Built-in Predicates

`print(X, Y, Z, ...)` prints its arguments, always succeeds

`Z is X+Y` succeeds if `X` and `Y` are bound to integer values, and sets `Z` to `X+Y`

`Z is X- Y` succeeds if `X` and `Y` are bound to integer values, and sets `Z` to `X-Y`

- `X` and `Y` must be integers (prevents infinite backtrack)
- also supports `*` and `/`

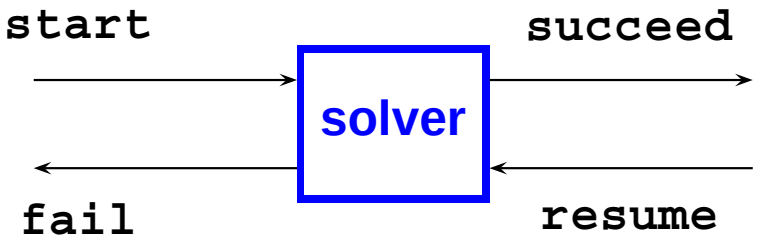
`X<Y` succeeds if `X` and `Y` are bound to integer values, and `X<Y`

- `X` and `Y` must be integers (prevents infinite backtrack)
- also supports `>`, `=<`, and `>=`

. . .and in real Prolog, many more. . .

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Recall: Continuations for search problems (from chapter 3)



`start`	Begin with partial solution
`fail`	Partial solution won't work
`succeed`	Pass the improved solution to the next step
`resume`	If the improved solution won't work, backtrack and try something else

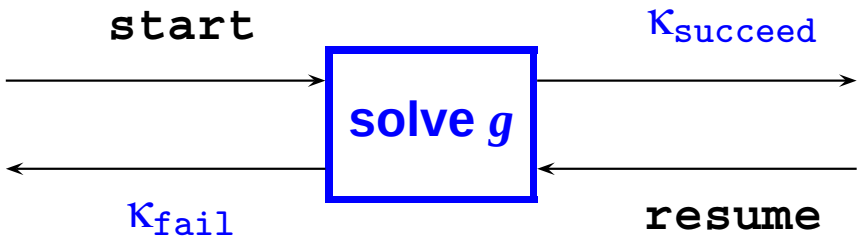
This is a composable unit, called a "Byrd box"

- Excellent conceptual model
- A very simple implementation (but not always the most efficient)

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Continuations (continued)

CPS (continuation-passing style) using the Byrd box



Each Byrd box tries every clause in sequence

- tries first clauses upon start
- may find a good clause and succeed using continuation $\kappa_{\texttt{succeed}}$
- tries next clause upon resume after backtracking
- may try all clauses and fail using continuation $\kappa_{\texttt{fail}}$

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog search

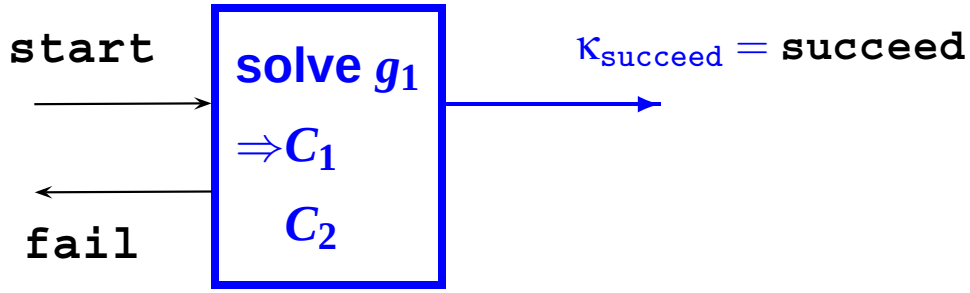
Recall the definition of the member predicate:

```
member(H,[H|T]).                    ; clause C1
member(X,[H|T]) :- member(X,T).     ; clause C2
```

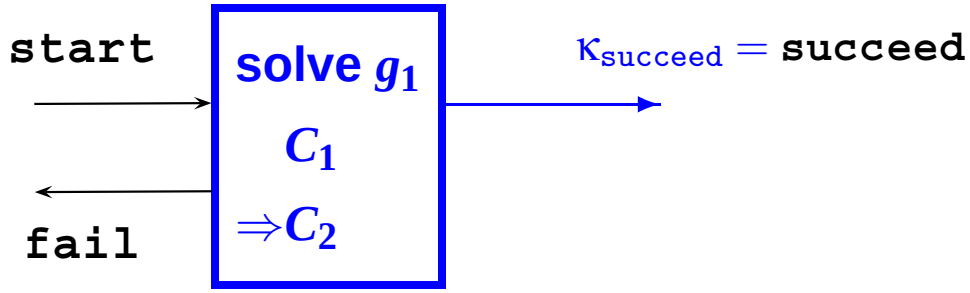
Consider this query (goal g_1):

```
?- member(b,[a,b]).
```

Create a Byrd box for g_1 that is prepared to consider clauses C_1 and C_2.



The goal g_1 does not unify with C_1, so now change state and look at C_2.



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog search (continued)

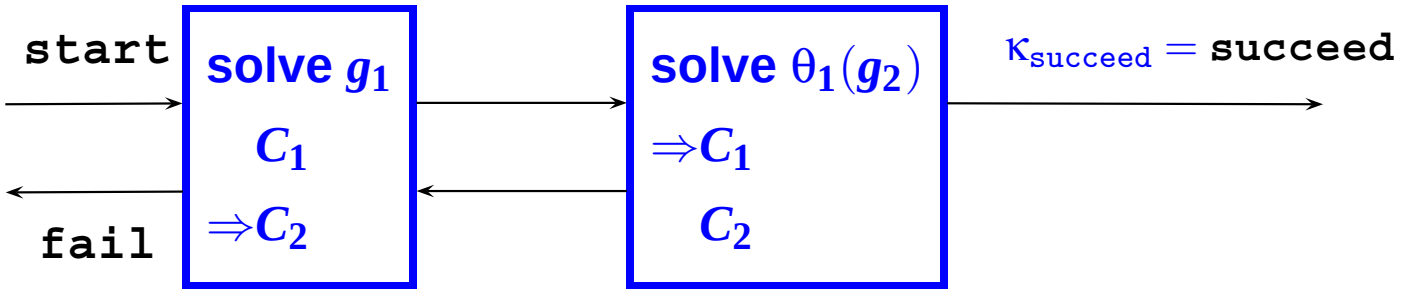
The goal g_1 unifies with clause C_2

```
member(X,[H|T]) :- member(X,T).
```

via substitution $\theta_1 = \{X_1 \mapsto b, H_1 \mapsto a, T_1 \mapsto [b]\}$.

Apply substitution θ_1 to clause C_2 to produce a new subgoal $\theta_1(g_2)$, which is

```
member(b,[b]).
```



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog search (continued)

The goal $\theta_1(g_2)$ unifies with clause C_1

```
member(H,[H|T]).
```

via substitution $\theta_2 = \{H_2 \mapsto b, T_2 \mapsto [\,]\}$.

But C_1 has no subgoals, so goals $\theta_1(g_2)$ and g_1 both now succeed.

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking

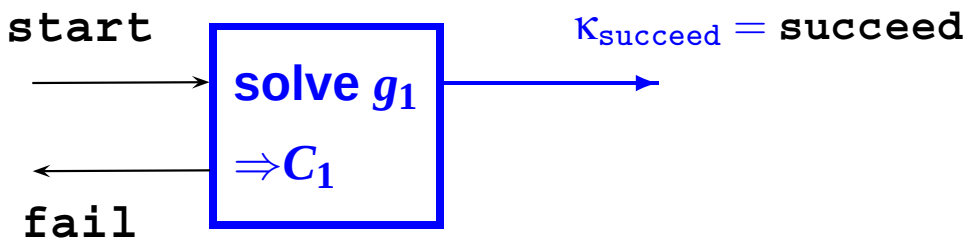
Place these clauses in the database:

```
p(f(Y)) :- q(Y), r(Y).     ; clause C1
q(s(Z)).                   ; clause C2
q(t(Z)).                   ; clause C3
r(t(a)).                   ; clause C4
```

Then ask this query (goal g_1):

```
?- p(X).
```

Create a Byrd box for g_1 that is prepared to consider clause C_1.



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking (continued)

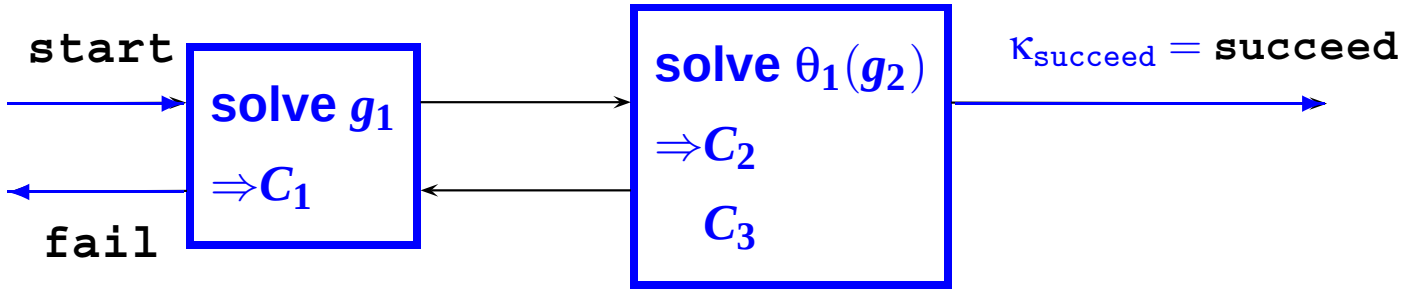
The goal g_1 unifies with clause C_1

```
p(f(Y)) :- q(Y), r(Y).
```

via substitution $\theta_1 = \{X \mapsto f(Y_1)\}$.

Apply substitution θ_1 to clause C_1 to produce new subgoal $\theta_1(g_2)$, which is `q(Y)`.

Create a Byrd box for $\theta_1(g_2)$ that is prepared to consider clauses C_2 and C_3.



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking (continued)

The goal $\theta_1(g_2)$ unifies with clause C_2

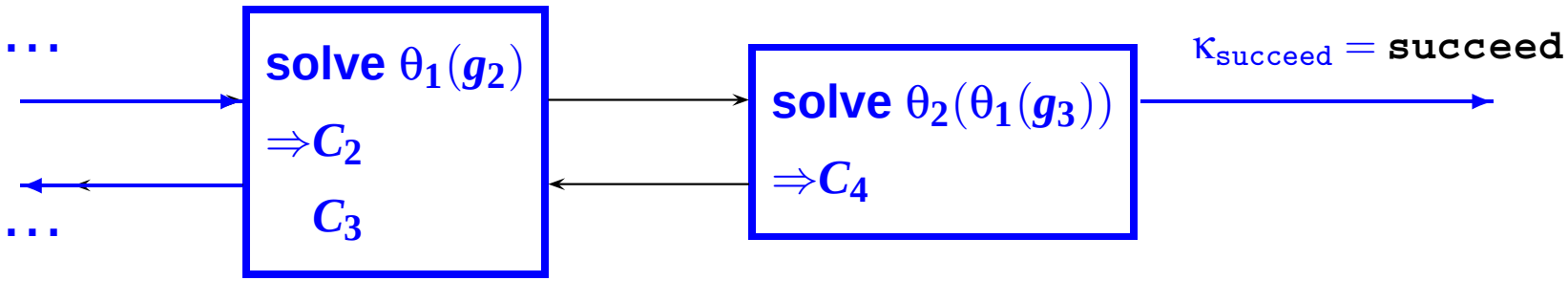
```
q(s(Z)).
```

via substitution $\theta_2 = \{Y_1 \mapsto s(Z_2)\}$.

But C_2 has no subgoals, so goal $\theta_1(g_2)$ immediately succeeds.

Next apply substitutions θ_2 and θ_1 to clause C_1 to produce next subgoal $\theta_2(\theta_1(g_3))$, which is `r(s(Z))`.

Create a Byrd box for $\theta_2(\theta_1(g_3))$ that is prepared to consider clause C_4.



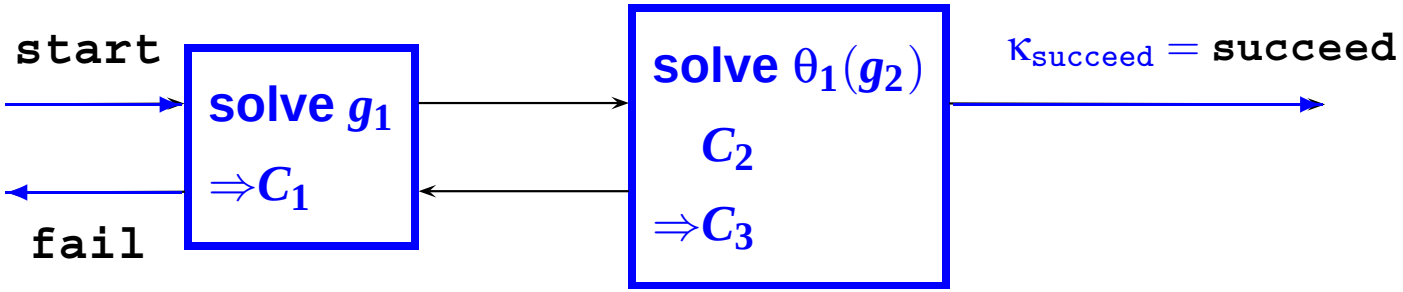
©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking (continued)

The goal $\theta_2(\theta_1(g_3))$ cannot unify with clause C_4, because the functor `s` is different from the functor `t`.

So we have failure and backtracking.

Resume to previous Byrd box, change state, and look at C_3.



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking (continued)

Now the goal $\theta_1(g_2)$ unifies with clause C_3

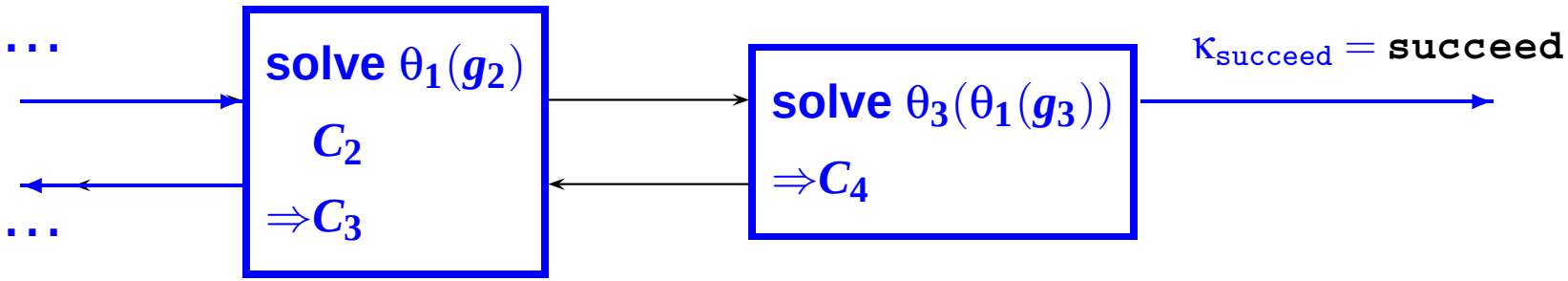
```
q(t(Z)).
```

via substitution $\theta_3 = \{Y_1 \mapsto t\ (Z_2)\}$.

But C_3 has no subgoals, so goal $\theta_1(g_2)$ immediately succeeds again.

Next apply substitutions θ_3 and θ_1 to clause C_1 to produce new subgoal $\theta_3(\theta_1(g_3))$, which is `r(t(Z)).`

Create a Byrd box for $\theta_3(\theta_1(g_3))$ that is prepared to consider clause C_4.



©Copyright 2006 Norman Ramsey. All Rights Reserved.

Example of Prolog backtracking (continued)

The goal $\theta_3(\theta_1(g_3))$ unifies with clause C_4

```
r(t(a)).
```

via substitution $\theta_4 = \{Z_2 \mapsto a\}$.

But C_4 has no subgoals, so goals $\theta_3(\theta_1(g_3))$ and $\theta_1(g_2)$ and g_1 all now succeed.

The solution reported is $\theta_4(\theta_3(\theta_1(g))$, that is,

```
X = f(t(a))
```

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Implementing Byrd Boxes using CPS

```
fun query database = let
  fun solveOne (goal as (func, args)) succ fail =
    find(func, builtins) args succ fail
    handle NotFound _ => let
      fun search [ ] = fail( )
        | search (clause :: clauses) =
          let fun resume( ) = search clauses
              val conclusion :- premises = freshen clause
              val subst = unify (goal, conclusion)
          in  solveMany (map (lift subst) premises)
                        subst succ resume
          end
          handle Unify => search clauses
    in search (potentialMatches (goal, database))
    end
```

To be continued...

©Copyright 2006 Norman Ramsey. All Rights Reserved.

Implementing Byrd Boxes using CPS (continued)

```
  and solveMany [ ] subst succ fail = succ subst fail
    | solveMany (goal::goals) subst succ fail =
        solveOne goal
                 (fn subst' => fn resume =>
                     solveMany (map (lift subst') goals)
                               (subst' o subst) succ resume)
                 fail
in fn goals => solveMany goals (fn x => x)
end
```

©Copyright 2006 Norman Ramsey. All Rights Reserved.