CS 1571 Introduction to AI
Lecture 17

Inference in first-order logic. Production systems.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Sentences in Horn normal form

- **Horn normal form (HNF) in the propositional logic**
  - a special type of clause with at most one positive literal

$$(A \vee \neg B) \wedge (\neg A \vee \neg C \vee D)$$

Typically written as: $(B \Rightarrow A) \wedge ((A \wedge C) \Rightarrow D)$

- A clause with one literal, e.g. A, is also called **a fact**
- A clause representing an implication (with a conjunction of positive literals in antecedent and one positive literal in consequent), is also called **a rule**

- **Generalized Modus ponens:**
  - is the **complete inference rule** for KBs in the Horn normal form. **Not all KBs are convertible to HNF !!!**

Horn normal form in FOL

First-order logic (FOL)

– **adds variables and quantifiers, works with terms**

Generalized modus ponens rule:

$$\sigma = \text{a substitution s.t. } \forall i \; SUBST(\sigma, \phi_i') = SUBST(\sigma, \phi_i)$$

$$\frac{\phi_1', \phi_2' \ldots, \phi_n', \quad \phi_1 \wedge \phi_2 \wedge \ldots \phi_n \Rightarrow \tau}{SUBST\ (\sigma, \tau)}$$

Generalized modus ponens:

- is **complete** for the KBs with sentences in Horn form;
- Not all first-order logic sentences can be expressed in this form

Forward and backward chaining

Two inference procedures based on modus ponens for **Horn KBs**:

- **Forward chaining**

  Idea: Whenever the premises of a rule are satisfied, infer the conclusion. Continue with rules that became satisfied.

  Typical usage: infer all sentences entailed by the existing KB.

- **Backward chaining (goal reduction)**

  Idea: To prove the fact that appears in the conclusion of a rule prove the premises of the rule. Continue recursively.

  Typical usage: If we want to prove that the target (goal) sentence α is entailed by the existing KB.

Both procedures are **complete for KBs in Horn form** !!!

Forward chaining example

- **Forward chaining**

  Idea: Whenever the premises of a rule are satisfied, infer the conclusion. Continue with rules that became satisfied

  Assume the KB with the following rules:

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$

R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$

R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

F1: $Steamboat\ (Titanic\ )$

F2: $Sailboat\ (Mistral\ )$

F3: $RowBoat(PondArrow)$

Theorem: $Faster\ (Titanic\ , PondArrow\ )$ **?**

Forward chaining example

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$

R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$

R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

F1: $Steamboat\ (Titanic\ )$

F2: $Sailboat\ (Mistral\ )$

F3: $RowBoat(PondArrow)$

?

Forward chaining example

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$
R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$
R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

F1: $Steamboat\ (Titanic\ )$
F2: $Sailboat\ (Mistral\ )$
F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$ ⬅

Forward chaining example

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$
R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$
R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

F1: $Steamboat\ (Titanic\ )$
F2: $Sailboat\ (Mistral\ )$
F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$ ⬅

Rule R2 is satisfied:

F5: $Faster(Mistral, PondArrow)$ ⬅

Forward chaining example

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$

R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$

R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

F1: $Steamboat\ (Titanic\ )$

F2: $Sailboat\ (Mistral\ )$

F3: $RowBoat(PondArrow)$

Rule R1 is satisfied:

F4: $Faster(Titanic, Mistral)$

Rule R2 is satisfied:

F5: $Faster(Mistral, PondArrow)$

Rule R3 is satisfied:

F6: $Faster(Titanic, PondArrow)$

Backward chaining example

- **Backward chaining (goal reduction)**

  Idea: To prove the fact that appears in the conclusion of a rule prove the antecedents (if part) of the rule & repeat recursively.

KB: R1: $Steamboat\ (x) \wedge Sailboat\ (y) \Rightarrow Faster\ (x, y)$

R2: $Sailboat\ (y) \wedge RowBoat\ (z) \Rightarrow Faster\ (y, z)$

R3: $Faster\ (x, y) \wedge Faster\ (y, z) \Rightarrow Faster\ (x, z)$

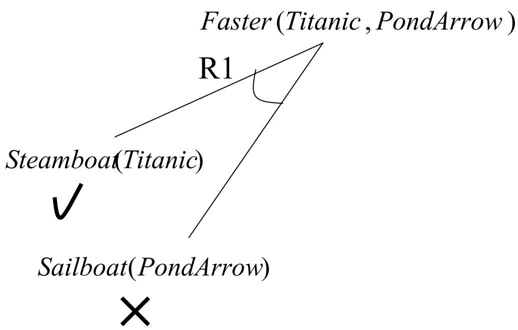
F1: $Steamboat\ (Titanic\ )$

F2: $Sailboat\ (Mistral\ )$

F3: $RowBoat(PondArrow)$

Theorem: $Faster\ (Titanic\ , PondArrow\ )$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

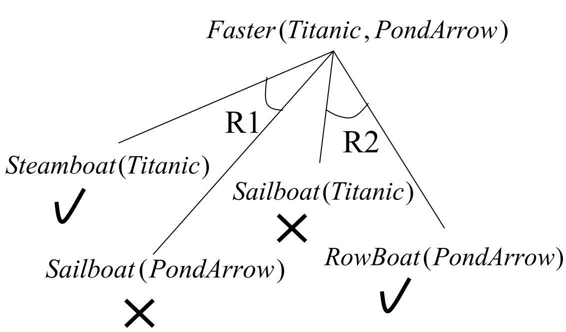
F3: $RowBoat(PondArrow)$

$$Steamboat(x) \wedge Sailboat(y) \Rightarrow Faster(x, y)$$

$$Faster(Titanic, PondArrow)$$

$$\{x / Titanic, y / PondArrow\}$$

Backward chaining example



F1: $Steamboat(Titanic)$

F2: $Sailboat(Mistral)$

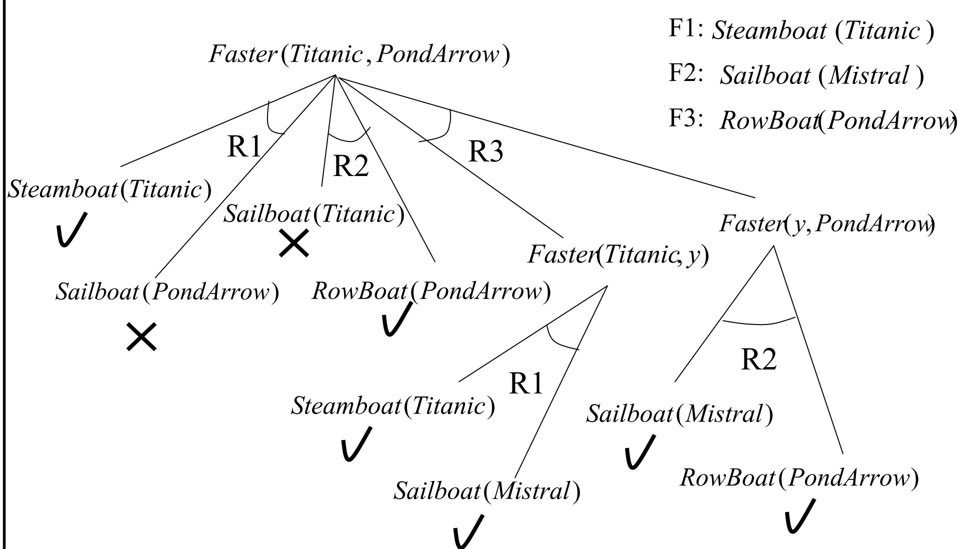
F3: $RowBoat(PondArrow)$

$$Sailboat(y) \wedge RowBoat(z) \Rightarrow Faster(y, z)$$

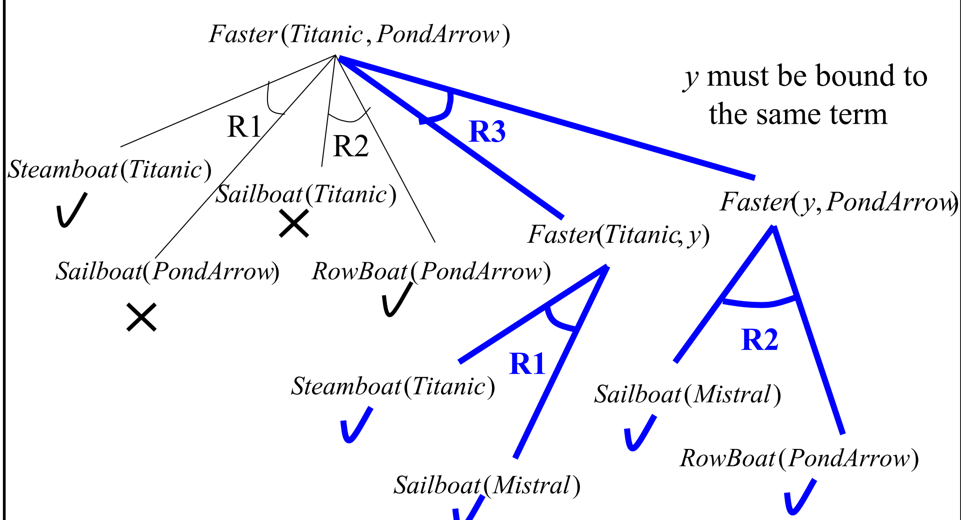
$$Faster(Titanic, PondArrow)$$

$$\{y / Titanic, z / PondArrow\}$$

Backward chaining example



Backward chaining



Knowledge-based system

Knowledge base
Inference engine

- **Knowledge base:**
  - A set of sentences that describe the world in some formal (representational) language (e.g. first-order logic)
  - Domain specific knowledge
- **Inference engine:**
  - A set of procedures that work upon the representational language and can infer new facts or answer KB queries (e.g. resolution algorithm, forward chaining)
  - Domain independent

Automated reasoning systems

Examples and main differences:

- **Theorem provers**
  - Prove sentences in the first-order logic. Use inference rules, resolution rule and resolution refutation.
- **Deductive retrieval systems**
  - Systems based on rules (KBs in Horn form)
  - Prove theorems or infer new assertions (forward, backward chaining)
- **Production systems**
  - Systems based on rules with actions in antecedents
  - Forward chaining mode of operation
- **Semantic networks**
  - Graphical representation of the world, objects are nodes in the graphs, relations are various links

Production systems

Based on rules, but different from KBs in the Horn form

Knowledge base is divided into:

- **A Rule base (includes rules)**
- **A Working memory (includes facts)**

A special type of if – then rule

$$p_1 \wedge p_2 \wedge \dots p_n \Rightarrow a_1, a_2, \dots, a_k$$

- **Antecedent:** a conjunction of literals
  - facts, statements in predicate logic
- **Consequent:** a conjunction of actions. An action can:
  - **ADD** the fact to the KB (working memory)
  - **REMOVE** the fact from the KB (consistent with logic ?)
  - **QUERY** the user, etc …

Production systems

Based on rules, but different from KBs in the Horn form

Knowledge base is divided into:

- **A Rule base (includes rules)**
- **A Working memory (includes facts)**

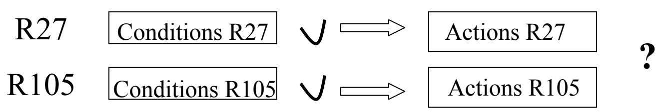
A special type of if – then rule

$$p_1 \wedge p_2 \wedge \dots p_n \Rightarrow a_1, a_2, \dots, a_k$$

- **Antecedent:** a conjunction of literals
  - facts, statements in predicate logic
- **Consequent:** a conjunction of actions. An action can:
  - **ADD** the fact to the KB (working memory)
  - **REMOVE** the fact from the KB ← !!! Different from logic
  - **QUERY** the user, etc …

Production systems

- Use **forward chaining to do reasoning**:
  - If the antecedent of the rule is satisfied (rule is said to be "active") then its consequent can be executed (it is "fired")
- **Problem:** Two or more rules are active at the same time. Which one to execute next?



- Strategy for selecting the rule to be fired from among possible candidates is called **conflict resolution**

Production systems

- Why is conflict resolution important? Or, why do we care about the order?
- Assume that we have two rules and the preconditions of both are satisfied:

R1: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete\ \ A(x)$

- What can happen if rules are triggered in different order?

Production systems

- Why is conflict resolution important? Or, Why do we care about the order?
- Assume that we have two rules and the preconditions of both are satisfied:

R1: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$

- What can happen if rules are triggered in different order?
  - If R1 goes first, R2 condition is still satisfied and we infer D(x)
  - If R2 goes first we may never infer D(x)

Production systems

- **Problems with production systems:**
  - Additions and Deletions can change a set of active rules;
  - If a rule contains variables testing all instances in which the rule is active may require a large number of unifications.
  - Conditions of many rules may overlap, thus requiring to repeat the same unifications multiple times.
- **Solution: Rete algorithm**
  - gives more efficient solution for managing a set of active rules and performing unifications
  - Implemented in the system **OPS-5** (used to implement XCON – an expert system for configuration of DEC computers)

Rete algorithm

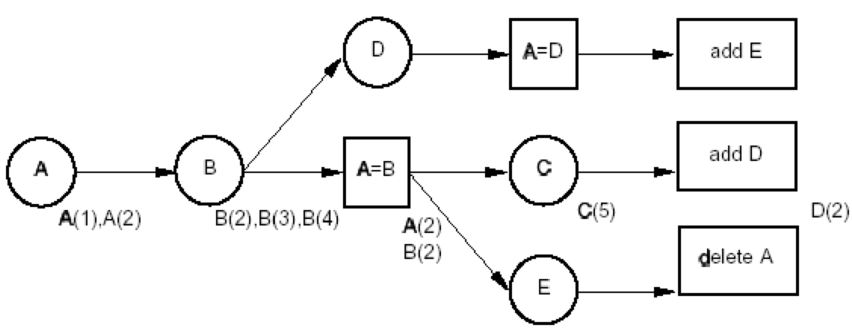
- Assume a set of rules:

$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$$
$$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$$
$$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$$

- And facts: $A(1), A(2), B(2), B(3), B(4), C(5)$

- **Rete:**
  - Compiles the rules to a network that merges conditions of multiple rules together (avoid repeats)
  - Propagates valid unifications
  - Reevaluates only changed conditions

Rete algorithm. Network.



Rules:

$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$$
$$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$$
$$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$$

Facts: $A(1), A(2), B(2), B(3), B(4), C(5)$

Conflict resolution strategies

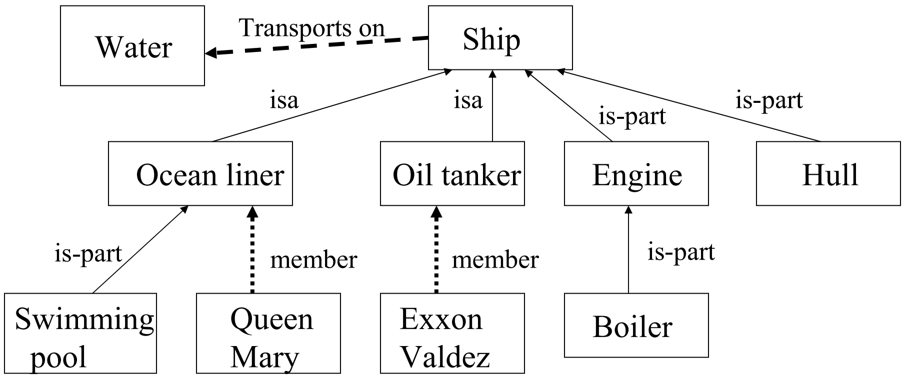
- **Problem:** Two or more rules are active at the same time. Which one to execute next?
- **Solutions:**
  - **No duplication** (do not execute the same rule twice)
  - **Recency.** Rules referring to facts newly added to the working memory take precedence
  - **Specificity.** Rules that are more specific are preferred.
  - **Priority levels.** Define priority of rules, actions based on expert opinion. Have multiple priority levels such that the higher priority rules fire first.

Semantic network systems

- Knowledge about the world described in terms of graphs. Nodes correspond to:
  - **Concepts or objects** in the domain.

  Links to relations. Three kinds:
  - **Subset links** (isa, part-of links) } Inheritance relation links
  - **Member links** (instance links) }
  - **Function links**.
- Can be transformed to the first-order logic language
- Graphical representation is often easier to work with
  - better overall view on individual concepts and relations

Semantic network. Example.



Inferred properties: *Queen Mary is a ship*
Queen Mary has a boiler

Planning: situation calculus

Representation of actions, situations, events

The world is dynamic:

- What is true now may not be true tomorrow
- Changes in the world may be triggered by our activities

Problems:

- Logic (FOL) as we had it referred to a static world. How to represent the change in the FOL ?
- How to represent actions we can use to change the world?

Planning

Planning problem:

- find a sequence of actions that achieves some goal
- an instance of a search problem
- the state description is typically very complex and relies on a logic-based representation

Methods for modeling and solving a planning problem:

- State space search
- Situation calculus based on FOL
- STRIPS – state space search algorithm
- Partial-order planning algorithms

Situation calculus

Provides a framework for representing change, actions and for reasoning about them

- **Situation calculus**
  - based on the first-order logic,
  - a situation variable models new states of the world
  - action objects model activities
  - uses inference methods developed for FOL to do the reasoning

Situation calculus

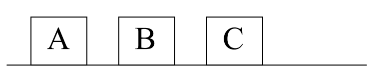
- Logic for reasoning about changes in the state of the world
- **The world is described by:**
  - Sequences of **situations** of the current state
  - Changes from one situation to another are **caused by actions**
- **The situation calculus allows us to:**
  - Describe the **initial state and a goal state**
  - Build the **KB that describes the effect of actions** (operators)
  - Prove that the KB and the initial state lead to a goal state
    - extracts a plan as side-effect of the proof

Situation calculus

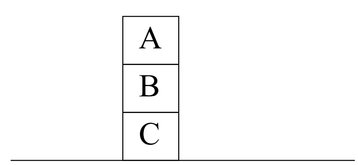
The language is based on the First-order logic plus:

- **Special variables:** *s,a* – objects of type situation and action
- **Action functions:** return actions.
  - E.g. *Move(A, TABLE, B)* represents a move action
  - *Move(x,y,z)* represents an action schema
- **Two special function symbols of type situation**
  - s_0 – initial situation
  - *DO(a,s)* – denotes the situation obtained after performing an action a in situation s
- **Situation-dependent functions and relations**
  (also called **fluents**)
  - **Relation:** *On(x,y,s)* – object x is on object y in situation s;
  - **Function:** Above(x,s) – object that is above x in situation s.

Situation calculus. Blocks world example.



Initial state



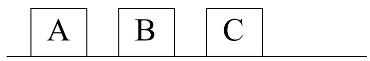
Goal

On(A,Table, s_0*)*
On(B,Table, s_0*)*
On(C,Table, s_0*)*
Clear(A, s_0*)*
Clear(B, s_0*)*
Clear(C, s_0*)*
Clear(Table, s_0*)*

Find a state (situation) s, such that

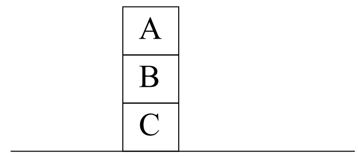
On(A,B, s)
On(B,C, s)
On(C,Table, s)

Blocks world example.



Initial state

On(A,Table, s_0*)*
On(B,Table, s_0*)*
On(C,Table, s_0*)*
Clear(A, s_0*)*
Clear(B, s_0*)*
Clear(C, s_0*)*
Clear(Table, s_0*)*



Goal

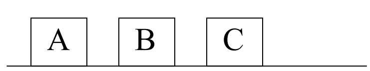
On(A,B, s)
On(B,C, s)
On(C,Table, s)

Note: It is not necessary that the goal describes all relations

Clear(A, s)

Blocks world example.

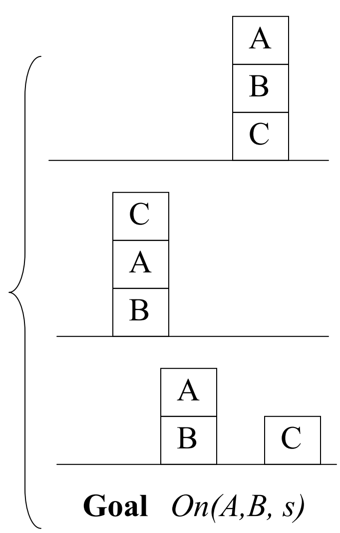
Assume a simpler goal *On(A,B, s)*



Initial state

On(A,Table, s_0*)*
On(B,Table, s_0*)*
On(C,Table, s_0*)*
Clear(A, s_0*)*
Clear(B, s_0*)*
Clear(C, s_0*)*
Clear(Table, s_0*)*

3 possible goal configurations



Goal *On(A,B, s)*

Knowledge base: Axioms.

Knowledge base needed to support the reasoning:

- Must represent changes in the world due to actions.

Two types of axioms:

- **Effect axioms**
  - changes in situations that result from actions
- **Frame axioms**
  - things preserved from the previous situation

Blocks world example. Effect axioms.

Effect axioms:

Moving x from y to z. $MOVE(x, y, z)$

Effect of move changes on **On** relations

$$On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \rightarrow On(x, z, DO(MOVE(x, y, z), s))$$

$$On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \rightarrow \neg On(x, y, DO(MOVE(x, y, z), s))$$

Effect of move changes on **Clear** relations

$$On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \rightarrow Clear(y, DO(MOVE(x, y, z), s))$$

$$On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \wedge (z \neq Table) \rightarrow \neg Clear(z, DO(MOVE(x, y, z), s))$$

Blocks world example. Frame axioms.

- **Frame axioms.**
  - Represent things that remain unchanged after an action.

On relations:

$$On(u,v,s) \wedge (u \neq x) \wedge (v \neq y) \rightarrow On(u,v,DO(MOVE(x,y,z),s))$$

Clear relations:

$$Clear(u,s) \wedge (u \neq z) \rightarrow Clear(u,DO(MOVE(x,y,z),s))$$

Planning in situation calculus

Planning problem:

- find a sequence of actions that lead to a goal

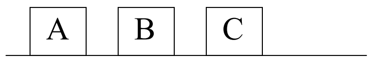
Planning in situation calculus is converted to the theorem proving problem

Goal state:

$$\exists s\ On(A,B,s) \wedge On(B,C,s) \wedge On(C,Table,s)$$

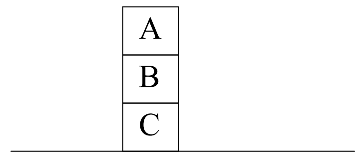
- Possible inference approaches:
  - **Inference rule approach**
  - **Conversion to SAT**
- **Plan** (solution) is a byproduct of theorem proving.
- **Example:** blocks world

Planning in a blocks world.



Initial state

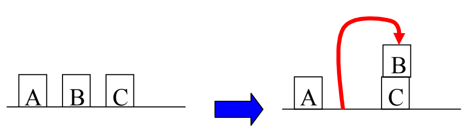
$On(A, Table, s_0)$
$On(B, Table, s_0)$
$On(C, Table, s_0)$
$Clear(A, s_0)$
$Clear(B, s_0)$
$Clear(C, s_0)$
$Clear(Table, s_0)$



Goal

$On(A, B, s)$
$On(B, C, s)$
$On(C, Table, s)$

Planning in the blocks world.



Initial state (s0)

s1

$s_0 =$

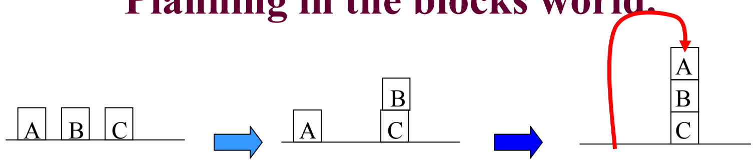
$On(A, Table, s_0)$ $Clear(A, s_0)$ $Clear(Table, s_0)$
$On(B, Table, s_0)$ $Clear(B, s_0)$
$On(C, Table, s_0)$ $Clear(C, s_0)$

Action: $MOVE(B, Table, C)$

$s_1 = DO(MOVE(B, Table, C), s_0)$

$On(A, Table, s_1)$ $Clear(A, s_1)$ $Clear(Table, s_1)$
$On(B, C, s_1)$ $Clear(B, s_1)$
$\neg On(B, Table, s_1)$ $\neg Clear(C, s_1)$
$On(C, Table, s_1)$

Planning in the blocks world.



Initial state (s0) **s1** **s2**

$$s_1 = DO(MOVE(B, Table, C), s_0)$$

$On(A, Table, s_1)$
$On(B, C, s_1)$
$\neg On(B, Table, s_1)$
$On(C, Table, s_1)$

$Clear(A, s_1)$
$Clear(B, s_1)$
$\neg Clear(C, s_1)$

$Clear(Table, s_1)$

Action: $MOVE(A, Table, B)$

$$s_2 = DO(MOVE(A, Table, B), s_1)$$
$$= DO(MOVE(A, Table, B), DO(MOVE(B, Table, C), s_0))$$

$On(A, B, s_2)$
$On(B, C, s_2)$
$On(C, Table, s_2)$

$\neg On(A, Table, s_2)$
$\neg On(B, Table, s_2)$
$Clear(A, s_2)$

$\neg Clear(B, s_2)$
$\neg Clear(C, s_2)$
$Clear(Table, s_2)$

Planning in situation calculus.

Planning problem:

- Find a sequence of actions that lead to a goal
- Is a special type of a search problem
- Planning in situation calculus is converted to theorem proving.

- **Problems:**
  - Large search space
  - Large number of axioms to be defined for one action
  - Proof may not lead to the best (shortest) plan.