INTELLIGENT SYSTEMS (CSE-303-F)

Section B

Rule Based Deduction System

Rule-Based Deduction Systems

The way in which a piece of knowledge is expressed by a human expert carries important information,

    example: if the person has fever and feels tummy-pain then she may have an infection.

In logic it can be expressed as follows:

$$\forall x.\ (has\_fever(x)\ \&\ tummy\_pain(x) \rightarrow has\_an\_infection(x))$$

If we convert this formula to clausal form we loose the content as then we may have equivalent formulas like:

    (i) $has\_fever(x)\ \&\ {\sim}has\_an\_infection(x) \rightarrow {\sim}tummy\_pain(x)$
    (ii) ${\sim}has\_an\_infection(x)\ \&\ tummy\_pain(x) \rightarrow {\sim}has\_fever(x)$

Notice that:
- (i) and (ii) are logically equivalent to the original sentence
- they have lost the main information contained in its formulation.

Forward production systems

- The main idea behind the forward/backward production systems is:
  - to take advantage of the implicational form in which production rules are stated by the expert
  - and use that information to help achieving the goal.
- In the present systems the formulas have two forms:
  - rules
  - and facts

Forward production systems

- Rules are the productions stated in implication form.
  - Rules express specific knowledge about the problem.
  - Facts are assertions not expressed as implications.
  - The task of the system will be to prove a goal formula with these facts and rules.
  - In a forward production system the rules are expressed as *F-rules*
  - F-rules operate on the global database of facts until the termination condition is achieved.
  - This sort of proving system is a *direct* system rather than a refutation system.

- Facts
  - Facts are expressed in AND/OR form.
  - An expression in AND/OR form consists on sub-expressions of literals connected by & and V symbols.
  - An expression in AND/OR form is not in clausal form.

Forward production systems

Steps to transform facts into AND/OR form for forward system:

1. *Eliminate* (temporarily) implication symbols.
2. Reverse quantification of variables in first disjunct by *moving negation symbol.*
3. *Skolemize* existential variables.
4. Move all **universal quantifiers** to the front an drop.
5. **Rename variables** so the same variable does not occur in different main conjuncts
   - Main conjuncts are small AND/OR trees, not necessarily sum of literal clauses as in Prolog.

EXAMPLE

Original formula: $\exists u. \forall v. \{q(v, u)$ & ~$[[r(v) \vee p(v)]$ & $s(u,v)]\}$
converted formula: $q(w, a)$ **&** $\{[\sim r(v)$ & $\sim p(v)] \vee \sim s(a,v)\}$

Conjunction of two main conjuncts

Various variables in conjuncts

All variables appearing on the final expressions are assumed to be universally quantified.

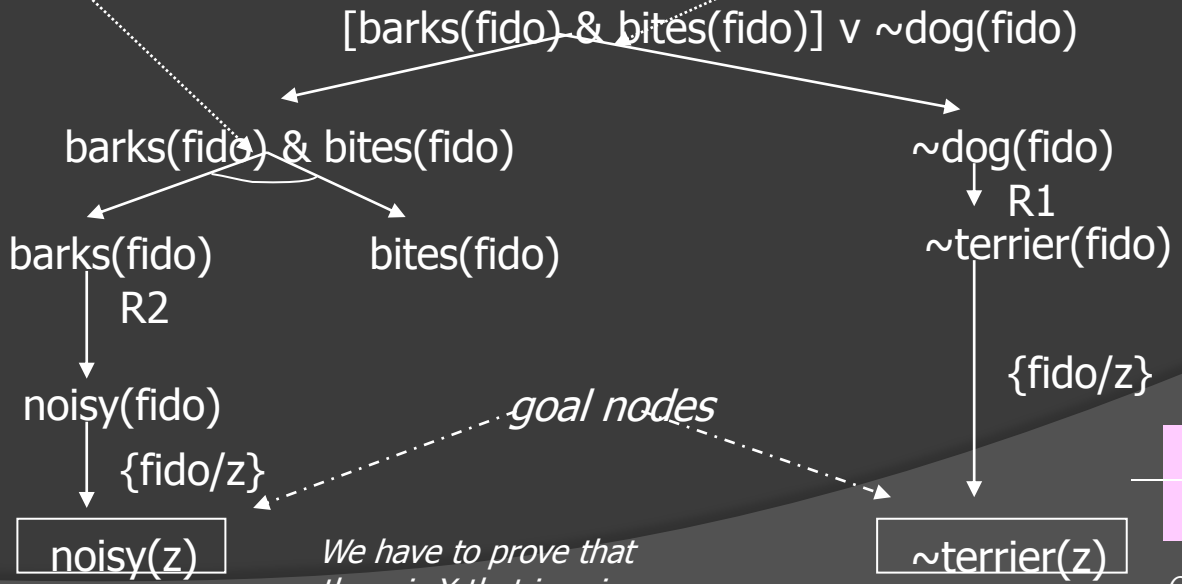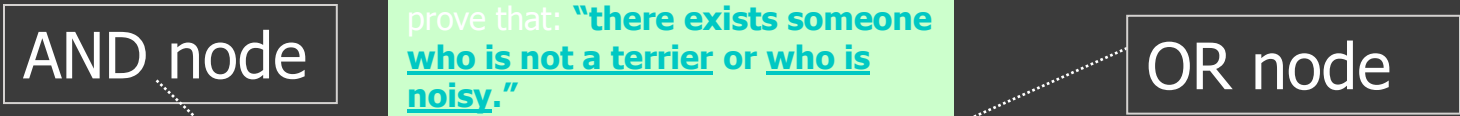# Rule-Based Deduction Systems: forward production systems

F-rules

Rules in a forward production system will be applied to the AND/OR graph to produce new transformed graph structures.

We assume that rules in a forward production system are of the form:

L ==> W,

where L is a literal and W is a formula in AND/OR form.

◉ Recall that a rule of the form (L1 V L2) ==> W is equivalent to the pair of rules: L1 ==> W V L2 ==> W.

AND node

prove that: **"there exists someone who is not a terrier or who is noisy."**

OR node

[barks(fido) & bites(fido)] v ~dog(fido)

barks(fido) & bites(fido)

~dog(fido)
R1
~terrier(fido)

We cannot prove this branch but we do not have to since one branch of OR was proven by showing Fido

barks(fido)   bites(fido)

R2

noisy(fido)

{fido/z}

goal nodes

{fido/z}

•Dog(Fido)

•barks(Fido)

•Not terrier(Fido)|

•Noisy(Fido)

noisy(z)

We have to prove that there is X that is noisy.
X=Fido

~terrier(z)

•NOT Dog(Fido)

•Not terrier(Fido)|

Or we have to prove that there is X that X is not a terrier

forward production systems

Steps to transform the rules into a <u>free-quantifier form</u>:
1.   Eliminate (temporarily) implication symbols.
2.   Reverse quantification of variables in first disjunct by moving negation symbol.
3.   Skolemize existential variables.
4.   Move all universal quantifiers to the front and drop.
5.   Restore implication.

All variables appearing on the final expressions are assumed to be universally quantified.

E.g.   Original formula: $\forall x.(\exists y. \forall z. (p(x, y, z)) \rightarrow \forall u. q(x, u))$
        Converted formula: $p(x, y, f(x, y)) \rightarrow q(x, u)$.

Skolem function

Restored implication

forward production systems

A full **_example_:**

- <u>Fact:</u> Fido barks and bites, or Fido is not a dog.
- (R1) All terriers are dogs.
- (R2) Anyone who barks is noisy.

Based on these facts, prove that: **"there exists someone <u>who is not a terrier</u> or <u>who is noisy</u>."**

Logic representation:

(barks(fido) & bites(fido)) ∨ ~dog(fido)
R1: terrier(x) → dog(x)
R2: barks(y) → noisy(y)

goal: ∃w.(~terrier(w) ∨ noisy(w))

goal

Rule-Based Deduction Systems: forward production systems

From facts to goal

AND node

OR node

AND/OR Graph for the 'terrier' problem:

[barks(fido) & bites(fido)] v ~dog(fido)

barks(fido) & bites(fido)

~dog(fido)

R1 *applied in reverse*

~terrier(fido)

barks(fido)

bites(fido)

R2 applied forward

noisy(fido)

goal nodes

{fido/z}

{fido/z}

noisy(z)

~terrier(z)
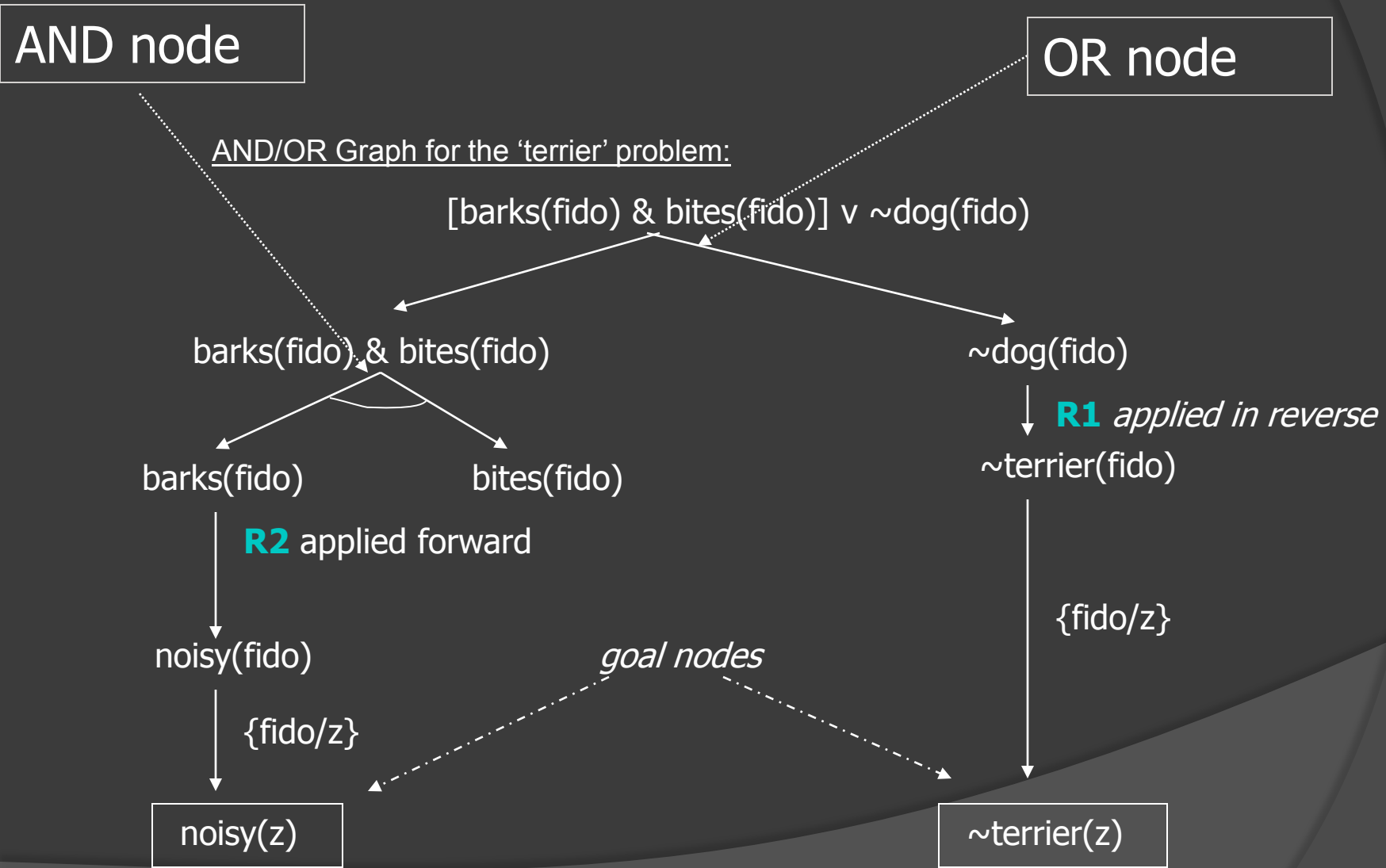
Backward production systems

B-Rules

We restrict B-rules to expressions of the form: W ==> L,

where W is an expression in AND/OR form and L is a literal,

and the scope of quantification of any variables in the implication is the entire implication.

Recall that W==>(L1 & L2) is equivalent to the two rules: W==>L1 and W==>L2.

An important property of logic is the duality between assertions and goals in theorem-proving systems.

Duality between assertions and goals allows the goal expression to be treated as if it were an assertion.

Conversion of the goal expression into AND/OR form:

1. Elimination of implication symbols.
2. Move negation symbols in.
3. Skolemize *existential* variables.
4. Drop existential quantifiers. Variables remaining in the AND/OR form are considered to be existentially quantified.

Goal clauses are conjunctions of literals and the disjunction of these clauses is the clause form of the goal well-formed formula.

Example 1 of formulation of Rule-Based Deduction Systems

1. Facts:

> dog(fido)
> ~barks(fido)
> wags-tail(fido)
> meows(myrtle)

Rules:

> R1: [wags-tail(x1) & dog(x1)] → friendly(x1)
> R2: [friendly(x2) & ~barks(x2)] → ~afraid(y2,x2)
> R3: dog(x3) → animal(x3)
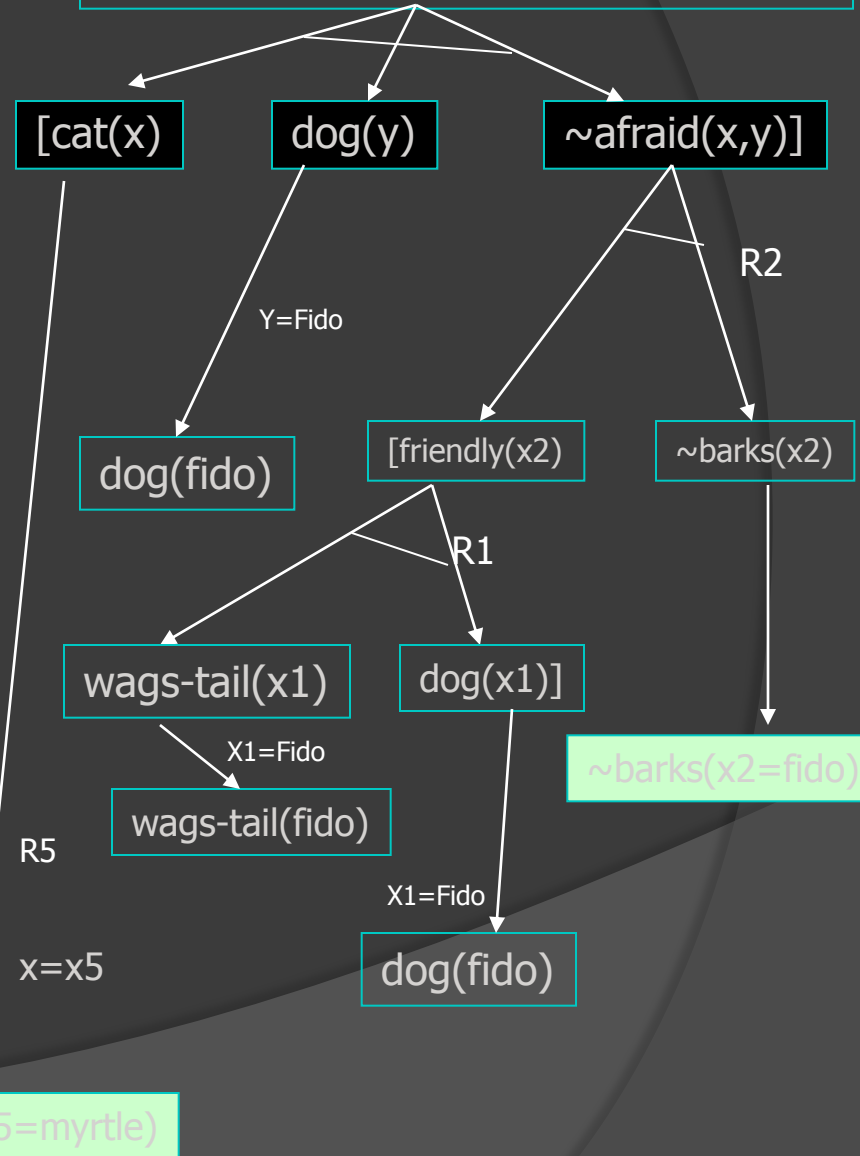> R4: cat(x4) → animal(x4)
> R5: meows(x5) → cat(x5)

Suppose we want to ask if there are a cat and a dog such that the cat is unafraid of the dog.

The goal expression is:

> ∃x. ∃y.[cat(x) & dog(y) & ~afraid(x,y)]

We treat the goal expression as an assertion

R2

∃x. ∃y.[cat(x) & dog(y) & ~afraid(x,y)]

[cat(x)   dog(y)   ~afraid(x,y)]

Y=Fido

dog(fido)   [friendly(x2)   ~barks(x2)

R2

R1

wags-tail(x1)   dog(x1)]

X1=Fido

wags-tail(fido)

X1=Fido

dog(fido)

~barks(x2=fido)

R5

x=x5

meows(x5=myrtle)

2. The blocks-word situation is described by the following set of wffs:

on_table(a)   clear(e)
on_table(c)   clear(d)
on(d,c)   heavy(d)
on(b,a)   wooden(b)
heavy(b)   on(e,b)

The following statements provide general knowledge about this blocks word:

Every big, blue block is on a green block.

Each heavy, wooden block is big.

All blocks with clear tops are blue.

All wooden blocks are blue.

Represent these statements by a set of implications having single-literal consequents.

Draw a consistent AND/OR solution tree (using B-rules) that solves the problem: "Which block is on a green block?"

HOMEWORK Problem 2. Transformation of rules and goal:

Facts:

f1: on_table(a)	f6: clear(e)
f2: on_table(c)	f7: clear(d)
f3: on(d,c)	f8: heavy(d)
f4: on(b,a)	f9: wooden(b)
f5: heavy(b)	f10: on(e,b)

Rules:

R1: $big(y1) \wedge blue(y1) \rightarrow green(g(y1))$ Every big, blue block is on a green block.

R2: $big(y0) \wedge blue(y0) \rightarrow on(y0,g(y0))$ "   "   "   "   " " "   "

R3: $heavy(z) \wedge wooden(z) \rightarrow big(z)$ Each heavy, wooden block is big.

R4: $clear(x) \rightarrow blue(x)$ All blocks with clear tops are blue.

R5: $wooden(w) \rightarrow blue(w)$ All wooden blocks are blue.

Goal:

$green(u) \wedge on(v,u)$ Which block is on a green block?

HOMEWORK PROBLEM 3. Information Retrieval System

- We have a set of facts containing personnel data for a business organization
- and we want an automatic system to answer various questions about personal matters.

⊙ Facts

John Jones is the manager of the Purchasing Department.

    manager(p-d,john-jones)

    works_in(p-d, joe-smith)

    works_in(p-d,sally-jones)

    works_in(p-d,pete-swanson)

Harry Turner is the manager of the Sales Department.

    manager(s-d,harry-turner)

    works_in(s-d,mary-jones)

    works_in(s-d,bill-white)

    married(john-jones,mary-jones)

Rule-Based Deduction Systems

Rules

place   person

R1: manager(x,y) ➔ works_in(x,y)

R2: works_in(x,y) & manager(x,z) ➔ boss_of(y,z)   person

R3: works_in(x,y) & works_in(x,z) ➔ ~married(y,z)

In this company married people should not work in the same department

R4: married(y,z) ➔ married(z,y)

R5: [married(x,y) & works_in(p-d,x) ➔ insured_by(x,eagle-corp)

With these facts and rules a simple backward production system can answer a variety of questions.

Build solution graphs for the following questions:

1. Name someone who works in the Purchasing Department.
2. Name someone who is married and works in the sales department.
3. Who is Joe Smith's boss?
4. Name someone insured by Eagle Corporation.
5. Is John Jones married with Sally Jones?

Planning

- Planning is fundamental to "intelligent" behavior. E.g.
  - assembling tasks   - route finding
  - planning chemical processes   - planning a report

- **Representation**

  The planner has to represent states of the world it is operating within, and to predict consequences of carrying actions in its world. E.g.
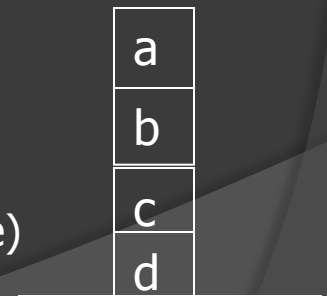
  initial state: final state:

  on(a,b)
  on(b,table)
  on(d,c)
  on(c,table)
  clear(a)
  clear(d)

  on(a,b)
  on(b,c)
  on(c,d)
  on(d,table)
  clear(a)

Planning

- Representing an action

  One standard method is by specifying sets of preconditions and effects, e.g.

  pickup(X) :

  > preconditions: clear(X), handempty.
  >
  > deletlist: on(X,_), clear(X), handempty.
  >
  > addlist: holding(X).

Planning

- ## <u>The Frame Problem in Planning</u>
- This is the problem of how to keep track in a representation of the world of all the effects that an action may have.

- The action representation given is the one introduced by STRIPS (Nilsson) and is an attempt to a solution to the frame problem
  - but it is only adequate for simple actions in simple worlds.

- ## <u>The Frame Axiom</u>
- The frame axiom states that a <u>fact is true</u> if it is not in the last delete list and was true in the previous state.
- The frame axiom states that a <u>fact is false</u> if it is not in the last add list and was false in the previous state.

Planning

- Control Strategies
    - Forward Chaining
    - Backward Chaining

    The choice on which of these strategies to use depends on the problem, normally backward chaining is more effective.

Planning

c

a

b

a

b

c

Example:

<u>Initial State</u>

    clear(b), clear(c), on(c,a), ontable(a), ontable(b), handempty

<u>Goal</u>

    on(b,c) & on(a,b)

<u>Rules</u>

R1: pickup(x)	R2: putdown(x)
P & D: ontable(x), clear(x), handempty	P & D: holding(x)
A: holding(x)	A: ontable(x), clear(x), handempty
R3: stack(x,y)	R4: unstack(x,y)
P & D: holding(x), clear(y)	P & D: on(x,y), clear(x), handempty
A: handempty, on(x,y), clear(x)	A: holding(x), clear(y)

Planning

{unstack(c,a), putdown(c), pickup(b), stack(b,c), pickup(a), stack(a,b)}

c
a

b

a
b
c

Initial situation

next situation

TRIANGLE TABLE   {unstack(c,a), putdown(c), pickup(b), stack(b,c), pickup(a), stack(a,b)}

0

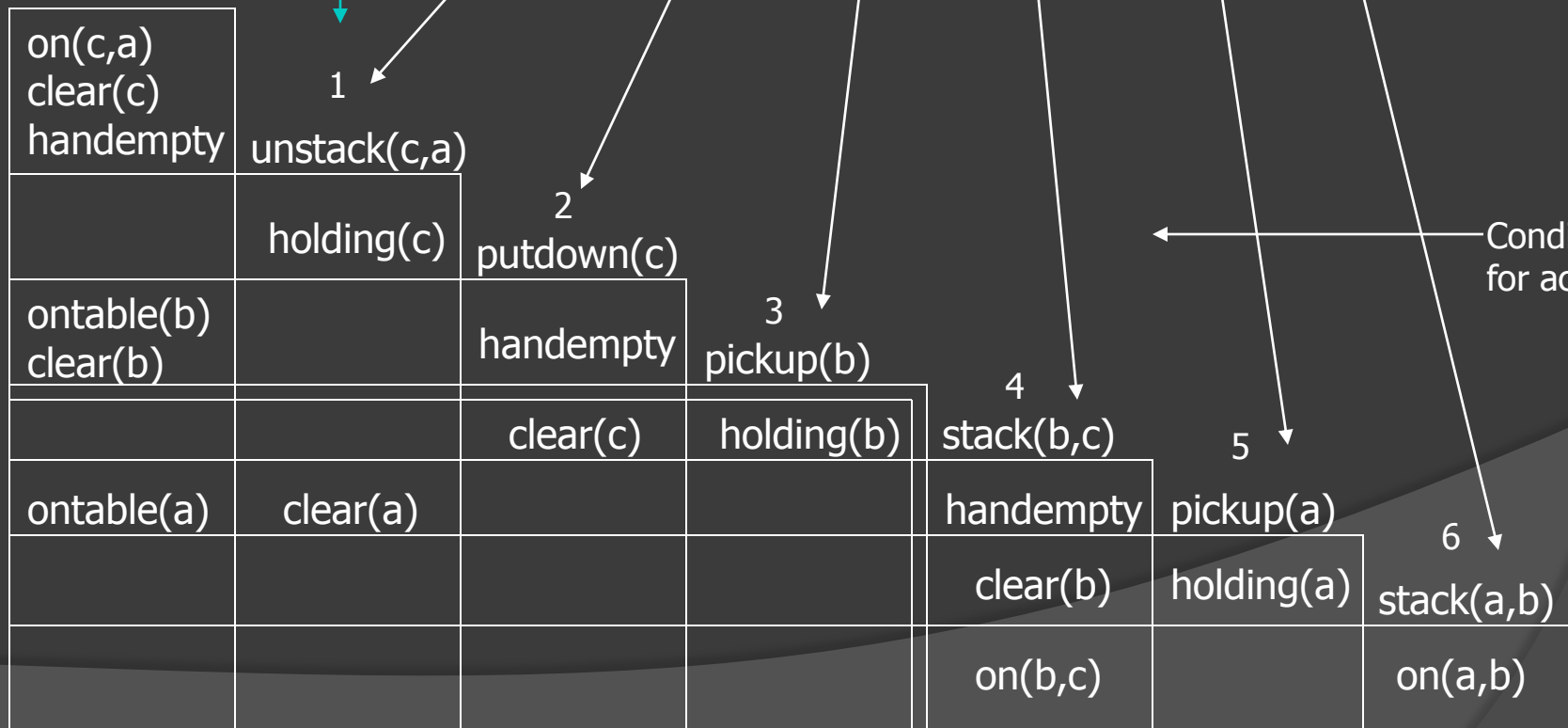| on(c,a) clear(c) handempty						
	unstack(c,a) 1					
	holding(c)	putdown(c) 2				
ontable(b) clear(b)		handempty	pickup(b) 3			
		clear(c)	holding(b)	stack(b,c) 4		
ontable(a)	clear(a)			handempty	pickup(a) 5	
				clear(b)	holding(a)	stack(a,b) 6
				on(b,c)		on(a,b)

Conditions for action

goal

Planning

Homework and exam exercises

1. Describe how the two SCRIPS rules **pickup**(x) and **stack**(x,y) could be combined into a macro-rule **put**(x,y).

   What are the preconditions, delete list and add list of the new rule.

   Can you specify a general procedure for creating macro-rules components?

1. Consider the problem of devising a plan for a kitchen-cleaning robot.

   (i) Write a set of STRIPS-style operators that might be used.

   - When you describe the operators, take into account the following considerations:

     (a) Cleaning the stove or the refrigerator will get the floor dirty.

     (b) The stove must be clean before covering the drip pans with tin foil.

     (c) Cleaning the refrigerator generates garbage and messes up the counters.

     (d) Washing the counters or the floor gets the sink dirty.

   (ii) Write a description of an initial state of a kitchen that has a dirty stove, refrigerator, counters, and floor.

   (The sink is clean, and the garbage has been taken out).

   Also write a description of the goal state where everything is clean, there is no trash, and the stove drip pans have been covered with tin foil.