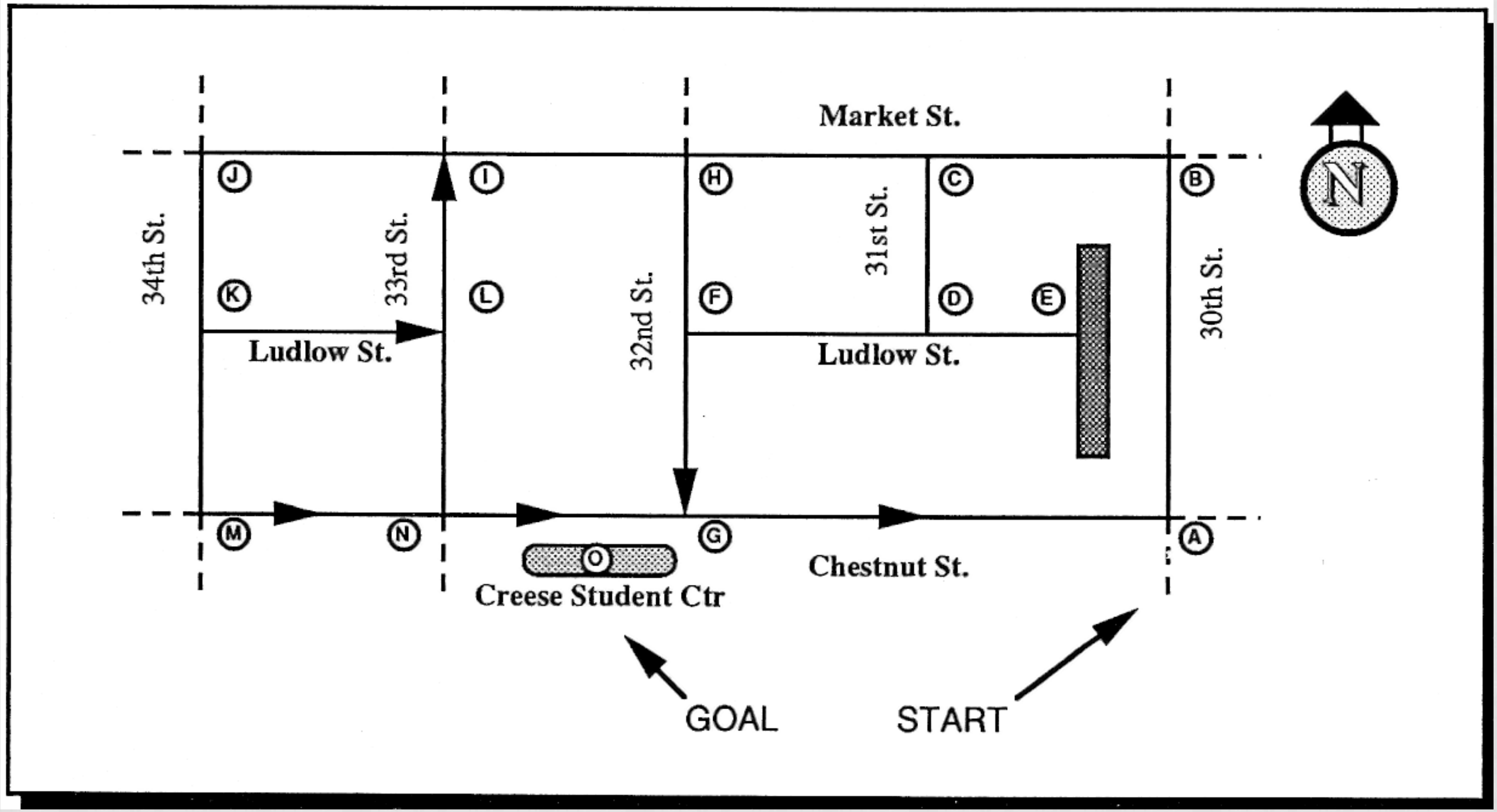
BACKTRACKING EXAMPLE

JEFFREY L. POPYACK

BACKTRACKING EXAMPLE



Drexel Campus, Early 1980's

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-1
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

state is “current location”

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

```
backTrack ( stateList )
=======================
   state = first element of stateList
   if state is a member of the rest of stateList, return 'FAILED-l
   if deadEnd?(state) return 'FAILED-2
   if goal(state), return NULL
   if length(stateList) > depthBound, return 'FAILED-3

   ruleSet = applicableRules(state)
   if ruleSet == NULL, return 'FAILED-4

   for each rule r in ruleSet,
     newState = applyRule(r,state)
     newStateList = addToFront(newState,stateList)
     path = backTrack(newStateList)
     if path ≠ 'FAILED return append(path,r)

   return 'FAILED-5
```

state is “current location”

a cycle has occurred: BAD

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

state is “current location”

a cycle has occurred: BAD

you recognize It's impossible to reach a solution from here

```
backTrack ( stateList )
=======================
   state = first element of stateList
   if state is a member of the rest of stateList, return 'FAILED-l
   if deadEnd?(state) return 'FAILED-2
   if goal(state), return NULL
   if length(stateList) > depthBound, return 'FAILED-3

   ruleSet = applicableRules(state)
   if ruleSet == NULL, return 'FAILED-4

   for each rule r in ruleSet,
     newState = applyRule(r,state)
     newStateList = addToFront(newState,stateList)
     path = backTrack(newStateList)
     if path ≠ 'FAILED return append(path,r)

   return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

state is "current location"

a cycle has occurred: BAD

you recognize It's impossible to reach a solution from here

Found solution – go no further

```
backTrack ( stateList )
========================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

list of states visited so far

returns a list of rules from start to goal

state is "current location"

a cycle has occurred: BAD

you recognize It's impossible to reach a solution from here

Found solution – go no further

establish a depth bound to prevent infinite recursion

```
backTrack ( stateList )
========================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-1
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

No Moves!!!

```
backTrack ( stateList )
========================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

No Moves!!!

For each rule that is applicable, try it and backtrack again.

- if success, return w/ this move at the front of solution

-if failure, try something else

```
backTrack ( stateList )
=======================
  state = first element of stateList
  if state is a member of the rest of stateList, return 'FAILED-l
  if deadEnd?(state) return 'FAILED-2
  if goal(state), return NULL
  if length(stateList) > depthBound, return 'FAILED-3

  ruleSet = applicableRules(state)
  if ruleSet == NULL, return 'FAILED-4

  for each rule r in ruleSet,
    newState = applyRule(r,state)
    newStateList = addToFront(newState,stateList)
    path = backTrack(newStateList)
    if path ≠ 'FAILED return append(path,r)

  return 'FAILED-5
```

No Moves!!!

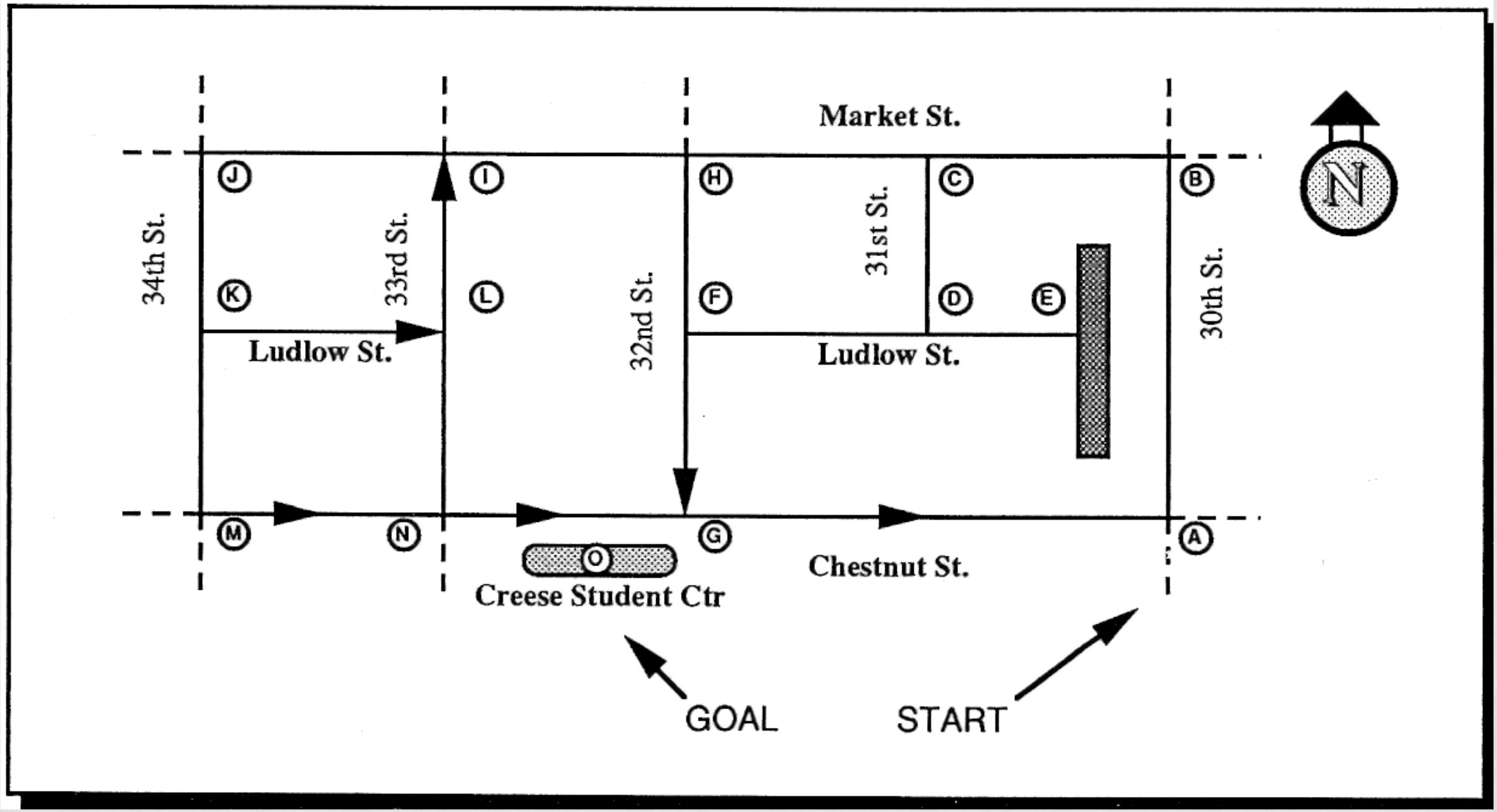
For each rule that is applicable, try it and backtrack again.

- if success, return w/ this move at the front of solution

-if failure, try something else

Nothing worked from here: failure!

BACKTRACKING EXAMPLE



Drexel Campus, Early 1980's

```
InitialState = A
StateList = {A}
backTrack ( StateList )
```

We will test rules for applicability in the order { ↑ ↓ → ← }

We will assume it is not possible to "drive off the map": only labeled nodes may be visited.

backtrack (StateList)

```
StateList : { A }

state = first (StateList) A
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ }
for each r ε ruleSet
    r :  ↑
    newstate = applyRule (r, state) B
    newStateList = { B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

```
StateList : { B A }

state = first (StateList) B
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓← }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) A
    newStateList = { A B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { A B A }

state = first (StateList) A

- state ε rest(StateList) ? Yes – return FAILED-1

backtrack (StateList)

```
StateList : { B A }

state = first (StateList) B
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓← }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) A
    newStateList = { A B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { B A }

state = first (StateList) B

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓← }

for each r ε ruleSet

```
    r : ↓
    newstate = applyRule (r, state) A
    newStateList = { A B A }
    path = backtrack (newStateList)
    path == FAILED
```

```
backtrack ( StateList )
StateList : { B A }

state = first (StateList) B
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓← }
for each r ε ruleSet
    r : ↓
    news
    newS
    path
    path

    r : ←
    newstate = applyRule (r, state) C
    newStateList = { C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

```
StateList : { C B A }

state = first (StateList) C
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }
for each r ε ruleSet
    r :  ↓
    newstate = applyRule (r, state) D
    newStateList = { D C B A }
    path = backtrack (newStateList)
```

```
backtrack ( StateList )
StateList : { D C B A }

state = first (StateList) D
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }
for each r ε ruleSet
    r : ↑
    newstate = applyRule (r, state) C
    newStateList = { C D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { C D C B A }

state = first (StateList) C

- state ε rest(StateList) ? Yes – return FAILED-1

```
backtrack ( StateList )
StateList : { D C B A }

state = first (StateList) D
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }
for each r ε ruleSet
    r : ↑
    newstate = applyRule (r, state) C
    newStateList = { C D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { D C B A }

state = first (StateList) D

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }

for each r ε ruleSet

r : ↑

newstate = applyRule (r, state) C

newStateList = { C D C B A }

path = backtrack (newStateList)

path == FAILED

backtrack (StateList)

StateList : { D C B A }

state = first (StateList) D

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }

for each r ε ruleSet

r : ↑
news
newS
path
path

r : →
newstate = applyRule (r, state) E
newStateList = { E D C B A }
path = backtrack (newStateList)

backtrack (StateList)

StateList : { E D C B A }

state = first (StateList) E

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ← }

for each r ε ruleSet

r : ←

newstate = applyRule (r, state) D

newStateList = { D E D C B A }

path = backtrack (newStateList)

backtrack (StateList)

StateList : { D E D C B A }

state = first (StateList) D

- state ε rest(StateList) ? Yes – return FAILED-1

```
backtrack ( StateList )
StateList : { E D C B A }

state = first (StateList) E
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ← }
for each r ε ruleSet
    r : ←
    newstate = applyRule (r, state) D
    newStateList = { D E D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { E D C B A }

state = first (StateList) E

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ← }

for each r ε ruleSet

```
r : ←
newstate = applyRule (r, state) D
newStateList = { D E D C B A }
path = backtrack (newStateList)
path == FAILED
```

backtrack (StateList)

StateList : { E D C B A }

state = first (StateList) E

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ← }

for each r ε ruleSet

```
r : ←
newstate = applyRule (r, state) D
newStateList = { D E D C B A }
path = backtrack (newStateList)
path == FAILED
```

return FAILED-5

backtrack (StateList)

```
StateList : { D C B A }

state = first (StateList) D
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }
for each r ε ruleSet
    r : ↑
    news
    newS
    path
    path

    r : →
    newstate = applyRule (r, state) E
    newStateList = { E D C B A }
    path = backtrack (newStateList)
```

```
backtrack ( StateList )
StateList : { D C B A }

state = first (StateList) D
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }
for each r ε ruleSet
    r : ↑
    news
    newS
    path
    path

    r : →
    newstate = applyRule (r, state) E
    newStateList = { E D C B A }
    path = backtrack (newStateList)
    path == FAILED
```

backtrack (StateList)

StateList : { D C B A }

state = first (StateList) D

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }

for each r ε ruleSet

r : ↑	r : →	r : ←
news	news	newstate = applyRule (r, state) F
newS	newS	newStateList = { F D C B A }
path	path	path = backtrack (newStateList)
path	path	

backtrack (StateList)

StateList : { F D C B A }

```
state = first (StateList) F
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) G
    newStateList = { G F D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { G F D C B A }

state = first (StateList) G

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ }

for each r ε ruleSet

r : ↓

newstate = applyRule (r, state) A

newStateList = { A G F D C B A }

path = backtrack (newStateList)

backtrack (StateList)

StateList : { A G F D C B A }

state = first (StateList) A

- state ε rest(StateList) ? Yes, FAILED-1

backtrack (StateList)

StateList : { G F D C B A }

state = first (StateList) G

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ }
for each r ε ruleSet

r : ↓
newstate = applyRule (r, state) A
newStateList = { A G F D C B A }
path = backtrack (newStateList)

backtrack (StateList)

StateList : { G F D C B A }

state = first (StateList) G

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ }

for each r ε ruleSet

r : ↓
newstate = applyRule (r, state) A
newStateList = { A G F D C B A }
path = backtrack (newStateList)
path == FAILED

backtrack (StateList)

StateList : { G F D C B A }

```
state = first (StateList) G
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) A
    newStateList = { A G F D C B A }
    path = backtrack (newStateList)
    path == FAILED
return FAILED-5
```

```
backtrack ( StateList )
StateList : { F D C B A }

state = first (StateList) F
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) G
    newStateList = { G F D C B A }
    path = backtrack (newStateList)
```

<u>backtrack (StateList)</u>

```
StateList : { F D C B A }

state = first (StateList) F
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }
for each r ε ruleSet
    r : ↓
    newstate = applyRule (r, state) G
    newStateList = { G F D C B A }
    path = backtrack (newStateList)
    path == FAILED
```

backtrack (StateList)

StateList : { F D C B A }

state = first (StateList) F

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }

for each r ε ruleSet

r : ↓
news
newS
path
path
	r : →
	newstate = applyRule (r, state) D
	newStateList = { D F D C B A }
	path = backtrack (newStateList)

backtrack (StateList)

StateList : { D F D C B A }

state = first (StateList) D

- state ε rest(StateList) ? Yes – return FAILED-1

```
backtrack ( StateList )
StateList : { F D C B A }

state = first (StateList) F
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }
for each r ε ruleSet
    r : ↓
    news
    newS
    path
    path

    r : →
    newstate = applyRule (r, state) D
    newStateList = { D F D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { F D C B A }

state = first (StateList) F

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }

for each r ε ruleSet

r : ↓	r : →
news	newstate = applyRule (r, state) D
newS	newStateList = { D F D C B A }
path	path = backtrack (newStateList)
path	path == FAILED

backtrack (StateList)

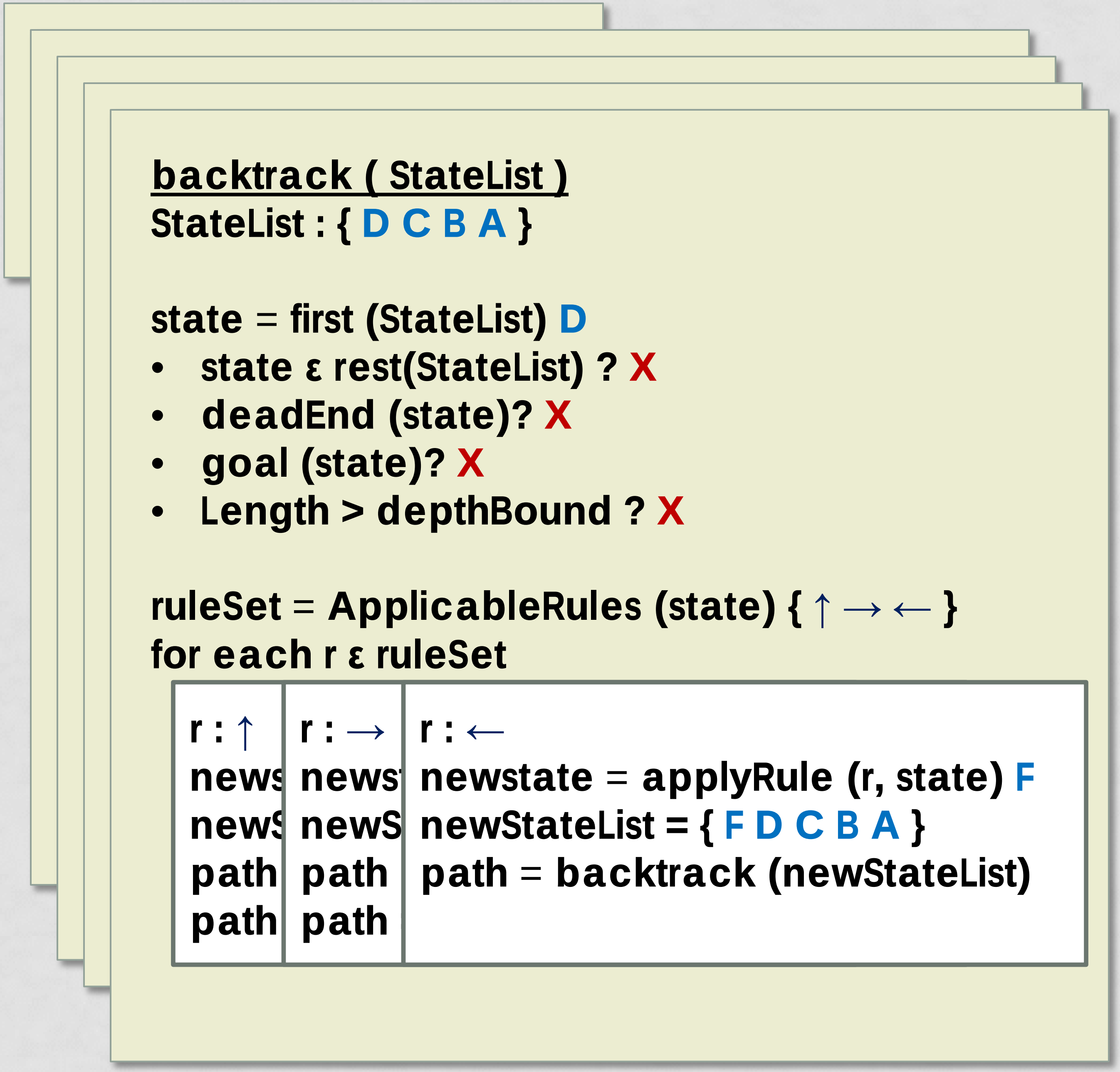
```
StateList : { F D C B A }

state = first (StateList) F
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → }
for each r ε ruleSet
```

r : ↓	r : →
news	newstate = applyRule (r, state) D
newS	newStateList = { D F D C B A }
path	path = backtrack (newStateList)
path	path == FAILED

```
return FAILED-5
```



```
backtrack ( StateList )
StateList : { D C B A }

state = first (StateList) D
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }
for each r ε ruleSet
```

r : ↑	r : →	r : ←
news	news	newstate = applyRule (r, state) F
newS	newS	newStateList = { F D C B A }
path	path	path = backtrack (newStateList)
path	path	

backtrack (StateList)

StateList : { D C B A }

state = first (StateList) D

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }

for each r ε ruleSet

r : ↑	r : →	r : ←
news	news	newstate = applyRule (r, state) F
newS	newS	newStateList = { F D C B A }
path	path	path = backtrack (newStateList)
path	path	path == FAILED

backtrack (StateList)

StateList : { D C B A }

state = first (StateList) D

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ → ← }

for each r ε ruleSet

r : ↑	r : →	r : ←
news	newst	newstate = applyRule (r, state) F
newS	newS	newStateList = { F D C B A }
path	path	path = backtrack (newStateList)
path	path	path == FAILED

return FAILED-5

backtrack (StateList)

```
StateList : { C B A }

state = first (StateList) C
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }
for each r ε ruleSet
    r :  ↓
    newstate = applyRule (r, state) D
    newStateList = { D C B A }
    path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { C B A }

state = first (StateList) C

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

r : ↓

newstate = applyRule (r, state) D

newStateList = { D C B A }

path = backtrack (newStateList)

path == FAILED

backtrack (StateList)

StateList : { C B A }

state = first (StateList) C

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

```
r :  ↓
newst
newSt
path =
path =
```

```
r : →
newstate = applyRule (r, state) B
newStateList = { B C B A }
path = backtrack (newStateList)
```

backtrack (StateList)

StateList : { B C B A }

state = first (StateList) B

- state ε rest(StateList) ? Yes – return FAILED-1

backtrack (StateList)

StateList : { C B A }

state = first (StateList) C

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

r : ↓

newst

newSt

path =

path =

r : →

newstate = applyRule (r, state) B

newStateList = { B C B A }

path = backtrack (newStateList)

backtrack (StateList)

StateList : { C B A }

state = first (StateList) C

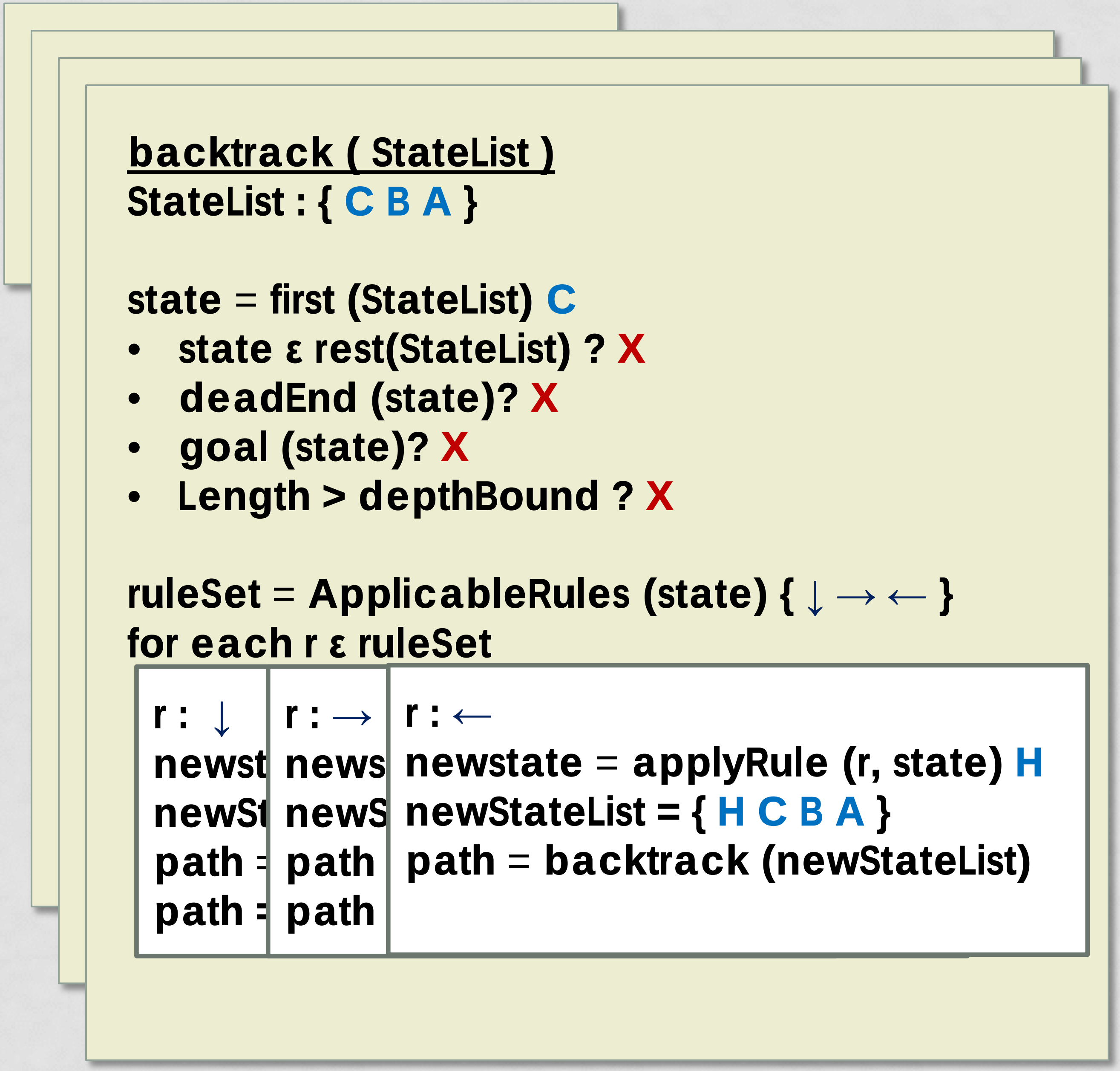
- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

r : ↓
newst
newSt
path =
path =

r : →
newstate = applyRule (r, state) B
newStateList = { B C B A }
path = backtrack (newStateList)
path == FAILED



backtrack (StateList)

StateList : { C B A }

state = first (StateList) C

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

r : ↓
newst
newSt
path =
path =

r : →
news
newS
path
path

r : ←
newstate = applyRule (r, state) H
newStateList = { H C B A }
path = backtrack (newStateList)

backtrack (StateList)

StateList : { H C B A }

state = first (StateList) H

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }

for each r ε ruleSet

r : ↓

newstate = applyRule (r, state) F

newStateList = { F H C B A }

path = backtrack (newStateList)

```
backtrack ( StateList )
StateList : { H C B A }

state = first (StateList) H
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }
for each r ε ruleSet
    r :  ↓
    newstate = applyRule (r, state) F
    newStateList = { F H C B A }
    path = backtrack (newStateList)
```

Note:
we have been through F already and it failed –

but we don't remember …

it will fail again

```
backtrack ( StateList )
StateList : { H C B A }

state = first (StateList) H
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↓ → ← }
for each r ε ruleSet
    r :  ↓
    newstate = applyRule (r, state) F
    newStateList = { F H C B A }
    path = backtrack (newStateList)
```

Note:
we have been through F already and it failed –

but we don’t remember …

it will fail again

Continue this process until eventually reaching the goal.

Eventually ...

...

backtrack (StateList)

StateList : { N M K J I H C B A }

state = first (StateList) N
- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { → ← }

for each r ε ruleSet

r : →

newstate = applyRule (r, state) O

newStateList = { O N M K J I H C B A }

path = backtrack (newStateList)

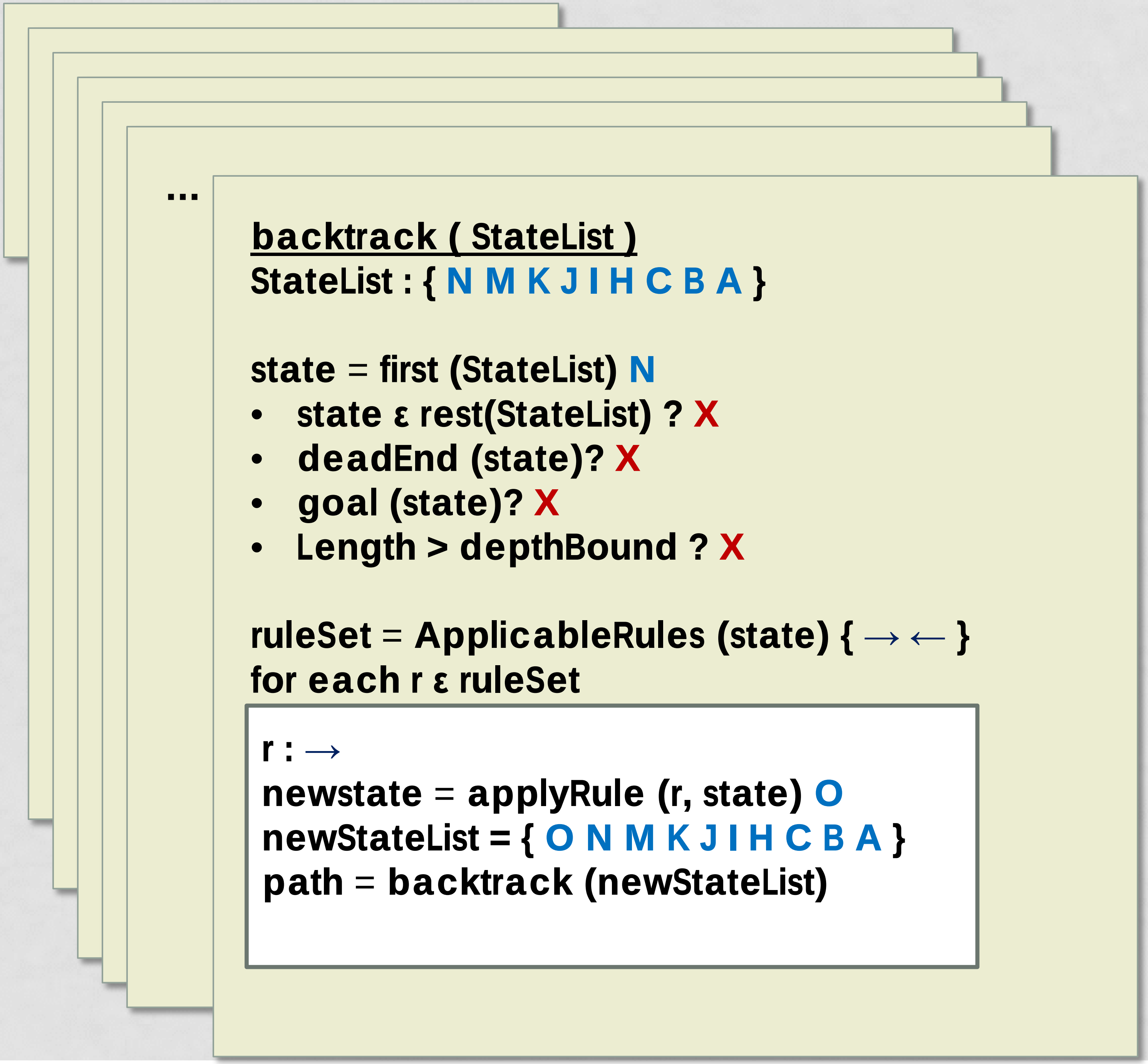
...

backtrack (StateList)

StateList : { O N M K J I H C B A }

state = first (StateList) O

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? YES! return NULL



```
...
backtrack ( StateList )
StateList : { N M K J I H C B A }

state = first (StateList) N
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { → ← }
for each r ε ruleSet
    r : →
    newstate = applyRule (r, state) O
    newStateList = { O N M K J I H C B A }
    path = backtrack (newStateList)
```

...

```
backtrack ( StateList )
StateList : { N M K J I H C B A }

state = first (StateList) N
•   state ε rest(StateList) ? X
•   deadEnd (state)? X
•   goal (state)? X
•   Length > depthBound ? X

ruleSet = ApplicableRules (state) { → ← }
for each r ε ruleSet
    r : →
    newstate = applyRule (r, state) O
    newStateList = { O N M K J I H C B A }
    path = backtrack (newStateList)
    path ≠ FAILED, return append(NULL, →)
```

..

<u>backtrack (StateList)</u>

StateList : { M K J I H C B A }

```
state = first (StateList) M
•  state ε rest(StateList) ? X
•  deadEnd (state)? X
•  goal (state)? X
•  Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ ← }
for each r ε ruleSet
```

r : ↑
news
newS
path
path
	r : →
	newstate = applyRule (r, state) N
	newStateList = { N M K J I H C B A }
	path = backtrack (newStateList)

..

backtrack (StateList)

StateList : { M K J I H C B A }

state = first (StateList) M

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ ← }

for each r ε ruleSet

r : ↑	r : →
news	newstate = applyRule (r, state) N
newS	newStateList = { N M K J I H C B A }
path	path = backtrack (newStateList)
path	path ≠ FAILED, return append({→}, →)

..

backtrack (StateList)

StateList : { M K J I H C B A }

state = first (StateList) M

- state ε rest(StateList) ? X
- deadEnd (state)? X
- goal (state)? X
- Length > depthBound ? X

ruleSet = ApplicableRules (state) { ↑ ← }

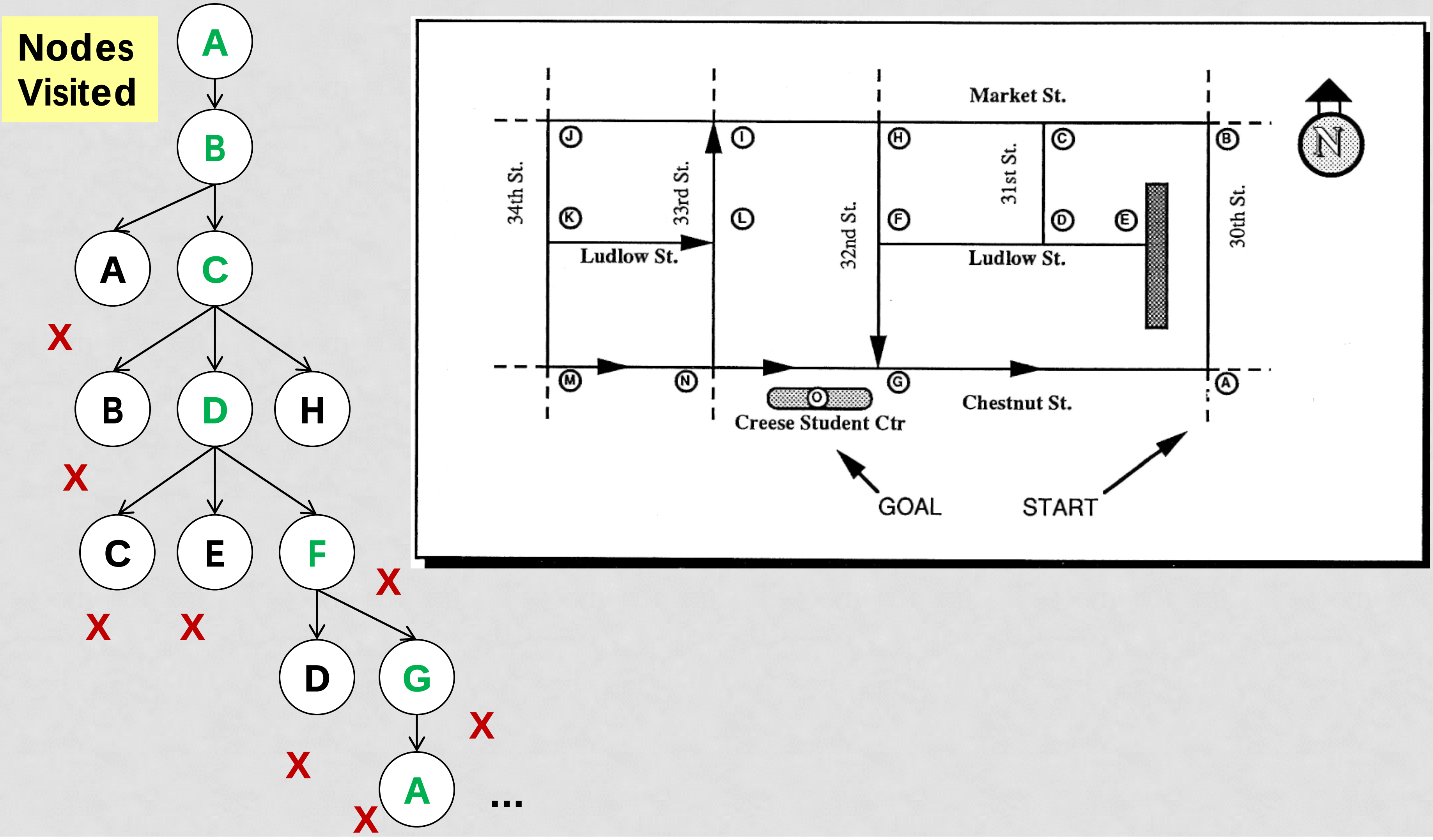
for each r ε ruleSet

r : ↑	r : →
news	newstate = applyRule (r, state) N
newS	newStateList = { N M K J I H C B A }
path	path = backtrack (newStateList)
path	path ≠ FAILED, return append({→}, →)

Continue... backtrack returns (↑ ← ← ← ← ↓ ↓ → →).

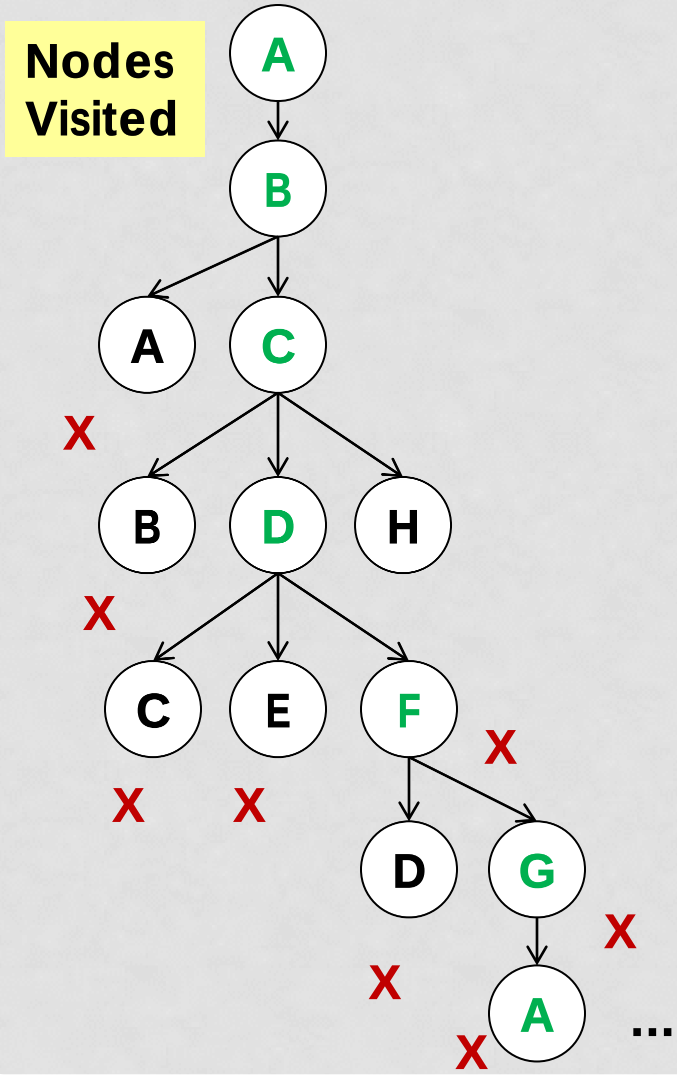
path: A → B → C → H → I → J → K → M → N → O

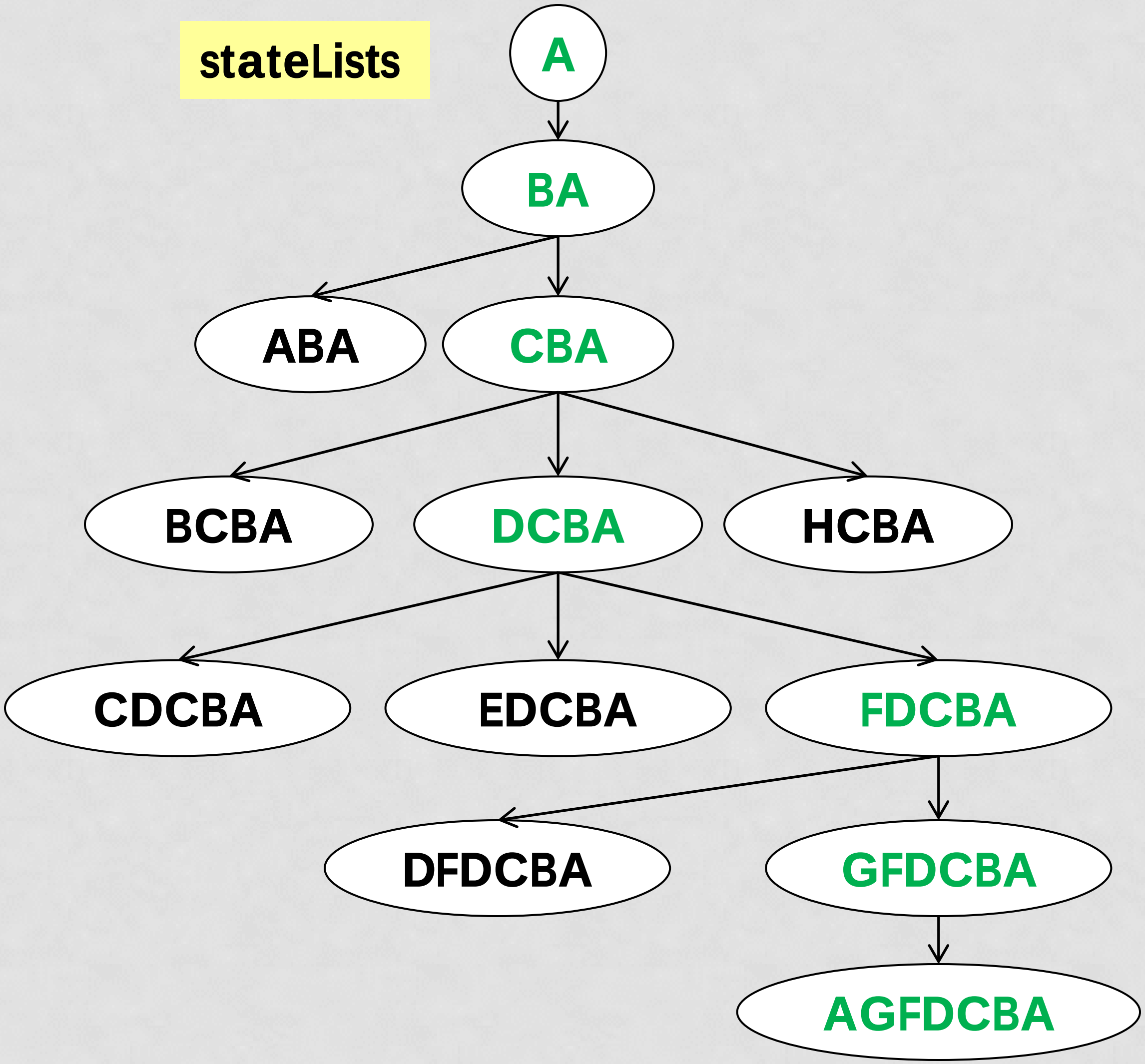
VIEWED AS A SEARCH TREE



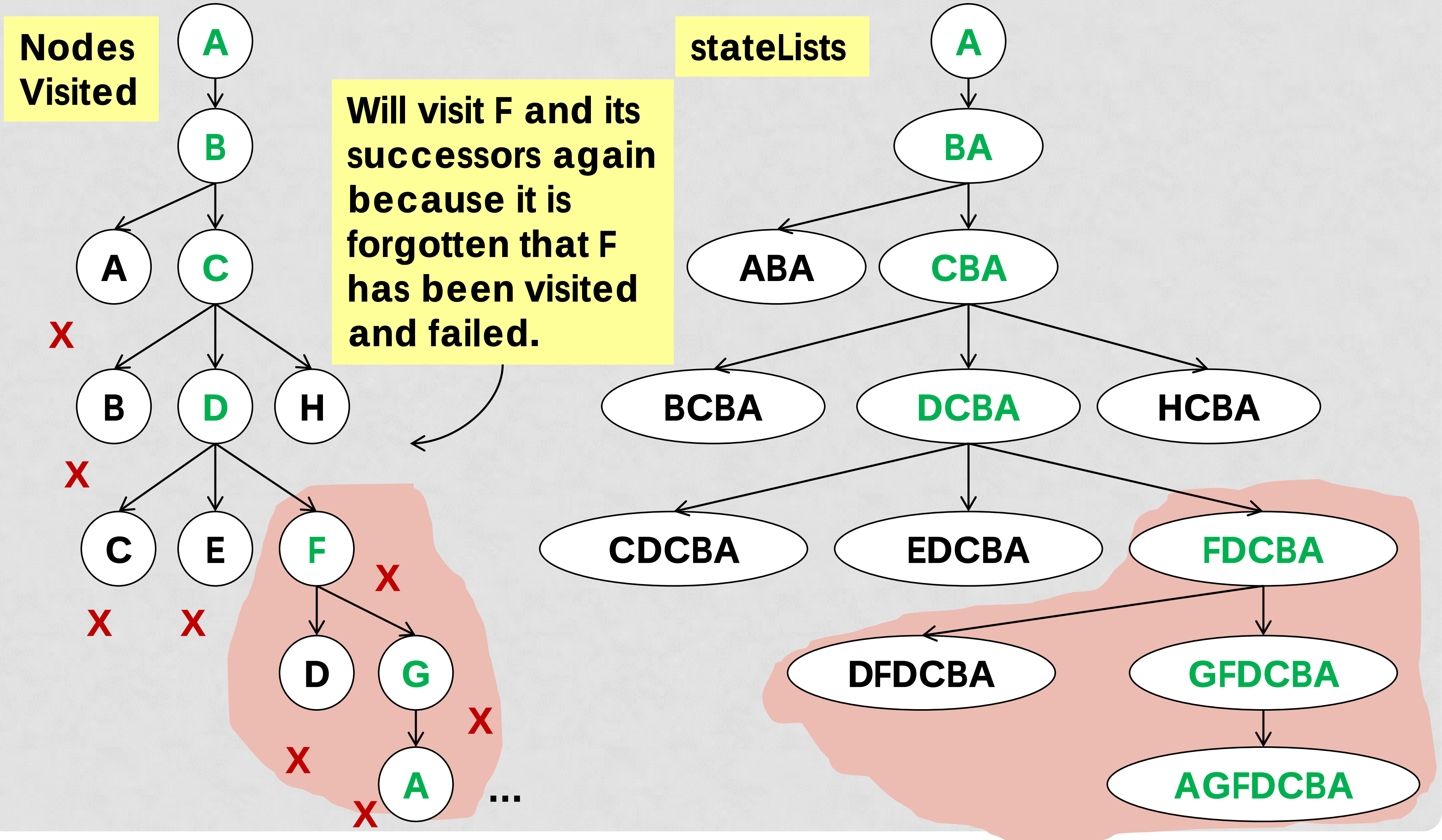
Nodes Visited

VIEWED AS A SEARCH TREE





VIEWED AS A SEARCH TREE



Nodes Visited

Will visit F and its successors again because it is forgotten that F has been visited and failed.

stateLists

MORAL OF THE STORY

- backTrack works
- Can be wasteful
- More efficient to remember all states visited, not just current path
- More intelligent to consider rules in a reasonable order, not just { ↑ ↓ → ← } every time.

REFINEMENTS

Iterative Deepening Backtrack:

LOOP:

```
Start with depthBound = n
stateList = { InitialState }
path = backTrack ( stateList, depthBound )
if path ≠ FAILED
    Exit with Success!
Increase depthBound
```

EFFICIENCY

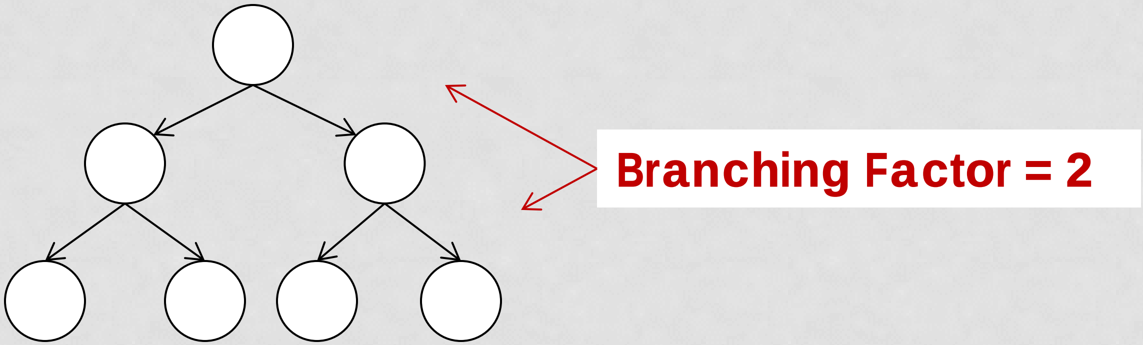
- Is **Iterative Deepening Backtrack** *efficient* or *inefficient*?

- *Inefficient*:
  *In order to solve with **MaxDepth=n+1**, we re-visit all paths we visited when solving with **MaxDepth=n**.*

- *Efficient*:
  But maybe that isn't so bad after all…

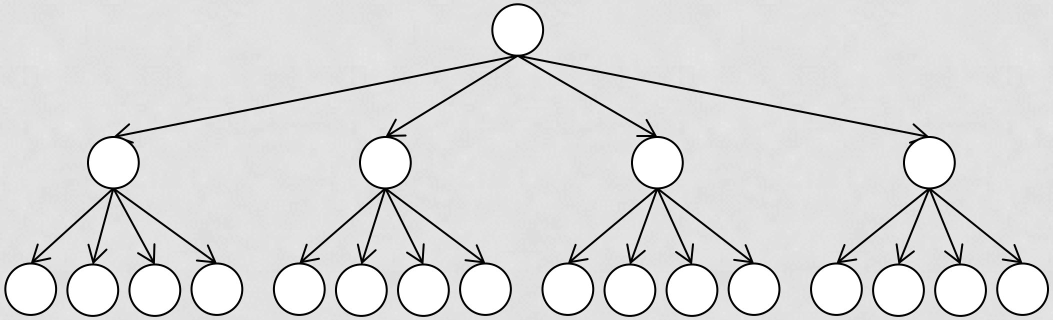
EFFICIENCY

- The ***branching factor*** for a given node is its number of successors – in effect, by what factor does the work increase by examining its successors?
- In general, the ***effective branching factor*** for a problem is the factor by which the number of nodes increases by examining another level of nodes.



Branching Factor = 2

BRANCHING FACTOR



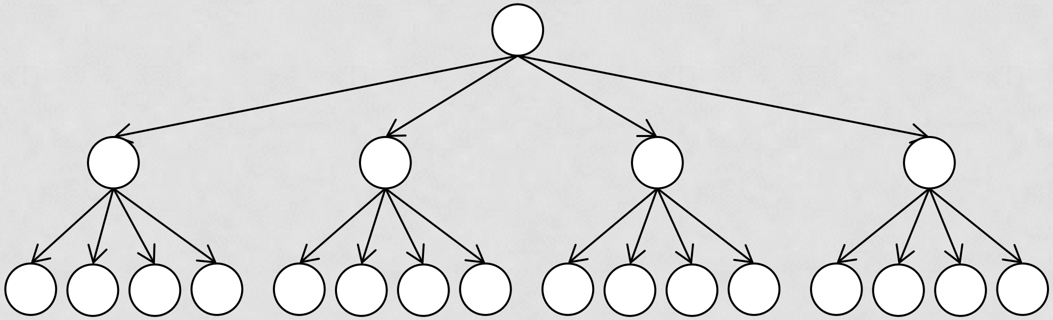
Level	Nodes at Level	Total Nodes in Tree
0	1	1
1	4	5
2	16	21

Total Nodes at Level ***n*** $= k^0 + k^1 + \ldots + k^n$

$$\boldsymbol{T(n)} = (k^{n+1} - 1)/(k - 1)$$

(*k* = *4* in example)

BRANCHING FACTOR



Level	Nodes at Level	Total Nodes in Tree
0	1	1
1	4	5
2	16	21

Total Nodes at Level ***n*** $= k^0 + k^1 + \ldots + k^n$

$$\boldsymbol{T(n)} = (k^{n+1} - 1)/(k - 1)$$

($k = 4$ in example)

Note $\boldsymbol{T(n+1) = k * T(n) + 1}$

(Each level has 4 times as many nodes as previous)

And so, re-doing the work for levels *1… n-1* is small compared to amount of work needed to do level *n*.