be on the south side. Also, if the Catalpa is on the north the Dogwood and Ginko must both be on the south side (since each house has one tree).
southside('Dogwood') :-
    northside('Larch'),
    northside('Catalpa').
southside('Ginko') :-
    northside('Larch'),
    northside('Catalpa').

% Are you a tree or a plant?
person(X) :- member(X, ['Grandes','Crewes','Dews','Lands']).
tree(X) :- member(X, ['Catalpa','Ginko','Dogwood','Larch']).

% No tree starts with the same letter as its owner's name.
not_own(X,Y) :-
    name(X, [A_|_]), name(Y, [A_|_]).

% The Grandes' tree and the 'Catalpa' are on the same side of the street.
not_own('Grandes','Catalpa').

% Only a person can own a tree.
not_own(X,Y) :- person(X), person(Y).
not_own(X,Y) :- tree(X), tree(Y).

% A person can only own a tree that's on the same side of the street as themselves.
not_own(X,Y) :- northside(X),southside(Y).
not_own(X,Y) :- southside(X),northside(Y).
% You can’t own what someone else owns.
not_own('Crewes', X) :- owns('Dews', X).
not_own('Lands', X) :- owns('Crewes', X).
not_own('Lands', X) :- owns('Dews', X).

owns(X, Y) :-
    person(X), tree(Y),
    not(not_owns(X, Y)).

solve :-
    owns(Person, Tree),
    write(Person), write(' owns the '), write(Tree), nl, fail.
solve.

Logic Arithmetic

% Represent S as the sum of 2 numbers.
% minus(S, D1, D2) -- S – D1 = D2
minus(S, S, 0).
minus(S, D1, D2) :- % Note that
    S > 0, S1 is S-1, % S must be
    minus(S1, D1, D3), % instantiated.
    D2 is D3 + 1.

?- minus(3, X, Y).
X = 3, Y = 0 ;
X = 2, Y = 1 ;
X = 1, Y = 2 ;
X = 0, Y = 3

Arithmetic in Logic

- Arithmetic in Prolog is just like arithmetic in imperative languages. We can’t do \[25 = X + Y\] and hope to get \(X\) and \(Y\) instantiated to every pair of numbers that sum to 25.
- There are cases when we need the power of logic arithmetic, rather than the efficient built-in operators. That is no problem, we can always define the logic arithmetic predicates ourselves.
- For example, how do we split a number into the two parts. Note that this is similar to splitting a list using append.

Arithmetic In Logic...

- We can always write our own logic arithmetic predicates.
The minus predicate splits \( S \) into \( D_1 + D_2 \). Why does it work? Well, look at this:

\[
\begin{align*}
  S_1 &= S - 1 \quad \text{first line} \\
  D_3 &= S_1 - D_1 \quad \text{second line} \\
  D_2 &= D_3 + 1 \quad \text{third line} \\
  S &= S_1 + 1 \\
     &= (D_3 + D_1) + 1 \\
     &= ((D_2 - 1) + D_1) + 1 \\
     &= D_2 + D_1
\end{align*}
\]

Note that the minus predicate require the first argument to be instantiated, but not the second and third. minus, below, is a lot like append.

Pythagorean Triples

?- pythag(X, Y, Z).
  X = 4, Y = 3, Z = 5 ;
  X = 3, Y = 4, Z = 5 ;
  X = 8, Y = 6, Z = 10 ;
  X = 6, Y = 8, Z = 10 ;
  X = 12, Y = 5, Z = 13 ;
  X = 5, Y = 12, Z = 13 ;
  X = 12, Y = 9, Z = 15

*is_int* is used to generate a sequence of numbers.

*int_triple* splits the generated integer \( S \) into the sum of three integer \( X, Y, Z \).

In other words, first we check all triples that sum to 1 to see if any of them are pythagorean triples, then all triples that sum to 2, etc. This obviously will eventually check “all” triples. It also will make sure that we get them “in order”, with the smallest triples first.
Exercise: Crossword Puzzle

Across
1. The Fifth Element.
2. Mumintroll mum.

Exercise: Crossword Puzzle

Now, assume that you have a much bigger database of words.

How would you organize the database for much faster searching?

How would you rewrite your code to make use of the new database structure?