An Introduction to PROLOG

- Syntax for Predicate Calculus
- Abstract Data Types (ADTs) in PROLOG
- A Production System Example in PROLOG

Represent Facts and Rules

- Facts (propositions):
  - e.g. male(philip).
- Rules (implications):
- e.g.

  parent(X, Y) := father(X, Y).

  parent(X, Y) := mother(X, Y).

  parent(X, Y) := father(X, Y); mother(X, Y).
- Questions/Queries (start point of execution):
  - e.g.   ?-parent(X, Y).

Prolog as Logic

- Horn clause – a subset of first-order logic
- Logic formula:
  - Description – A – e.g. philip is male.
  - Logic OR – $A \vee B$: **A; B.**
  - Logic AND – $A \wedge B$: **A, B.**
  - Logic NOT -- $\neg A$: **not A.**
  - Logic implication: $A \rightarrow B$: **B :- A.**
- Each expression terminates with a .
- Deductive inference – Modus ponens

A

$A \rightarrow B$

B

A Simple Rule

English description:

If
X is male,
F is the father of X,
M is the mother of X,
F is the father of Y,
M is the mother of Y
Then
X is a brother of Y

Logic formula:

$$
\begin{aligned}
& male(X) \wedge \\
& father(F, X) \wedge \\
& mother(M, X) \wedge \\
& father(F, Y) \wedge \\
& mother(M, Y) \\
\rightarrow & \\
& brother(X, Y)
\end{aligned}
$$

Prolog:

```
brother(X, Y) :-
        male(X),
        father(F,X),
        mother(M,X),
        father(F,Y),
        mother(M, Y).
```

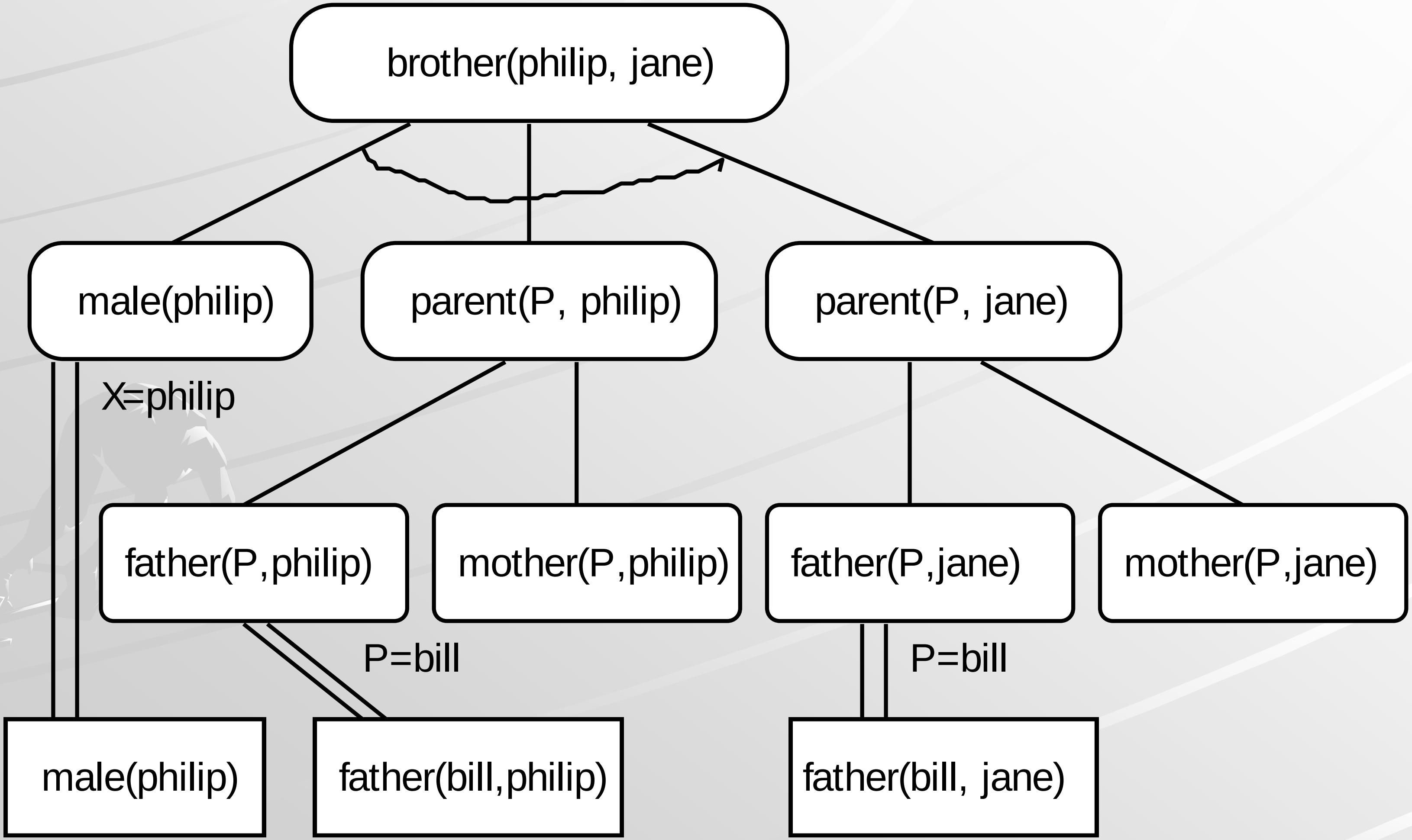
A Simple Program

A Prolog program is a sequence of facts and rules

```
brother(X, Y) :- male(X), parent(P, X), parent(P, Y).
parent(X, Y):- father(X, Y); mother(X, Y).
male(philip).
father(bill, philip).
father(bill,jane).

?- brother(philip, jane).
```

Program Trace Tree



Program Execution

- Start from the question
- Matching – match the question with facts and rules by substitution (try)
- Unification – looking for unified solution (consistent solution)
- Backtracking – once fails, go back to try another case
- Divide-and-conquer
  - divide problem into sub problems
  - solve all sub problems
  - integrate sub solutions to build the final solution
  - Solutions to all sub problems must be consistent

Another Example

- Facts

  mother(jane, george).
  father(john, george).
  Brother(bill, john).

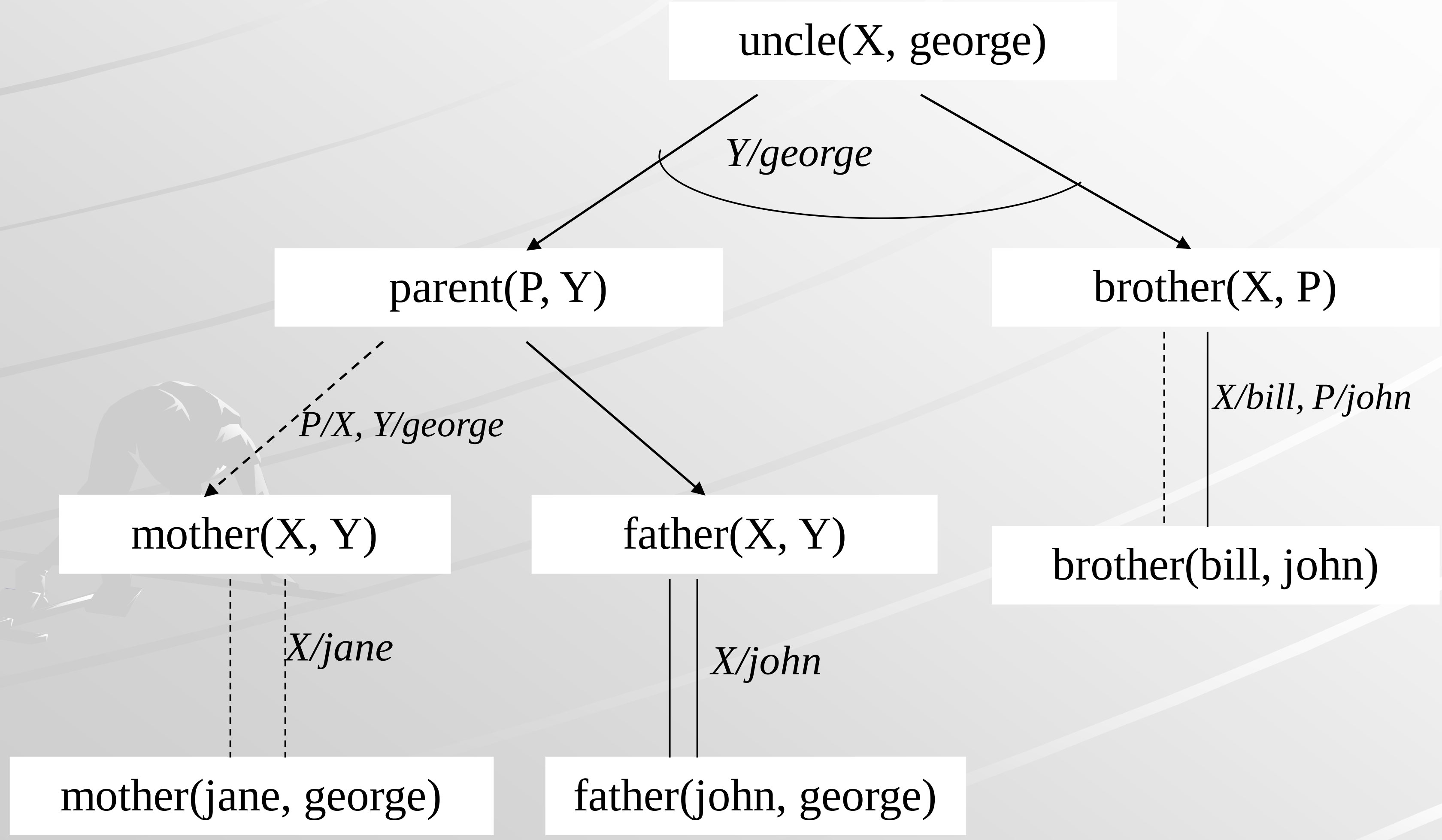
- Rules:

  parent(X, Y) :- mother(X, Y).
  parent(X, Y) :- father(X, Y).
  uncle(X, Y) :- parent(P, Y), brother(X, P).

- Question:

  ?- uncle(X, george).

Program Execution Trace



Unification and Backtracking

- First matching (dashed line):
  - parent(jane, george) -- P ← jane
  - brother(bill, john) -- P ← john

  which is not consistent.
- Backtracking (solid line):
  - parent(john, george) -- P ← john
  - Brother(bill, john) -- P ← john

  which is unified
- Multiple solutions
  - Interactive program
  - Interpreter
  - Once the program outputs, the user can respond with ; to ask for more solutions
  - If no more solutions, the program answers “no”

Prolog Environment

- Add new predicates
  - assert(P) – add new predicate P to database
  - asserta(P) – add P to the beginning of database
  - assertz(P) – add P to the end of database
- Delete predicates
  - retract(P) – remove P fromdatabase
- Import database from a file
  - consult(File)
- Input/output
  - read(X) – read a term from the input stream to X
  - write(X) – put the term X in the output stream
  - see(File) – open a file for reading (input stream)
  - tell(File) – open a file for writing (output stream)

Prolog Program Execution Tracing

- List all clauses with the predicate name
  - listing(P) – regardless of the number of arguments
- Monitor the program progress

trace	print every goal
exit	when a goal is satisfied
retry	if more matches
fail	enforce fail
spy	print all uses of predicates
notrace	stop the trace

Recursion

- Base case – a set of rules for direct solution
- Normal case – another set of rules for indirection solution
- E.g. Fibnacci number:

  fib(0) = 0, fib(1) = 1, → base case

  fib(n) = fib(n-1) + fib(n-2) → normal case
- Prolog program:

```
Facts:       fib(0, 0).
             fib(1, 1).
Rules:       fib(X, N) :- N > 1, N1 is N - 1, N2 is N – 2,
                  fib(Y, N1), fib(Z, N2), X is Y + Z.
Question:    ? :- fib(X, 5).
      X = 5.
      ? :- fib(5, N).
      N = 5.
      ? :- fib(X, 8).
      X = 21.
```

Data Representation

- Atom – numbers, strings, etc.
- Structured data – pattern
  - Format: functor(parameter-list) where parameters in the list are either atom or structured data separated with “,”
  - E.g. person(name)
    student(person(name), age)
    male(bill)
    female(jane)
  - Special function dot (.)
    - Represents list:.(p, .pattern)
    - Special .pattern: .() ➔ [], empty list
    - E.g.: .(p, .(q, .(r, .(s, .())))))
    - List: [p, q, r, s]
    - Operations: [X|Y] ➔ pattern
  - Anonyous variable: underscore (_)
  - E.g.

```
member(X, [X|_]) :- !.
member(X, [_|Y]) :- member(X, Y).
```

Lists

- A sequence of elements separated by comma and enclosed by [and]
- List elements can be other lists
- Examples: [1, 2, 3, 4]
  [a, george, [b, jack], 3]
- Empty list: []
- Non-empty lists consists of two parts
  - Header – the first element
  - Tail – the list containing all elements except the first
  - Can be written [Header|Tail]
- Example: for list [a, 1, b, 2]
  - Header: a
  - Tail: [1, b, 2]
- Match:
  - headers match and tails match
  - Match [a, 1, b, 2] with [H|T]
    - H = a
    - T = [1, b, 2]

Count the number of elements

- Count the number of elements in a list
  - Base case: if the list is empty, then number of elements is 0
  - Normal case: the number of elements in the list is the number of elements in the tail plus 1
  - Rules?

Count the number of elements

- Find the number of elements in a list

```
length:-length([], 0) :- !.
length([X|Y], N) :- length(Y, M), N is M+1.
```

Recursive Search

- Depth-first search with backtracking
- Ordering of facts and rules affect the problem-solving efficiency
- Search a subgoal:
  - Top → down: matching facts and heads of rules
- Decomposition of goals:
  - Left → right
- Backtracking:
  - Matching: Top → down
  - Subgoals: right → left

Abstract Data Types in Prolog

- ADT
  - A set of operations
  - No data structure specified
- ADT building components
  - Recursion, list, and pattern matching
  - List handing and recursive processing are hidden in ADTs
- Typical ADTs
  - Stack, Queue, Set, Priority Queue

The ADT Stack

- Characteristics
  - LIFO (Last-in-first-out)
- Operations
  - Empty test
  - Push an element onto the stack
  - Pop the top element from the stack
  - Peek to see the top element
  - Member_stack to test members
  - Add_list to add a list of elements to the Stack

The ADT Queue

- Characteristics
  - FIFO (First-in-first-out)
- Operations
  - Empty test
  - Enqueue – add an element to the end
  - Dequeue – remove the first element
  - Peek – see the first element
  - Member test
  - Merge two queues

The ADT Priority Queue

- Characteristics
  - Each element is associated with a priority
  - Always remove the “smallest” one
- Operations
  - Empty test
  - Member test
  - Add an element to the priority queue
  - Min – find the minimum element
  - Remove the minimum element
  - Add a list to a priority queue

The ADT Set

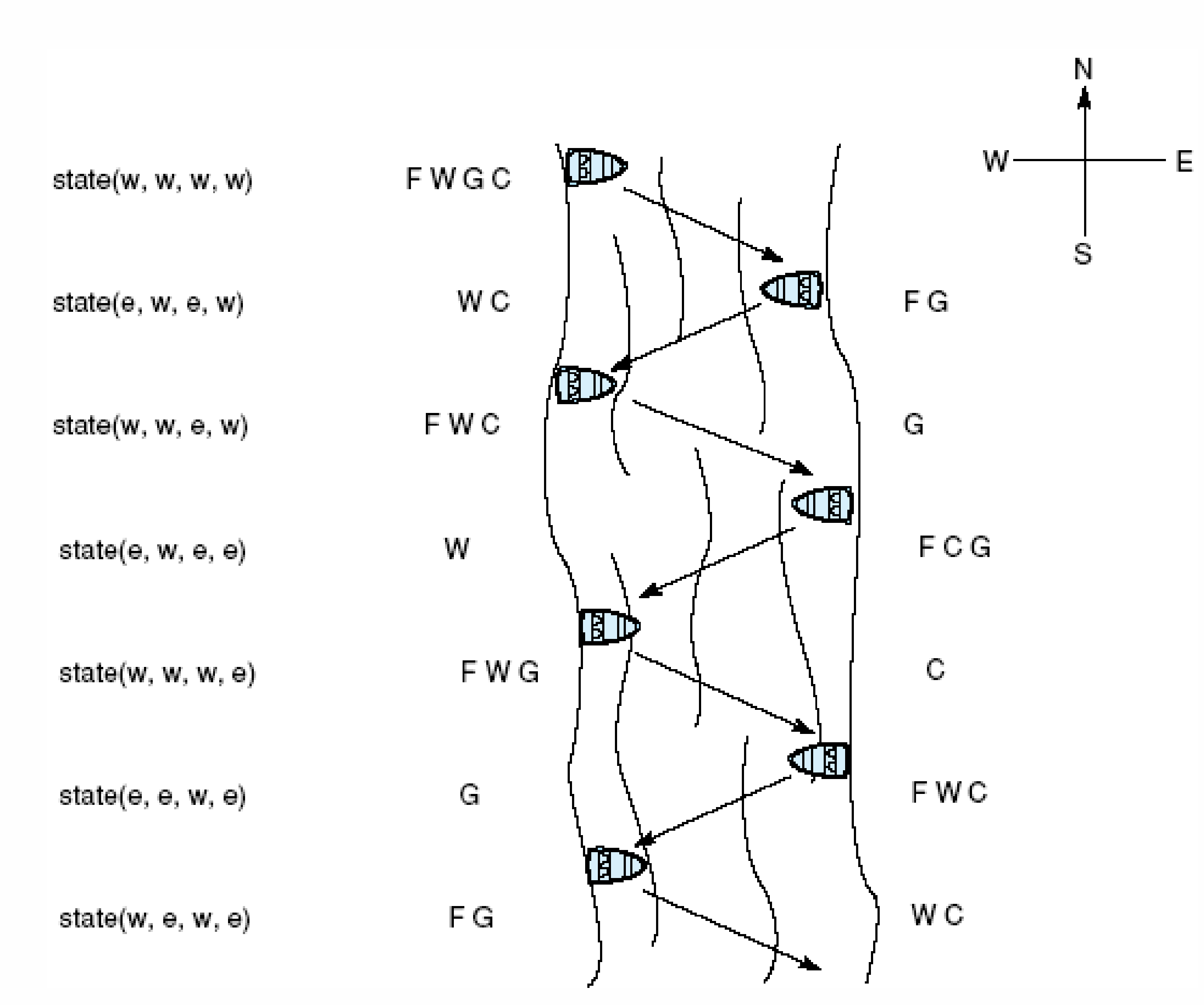
- A collection of elements
  - No order
  - No duplicates
- Operations
  - Empty test
  - Member test
  - Add an element to the set
  - Remove an element from the set
  - Union two sets
  - Intersect two sets
  - Difference two sets
  - Subset test
  - Equality test

A Production System in Prolog

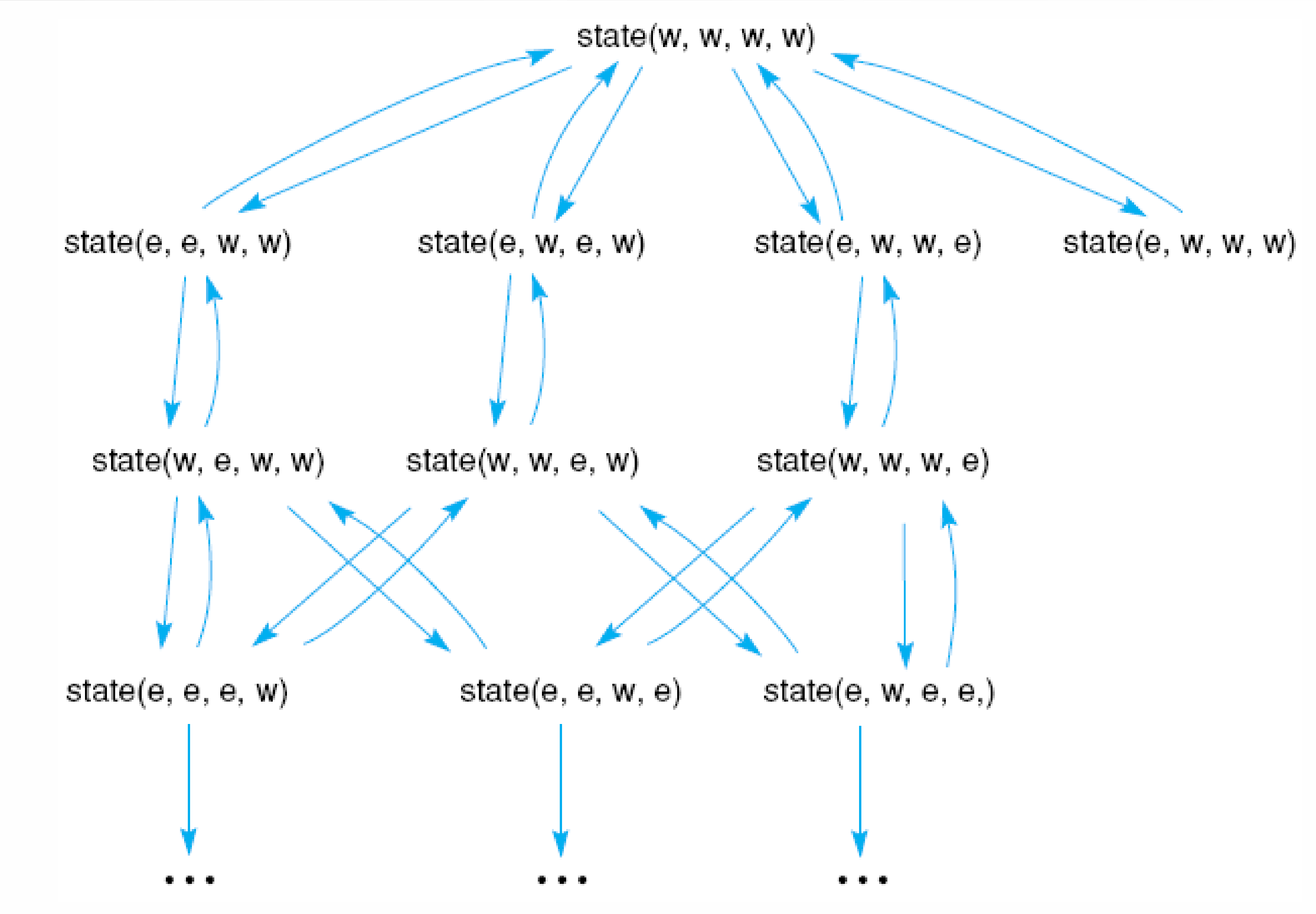
- Farmer, wolf, goat, and cabbage problem
  - A farmer with his wolf, goat, and cabbage come to the edge of a river they wish to cross. There is a boat at the river's edge, but, of course, only the farmer can row. The boat also can carry only two things, including the rower, at a time. If the wolf is ever left alone with the goat, the wolf will eat the goat; similarly, if the goat is left alone with the cabbage, the goat will eat the cabbage. Devise a sequence of crossings of the river so that all four characters arrive safely on the other side of the river.
- Representation
  - state(F, W, G, C) describes the location of Farmer, Wolf, Goat, and Cabbage
  - Possible locations are e for east bank, w for west bank
  - Initial state is state(w, w, w, w)
  - Goal state is state(e, e, e, e)
  - Predicates opp(X, Y) indicates that X and y are opposite sides of the river
  - Facts:

```
opp(e, w).
opp(w, e).
```

Sample crossings for the farmer, wolf, goat, and cabbage problem.



Portion of the state space graph of the farmer, wolf, goat, and cabbage problem, including unsafe states.



Production Rules in Prolog

- Unsafe states

```
unsafe(state(X, Y, Y, C)) :- opp(X, Y).
unsafe(state(X, W, Y, Y)) :- opp(X, Y).
```

- Move rules

```
move(state(X, X, G, C), state(Y, Y, G, C))) :- opp(X, Y),
                not(unsafe(state(Y, Y, G, C))),
                writelist([‘farms takes wolf’, Y, Y, G, C]).
move(state(X, W, X, C), state(Y, W, Y, C)) :- opp(X, Y),
                not(unsafe(state(Y, W, Y, C))),
                writelist([‘farmers takes goat’, Y, W, Y,C]).
move(state(X, W, G, X), state(Y, W, G, Y)) :- opp(X, Y),
                not(unsafe(state(Y, W, G, Y))),
                writelist(‘farmer takes cabbage’, Y, W, G, Y]).
move(state(X, W, G, C), state(Y, W, G, C)) :-opp(X, Y),
                not(unsafe(state(Y, W, G, C))),
                writelist([‘farmer takes self’, Y, W, G, C]).
move(state(F, W, G, C), state(F, W, G, C)) :-
                writelist([‘Backtrack from ‘, F, W, G, C]), fail.
```

Production Rules in Prolog

- Path rules

```
path(Goal, Goal, Stack) :-
        write('Solution Path Is: '), nl,
        reverse_print_stack(Stack).
path(State, Goal, Stack) :-
        move(State, Next),
        not(member_stack(Next, Stack)),
        stack(Next, Stack, NewStack),
        path(Next, Goal, NewStack), !.
```

- Start rule

```
go(Start, Goal) :-
        empty_stack(EmptyStack),
        stack(Start, EmptyStack, Stack),
        path(Start, Goal, Stack).
```

- Question

```
?- go(state(w, w, w, w), state(e, e, e, e)
```