In [1]:

```
```

Categories and Objects

The organization of objects into categories is a vital part of knowledge representation. Although interaction with the world takes place at the level of individual objects, much reasoning takes place at the level of categories.

There are two choices for representing categories in first-order logic: predicates and objects.

> We can use the predicate Basketball(b), or we can reify the category as an object, Basketballs. We could then say `Member(b,Basketballs)`, which we will abbreviate as $b \in Basketballs$, to say that b is a member of the category of basketballs.

We say `Subset(Basketballs,Balls)`, abbreviated as $Basketballs \subset Balls$, to say that Basketballs is a subcategory of Balls. We will use **subcategory, subclass, and subset** interchangeably.

Subcategory organize knowledge through **inheritance**. If we say that all instances of the category Food are edible, and if we assert that Fruit is a subclass of Food and Apples is a subclass of Fruit, then we can infer that every apple is edible. We say that the individual apples inherit the property of edibility, in this case from their membership in the Food category.

Subclass relations organize categories into a taxonomic hierarchy or taxonomy.

> First-order logic makes it easy to state facts about categories. Here are some example facts:

- An object is a member of a category.
  $BB9 \in Basketballs$
- A category is a subclass of another category.
  $Basketballs \subset Balls$

- All members of a category have some properties.
  $(x \in Basketballs) \Rightarrow Spherical(x)$
- Members of a category can be recognized by some properties.
  $Orange(x) \wedge Round(x) \wedge Diameter(x)=9.5" \wedge x \in Balls \Rightarrow x \in Basketballs$
- A category as a whole has some properties.
  $Dogs \in DomesticatedSpecies$

We say that two or more categories are **disjoint** if they have no members in common. We say that the classes undergrad and graduate student form an **exhaustive decomposition** of university students. A exhaustive decomposition of disjoint sets is known as a **partition**.

> Examples of these three concepts:

$Disjoint(\{Animals,Vegetablesg\})$
$ExhaustiveDecomposition(\{Americans,Canadians,Mexicans\}, NorthAmericans)$
$Partition(\{Animals, Plants, Fungi, Protista, Monera\}, LivingThings)$

(***Note that the ExhaustiveDecomposition of NorthAmericans is not a Partition, because some people have dual citizenship.***)

Categories can also be defined by providing necessary and sufficient conditions for membership.

> For example, a bachelor is an unmarried adult male:

$x \in Bachelors \Leftrightarrow Unmarried(x) \wedge x \in Adults \wedge x \in Males$.

Physical composition

We use the general $PartOf$ relation to say that one thing is part of another. Objects can be grouped into PartOf hierarchies, similar to Subset hierarchy:

$PartOf(Bucharest,Romania)$
$PartOf(Romania,EasternEurope)$
$PartOf(EasternEurope,Europe)$

PartOf(Europe,Earth)

The PartOf relation is transitive and reflexive; that is,

$$PartOf(x,y) \land PartOf(y,z) \Rightarrow PartOf(x,z)$$
$$PartOf(x,x)$$

Categories of composite objects are often characterized by structural relations among parts.

For example, a biped is an object with exactly two legs attached to a body:

$$Biped(a) \Rightarrow \exists\ l1,l2,b\ Leg(l1) \land Leg(l2) \land Body(b) \land PartOf(l1,a) \land PartOf(l2,a) \land PartOf(b,a) \land Attached(l1,b) \land Attached(l2,b) \land l1 \neq l2 \land [\forall\ l3\ Leg(l3) \land PartOf(l3,a) \Rightarrow (l3=l1 \lor l3=l2)]$$

Measures

$$Diameter(Basketball12)=Inches(9.5) \qquad ListPrice(Basketball12)=\$(19)$$
$$Weight(BunchOf(\{Apple1,Apple2,Apple3\})) = Pounds(2)$$
$$d \in Days \Rightarrow Duration(d)=Hours(24)$$

Intrinsic Vs Extrinsic properties

Any part of a butter-object is also a butter-object: $b \in Butter\ wedge\ PartOf(p,b) \Rightarrow p \in Butter$ Butter melts at around 30 degrees centigrade: $b \in Butter \Rightarrow MeltingPoint(b,Centigrade(30))$

Some properties are intrinsic: they belong to the very substance of the object, rather than to the object as a whole.

When you cut an instance of stuff in half, the two pieces retain the intrinsic properties—things like density, boiling point, flavor, Extrinsic color, ownership, and so on.

On the other hand, their extrinsic properties—weight, length, shape, and so on—are not retained under subdivision.

A category of objects that includes in its definition only intrinsic properties is then a **substance**, or **mass noun**; a class that includes any extrinsic properties in its definition is a **count noun**.

Stuff and **Thing** are the most general substance and object categories, respectively.

Events

Actions: things that happen, such as Shoot; and fluents: aspects of the world that change, such as HaveArrow come under the category Events.

Example predicates	from the event calculus
T(f,t1,t2)	Fluent f is true for all times between t1 and t2
Happens(e,t1,t2)	Event e starts at time t1 and ends at t2
Initiates(e,f,t)	Event e causes fluent f to become true at time t
Terminates(e,f,t)	Event e causes fluent f to cease to be true at time t
Initiated(f,t1,t2)	Fluent f become true at some point between t1 and t2
Terminated(f,t1,t2)	Fluent f cease to be true at some point between t1 and t2
t1 < t2	Time point t1 occurs before time t2

Assume an event happens between time t1 and t3, and at t2 somewhere in that time interval the event changes the value of fluent f , either initiating it (making it true) or terminating it (making it false). Then at time t4 in the future, if no other intervening event has changed the fluent (either terminated or initiated it, respectively), then the fluent will have maintained its value.

Formally, the axioms are:

Happens(e,t1,t3) ∧ Initiates(e,f,t2) ∧ Terminated(f, t2,t4) ∧ $t1 \leq t2 \leq t3 \leq t4$ ⇒ T(f,t2,t4)

Happens(e,t1,t3) ∧ Terminates(e,f,t2) ∧ ¬ Initiated(f,t2, t4) ∧ $t1 \leq t2 \leq t3 \leq t4$ ⇒ ¬ T(f,t2,t4)

where *Terminated* and *Initiated* are defined by:

$$Terminated(f,t_1,t_5) \Leftrightarrow \exists e,t_2,t_3,t_4\ Happens(e,t_2,t_4) \wedge Terminates(e,f,t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

$$Initiated(f,t_1,t_5) \Leftrightarrow \exists e,t_2,t_3,t_4\ Happens(e,t_2,t_4) \wedge Initiates(e,f,t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

Time

There are two kinds of time intervals: moments and extended intervals. The distinction is that only moments have zero duration:

Partition({Moments, ExtendedIntervals}, Intervals)

i ∈ Moments ⇔ Duration(i) = Seconds(0)

Semantic Networks

- Semantic network is a knowledge representation model which is in a form of graphical schemes consisting of nodes and links among nodes.
- Nodes in a semantic network can show concepts, objects, features, events, time.
- Links indicates the connection among nodes. The links should be labeled and directed.

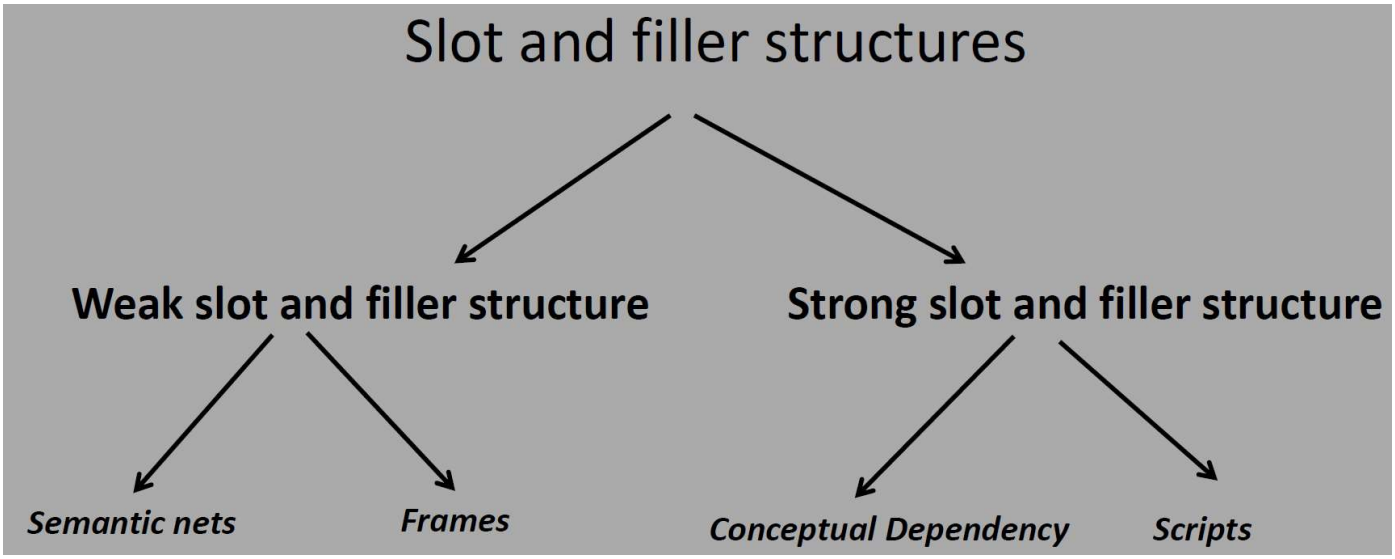
where *Terminated* and *Initiated* are defined by:

$$Terminated(f,t_1,t_5) \Leftrightarrow \exists e,t_2,t_3,t_4\ Happens(e,t_2,t_4) \wedge Terminates(e,f,t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

$$Initiated(f,t_1,t_5) \Leftrightarrow \exists e,t_2,t_3,t_4\ Happens(e,t_2,t_4) \wedge Initiates(e,f,t_3) \wedge t_1 \leq t_2 \leq t_3 \leq t_4 \leq t_5$$

Weak slot and filler structures

They are “Knowledge- Poor” or “weak” as very little importance is given to the specific knowledge the structure should contain. Here **attribute means slot and its value is called filler**.

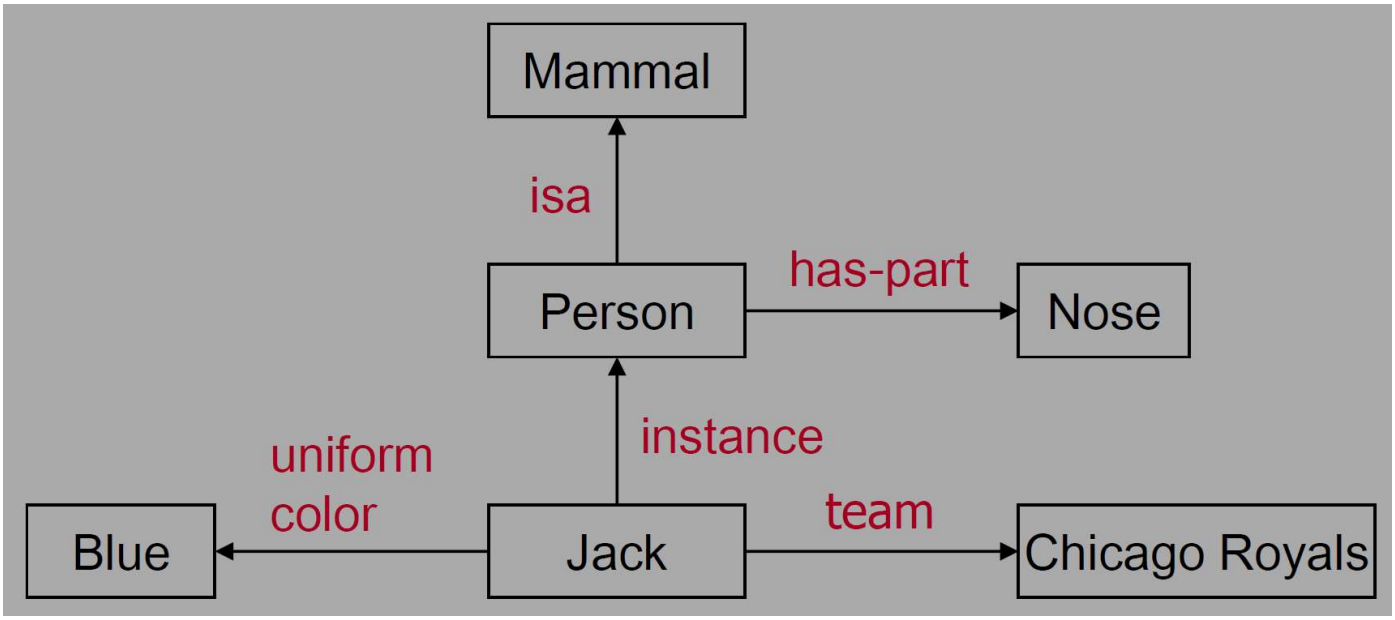


Semantic nets

In semantic nets information is represented as set of nodes connected to each other by a set of labelled arcs.

Nodes represent: various objects / values of the attributes of object.

Arcs represent: relationships among nodes.



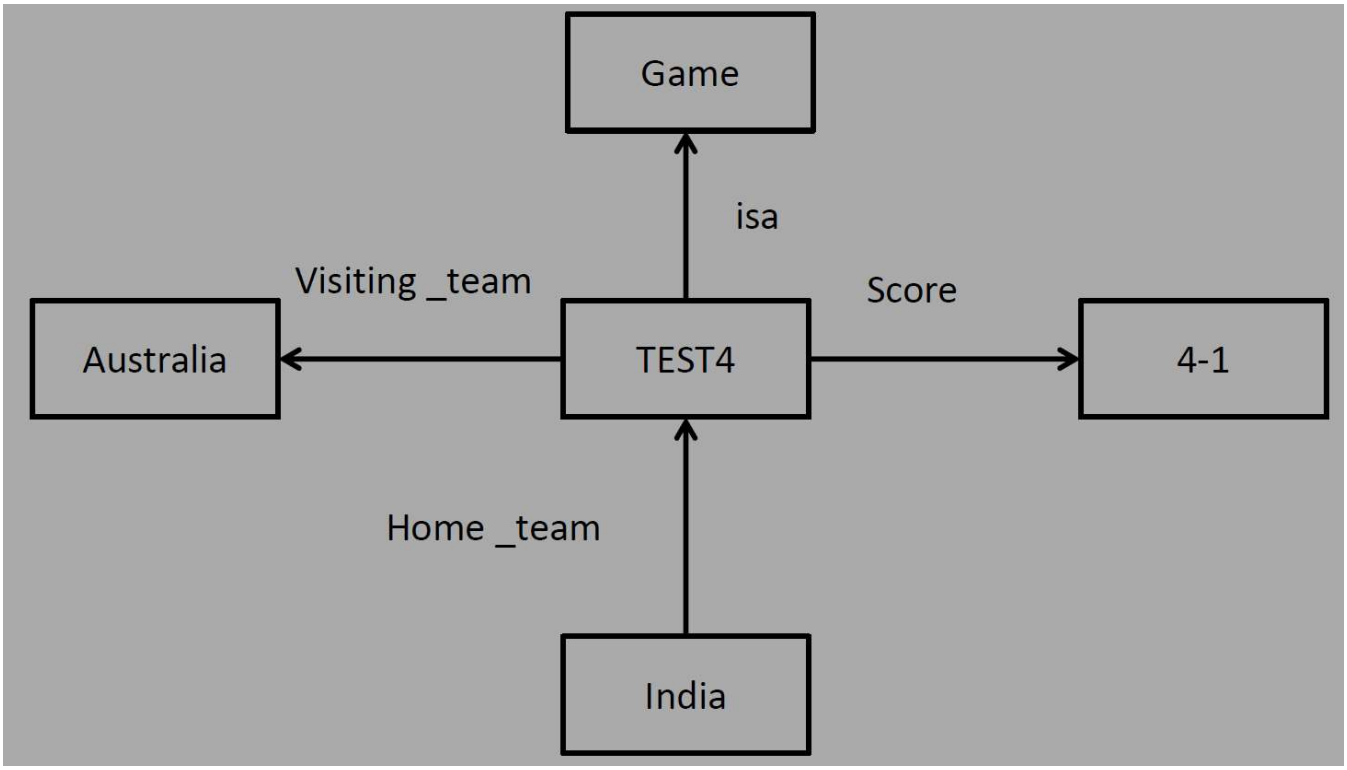
Representing non binary predicates:

1. Unary

> `Man(marcus)` can be converted into: `instance(marcus,Man)`

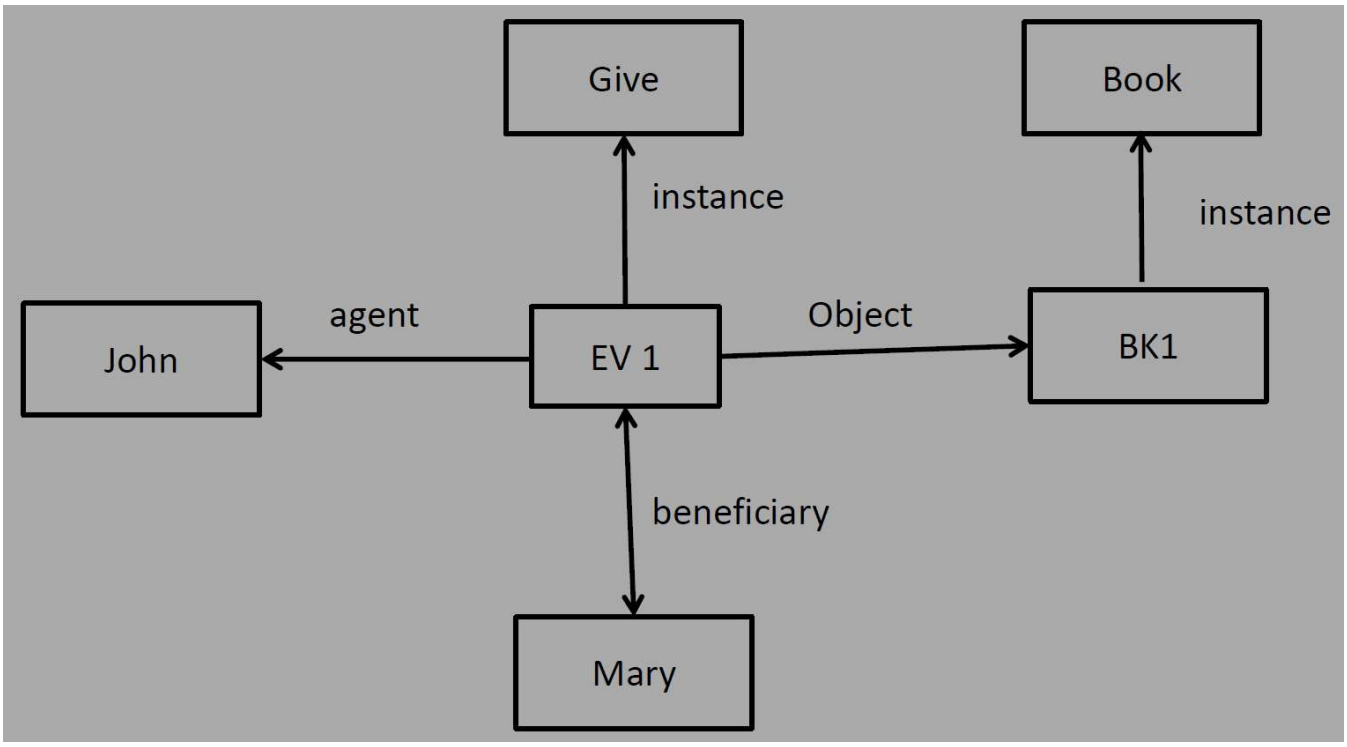
2. Other aritiese. Three or more place predicates can be converted to binary form as follows:
   - Create new object representing the entire predicate.
   - Introduce binary predicates to describe relation to this new object.

`Score(india,australia,4-1)`

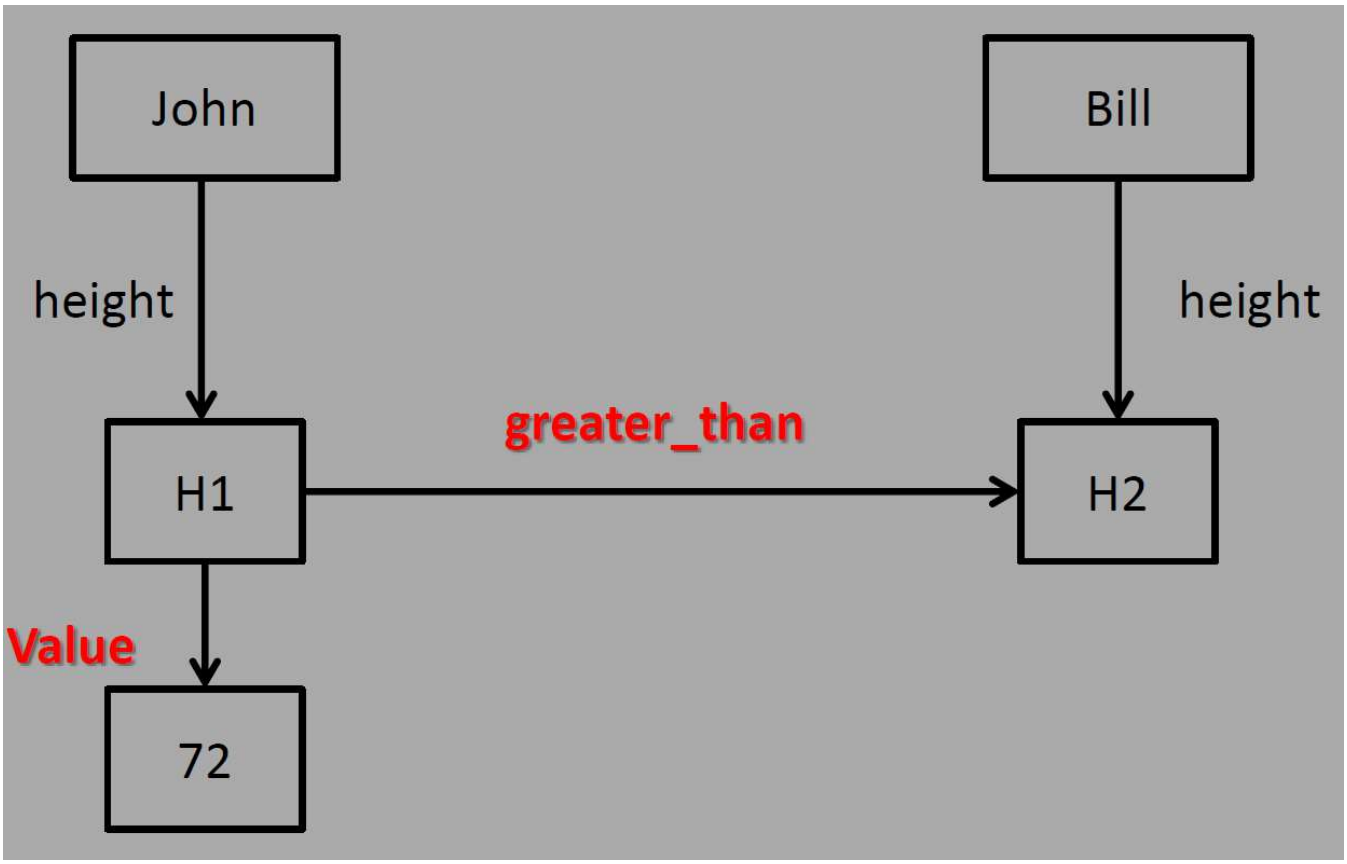


John gave the book to Mary :- `give(john,mary,book)`

John gave the book to Mary :- `give(john,mary,book)`



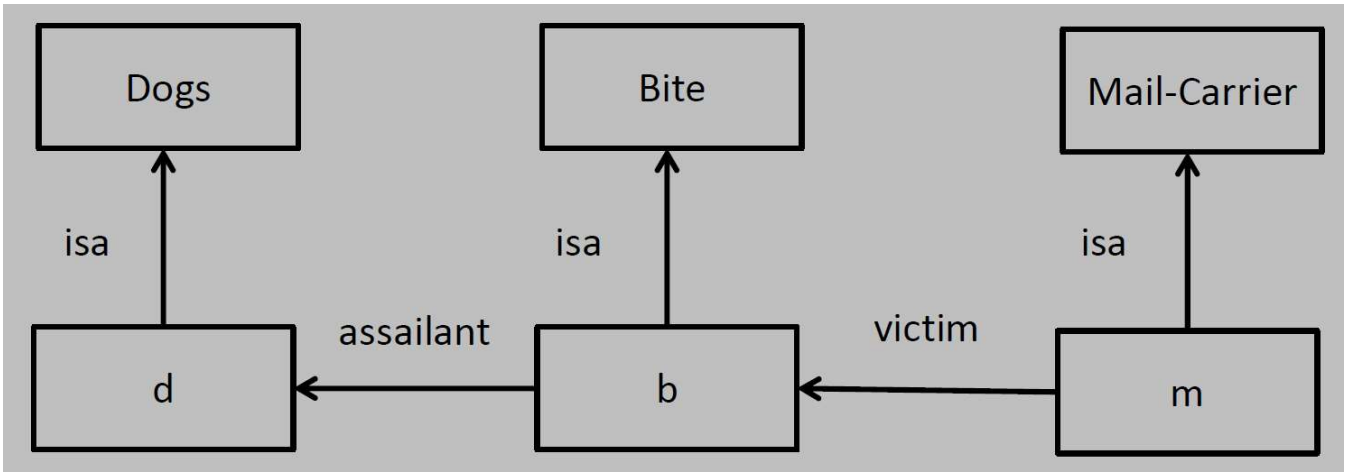
John is taller than Bill :- `taller(john,Bill)`



Partitioned semantic nets

Used to represent quantified expressions in semantic nets. One way to do this is to partition the semantic net into a hierarchical set of spaces each of which corresponds to the scope of one or more variable.

The dog bit the mail carrier (partitioning not required)



> Every dog has bitten a mail carrier

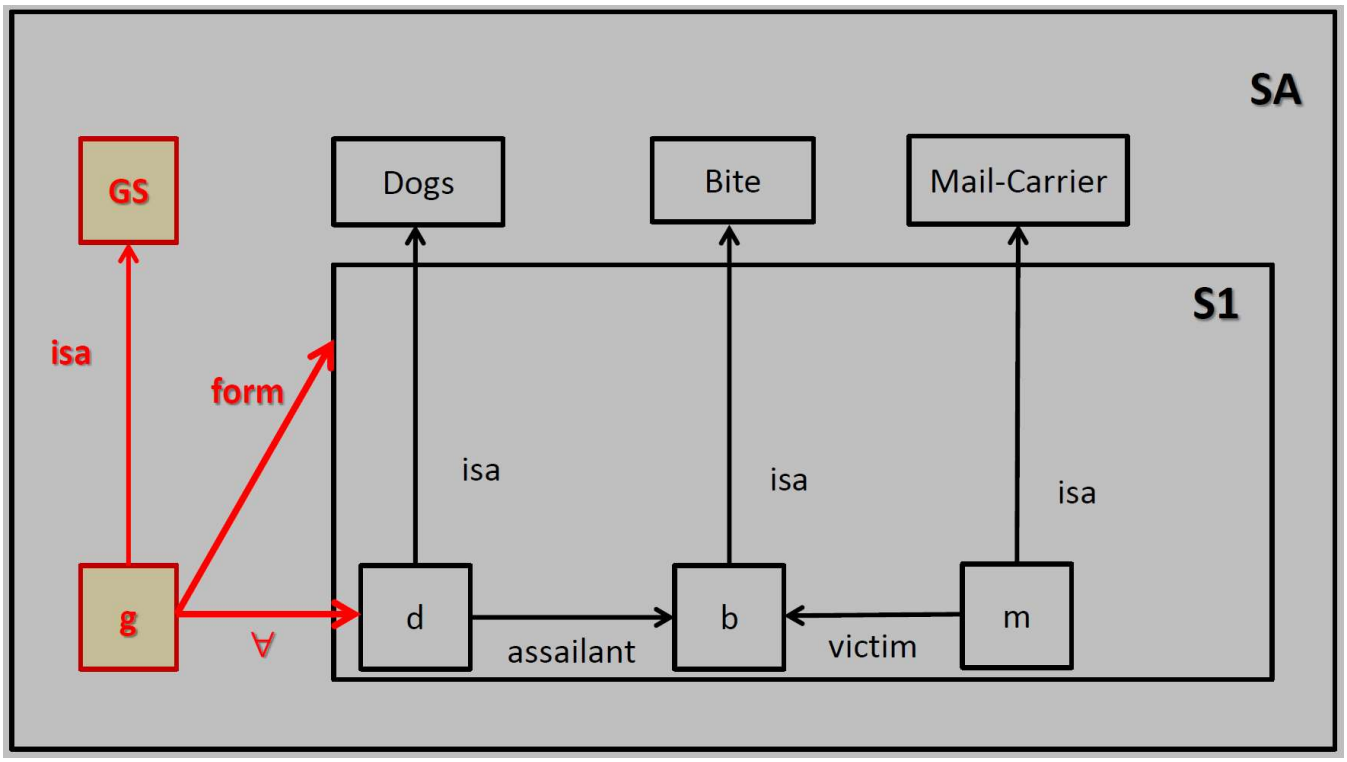
$$\forall x : dog(x) \Rightarrow \exists y : mail - carrier(y) \wedge bite(x, y)$$

How to represent universal quantifiers?

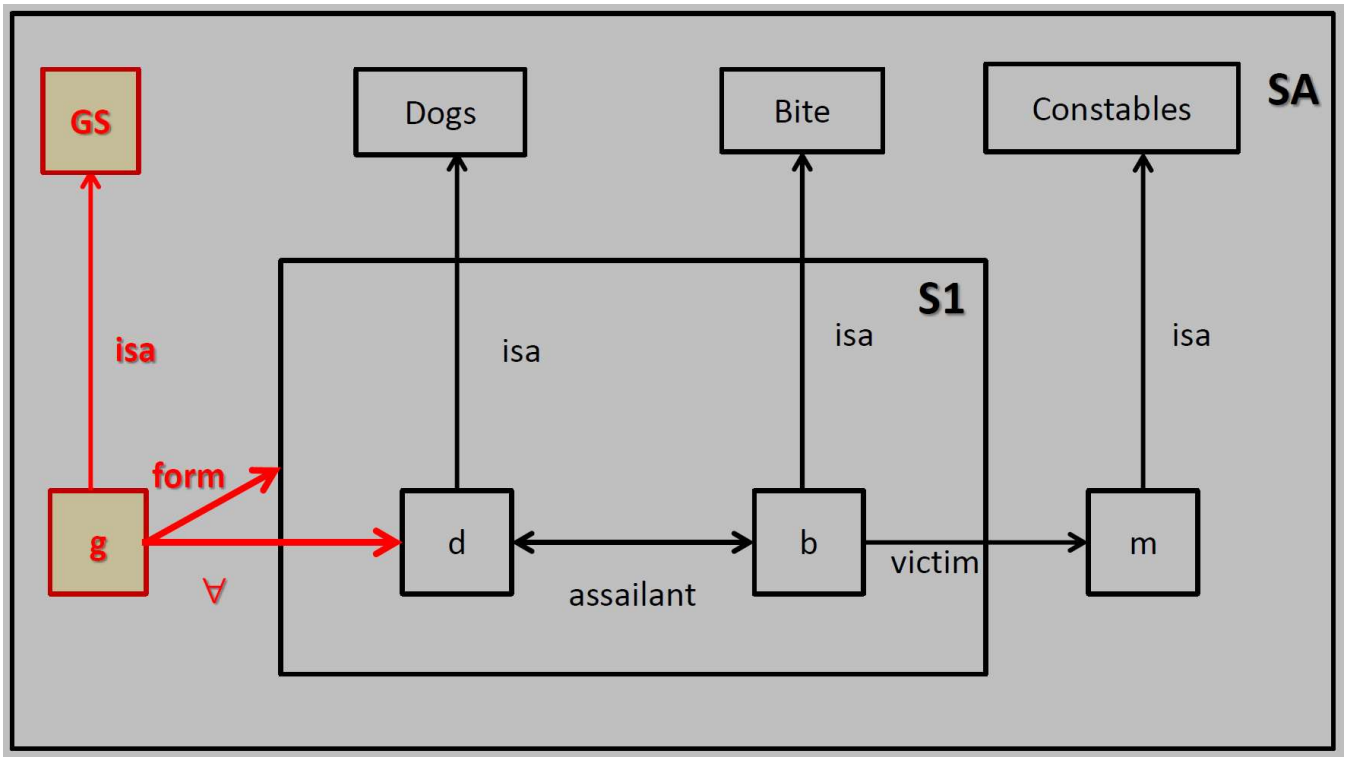
- Let node **g** stands for assertion given above
- This node is an instance of a special class **GS** of general statements about the world.
- Every element in **GS** has 2 attributes:-
  - Form - states relation that is being asserted.
  - ∀ connections - one or more, one for each of the universally quantified variables.
- **SA** is the space of partitioned sementic net.

Every dog has bitten a mail carrier

Every dog has bitten a mail carrier

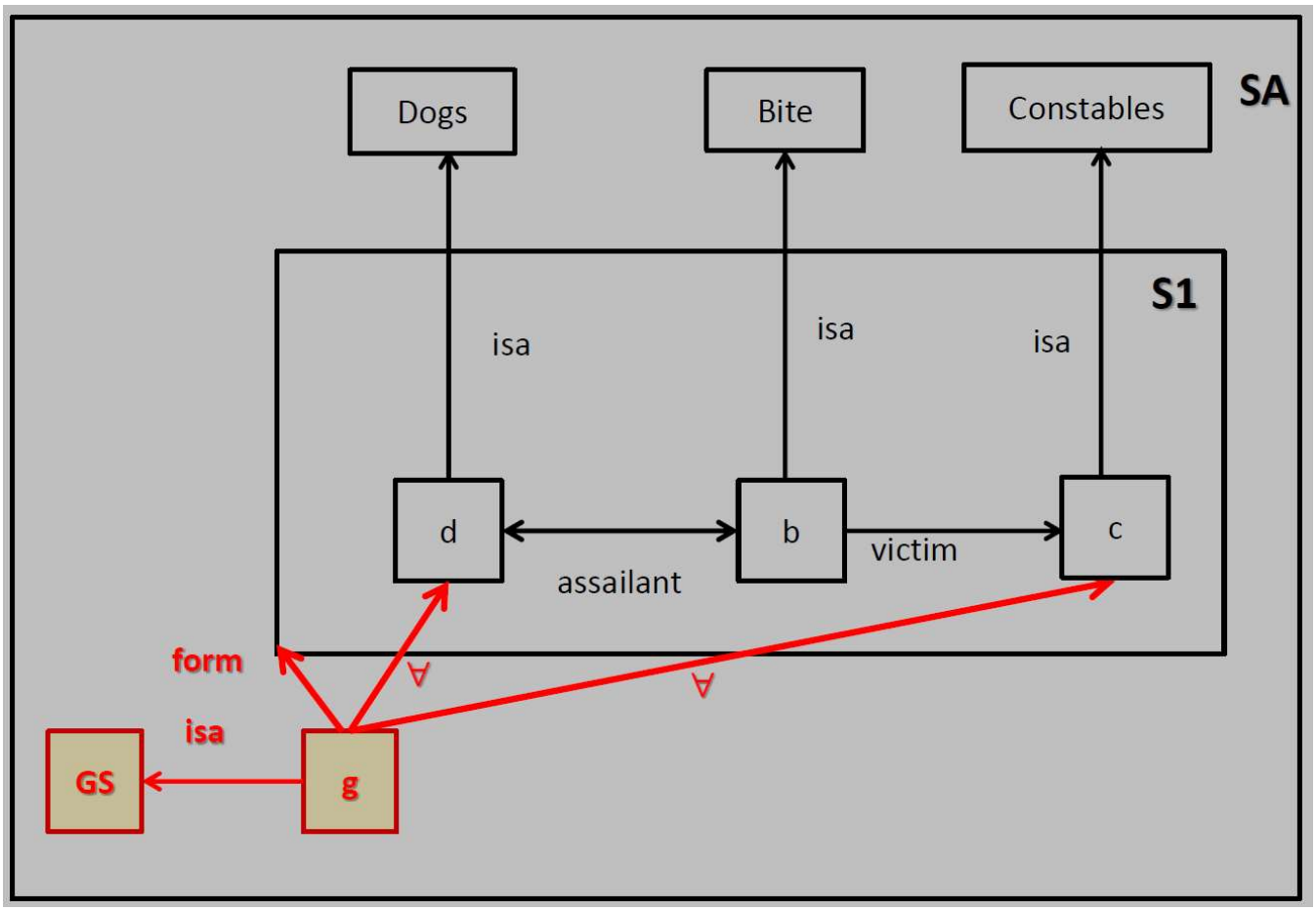


Every dog in the town has bitten the constable

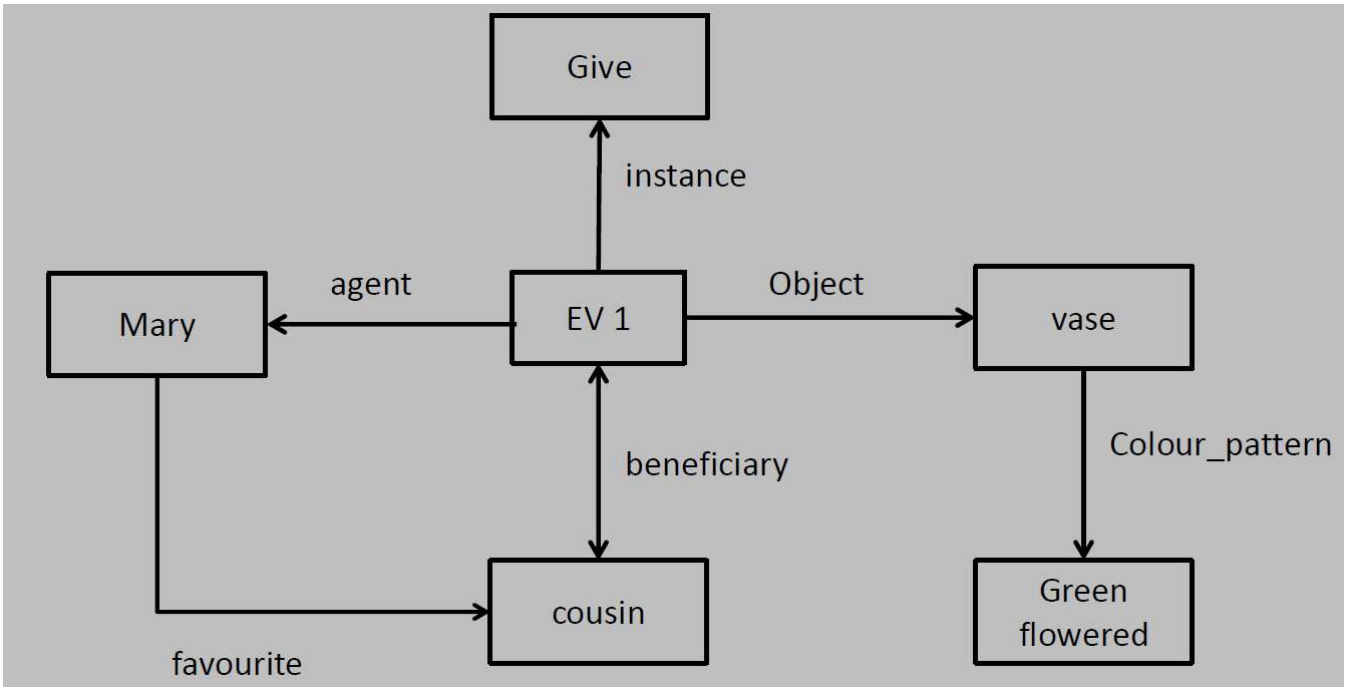


Every dog in the town has bitten every constable

Every dog in the town has bitten every constable

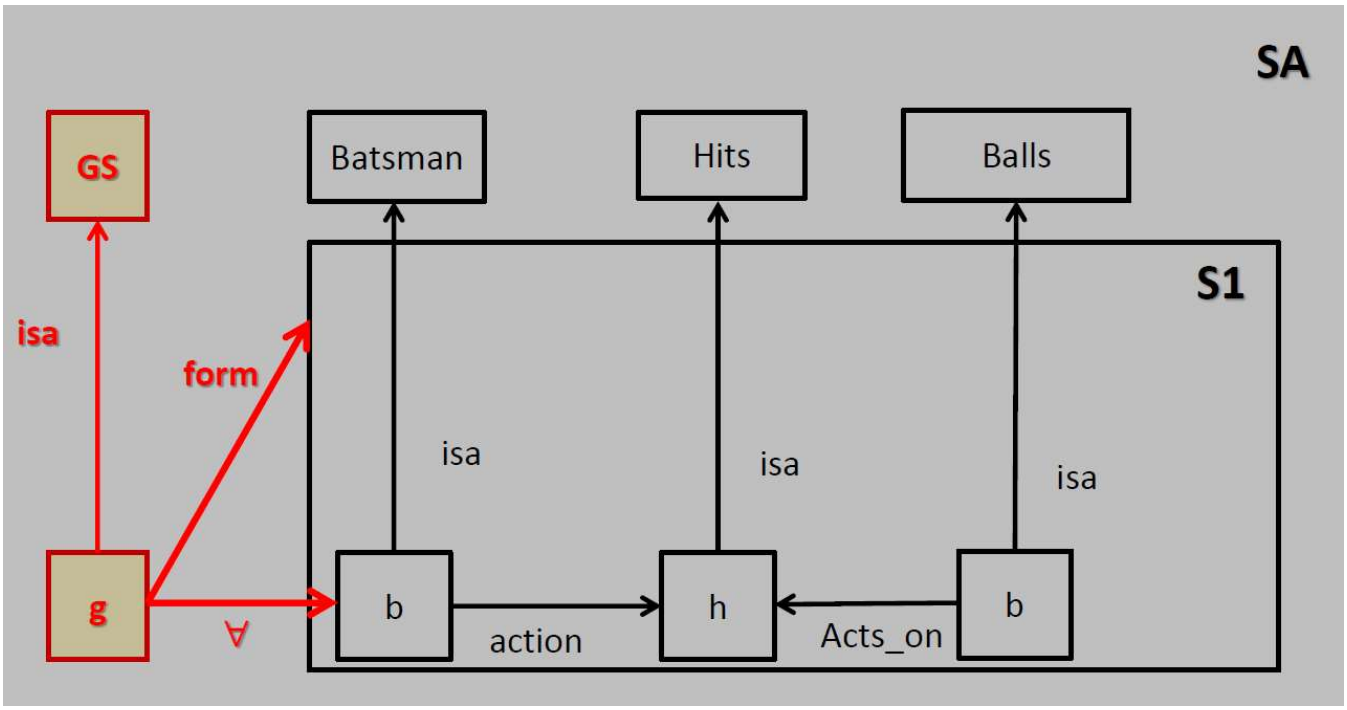


Mary gave the green flowered vase to her favourite cousin

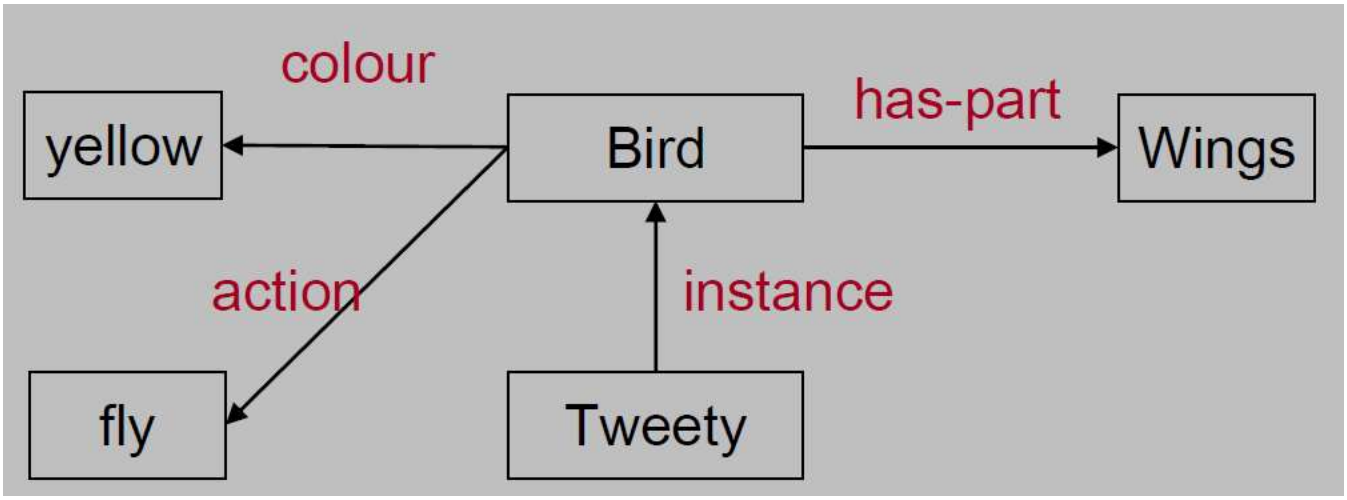


Every batsman hits a ball

Every batsman hits a ball



Tweety is a kind of bird who can fly. It is Yellow in colour and has wings



Frame

A frame is a collection of attributes (usually called slots) and associated values (and possibly constraints on values) that describe some entity in the world.

A single frame taken alone is rarely useful. Instead, we build frame systems out of collections of frames that are connected to each other by virtue of the fact that tbe value of an attribute of one frame may be another frame.

Each frame represents either a class (a set) or an instance (an element of a class).

The `isa` relation is used for subset relation. The set of adult males is a subset of the set of people. The set of major league baseball players is a subset of the set of adult males, and so forth.

The instance relation corresponds to the relation element-of. Pee Wee Reese is an element of the set of fielders.

Because a class represents a set, there are two kinds of attributes that can be associated with it. There are **attributes about the set itself, and there are attributes that are to be inherited by each element of the set. We indicate the difference between these two by prefixing the later with an asterisk (*)**.

> For example, consider the class ML-Baseball-Player, We have shown only two properties of it as a set: (it is a subset of the set of adult males. And it has cardinality 624 (i.e., there are 624 major league baseball players). We have listed five properties that all major league baseball players have (height, bats, batting-average, team, and uniform-color), and we have specified default values for the first three of them. By providing both kinds of slots, we allow a class both to define a set of objects and to describe a prototypical object of the set.

Person

isa: Mammal

cardinality: 6,000,000,000

* Handed: right

Adult__Male

isa: Person

Cardinality: 2,000,000,000

* Height: 5- 10

ML_Baseball_Player

isa: Adult_Male

cardinality: 624

* height: 6-1

* bats: equal to handed

* batting-avg: 0.252

* team:

*uniform_color:

Jack_Roberts

instance: Fielder

height: 5-10

balls: right

batting_avg: 0.309

team: Chicago cubs

uniform_color: blue

Individual frame

Fielder

isa: ML_Baseball_Player

cardinality: 376

batting_avg: 0.262

ML_Baseball_Team

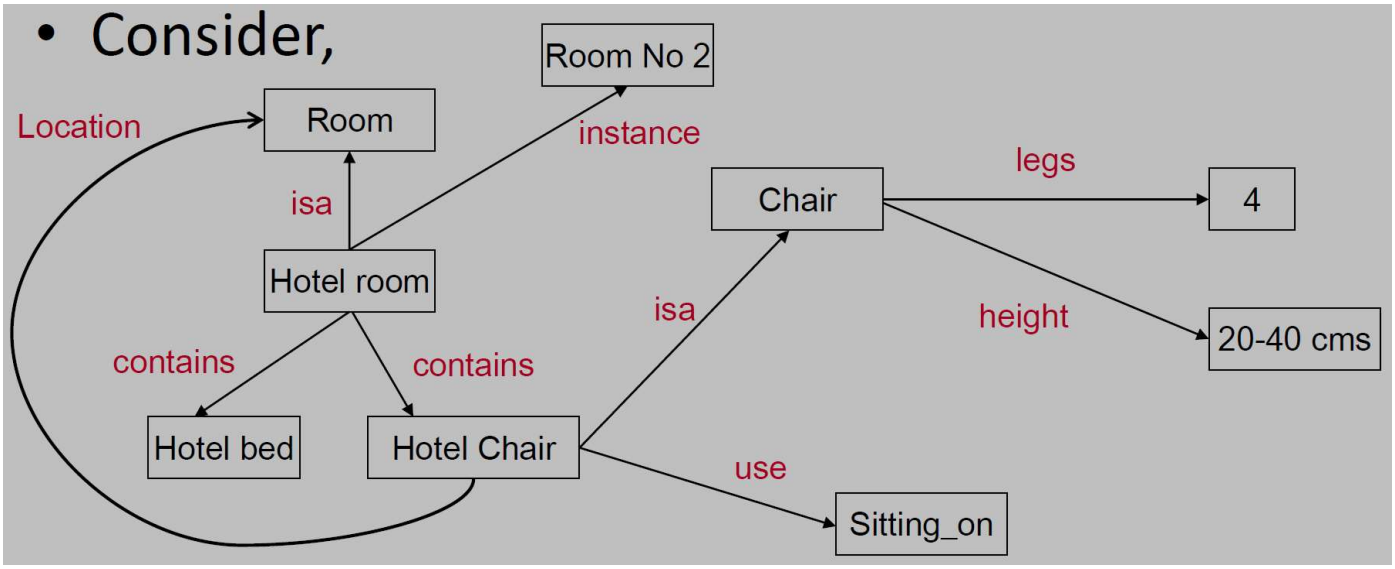
isa: Team

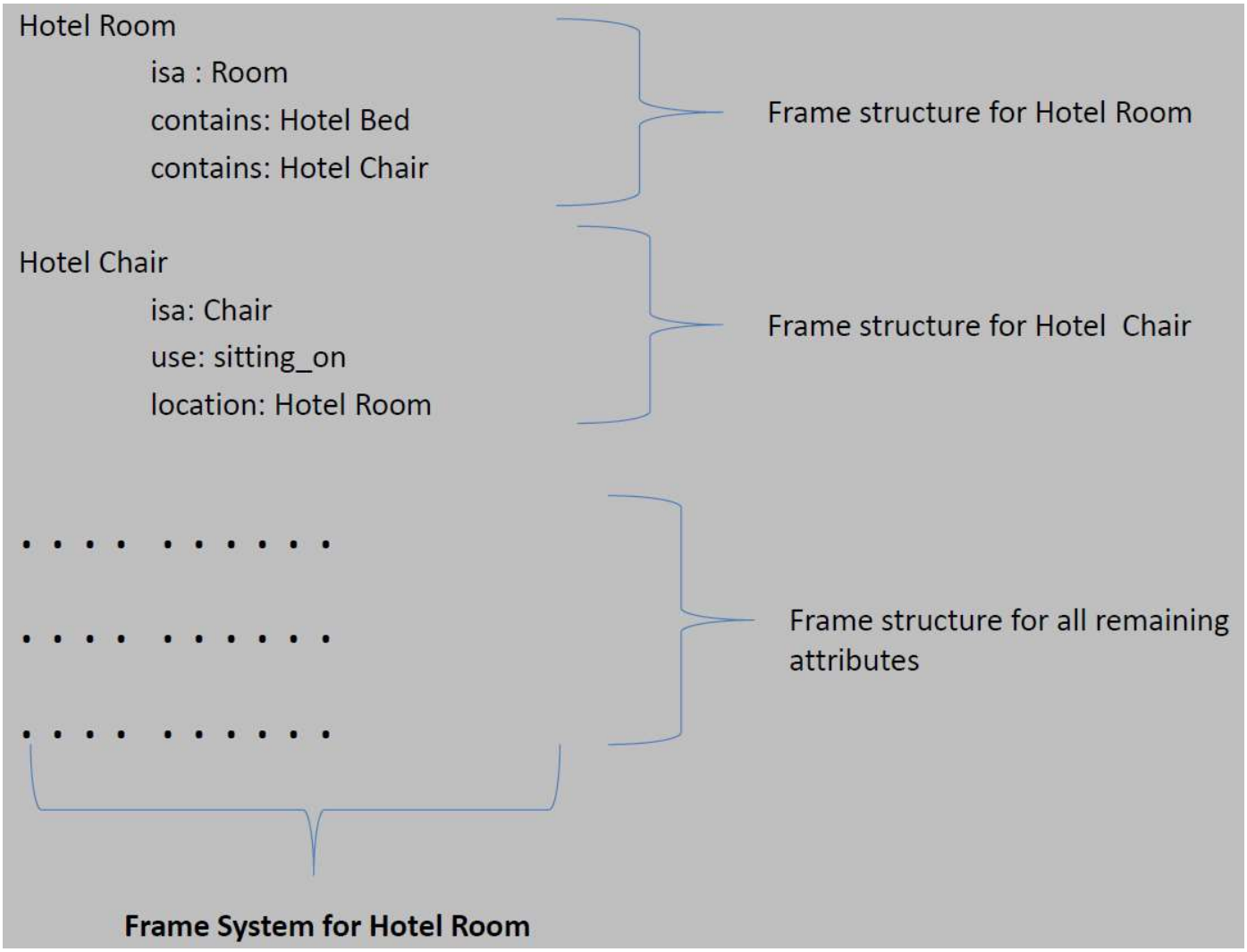
cardinality: 26

team_size: 24

manager:

Example 2





Class Vs Meta Class

A class can be viewed as a subset (isa) of a larger class that also contains its elements and an instance (instance) of a class of sets, from which it inherits its set-level properties.

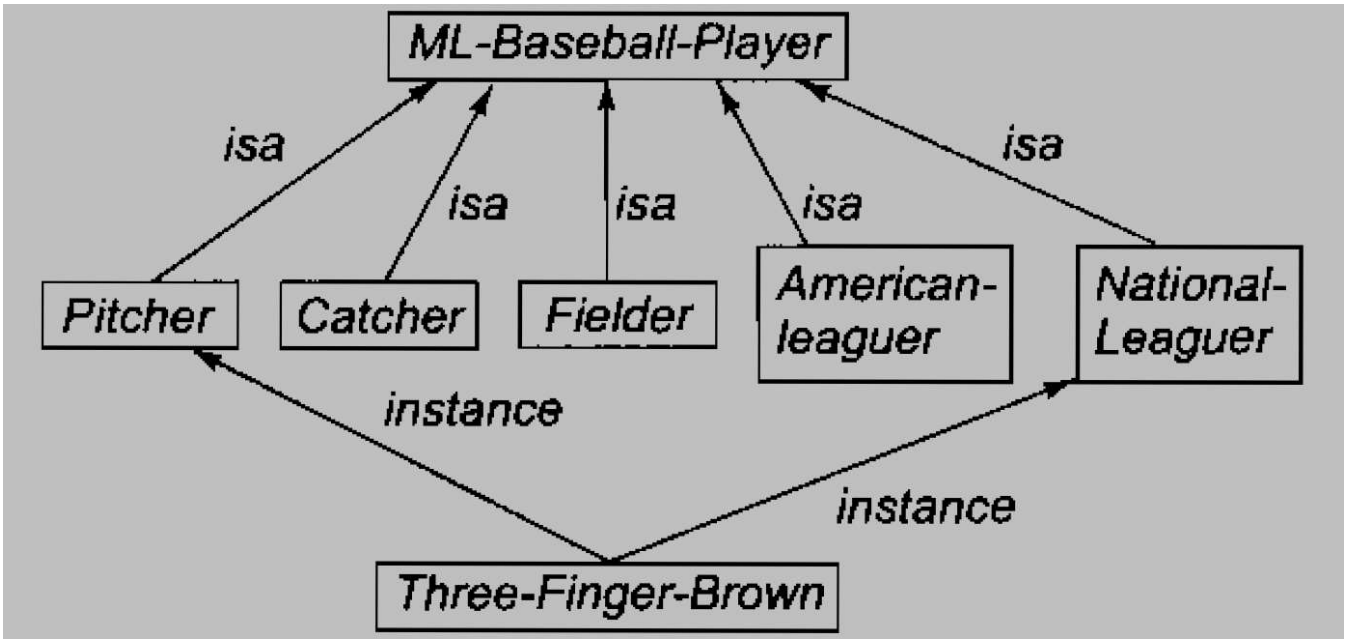
To make this distinction clear, it is useful to distinguish between regular classes, whose elements are individual entities, and meta classes, which are special classes whose elements are themselves classes.

A class is now an element of (instance) some class (or classes) as well as a subclass (isa) of one or more classes.

A class inherits properties from the class of which it is an instance, just as any instance does. In addition, a class passes inheritable properties down from its superclasses to its instances.

Is covered by: relationship is called as 'covered-by' when we have a class and it has set of subclasses, the union of which is equal to the superclass

Is covered by: relationship is called as 'covered-by' when we have a class and it has set of subclasses, the union of which is equal to the superclass



ML_Baseball_Player

is covered-by: { Pitcher, Catcher, Fielder,American leaguer, National leaguer}

Pitcher

isa: ML_Baseball_Player

mutually_disjoint-with: {Catcher, Fielder }

Catcher

isa: ML_Baseball_Player

mutually_disjoint-with: {Pitcher, Fielder }

Fielder

isa: ML_Baseball_Player

mutually_disjoint-with: {Pitcher, Catcher}

Conceptual Dependancy

- Focuses on concepts instead of syntax.
- Focuses on understanding instead of structure.
- Assumes inference is fundamental to understanding.

- Introduced idea of a canonical meaning representation.
  - different words and structures represent the same concept.
  - language-independent meaning representation.

> Canonical Meaning Representations

John gave Mary a book.

John gave a book to Mary.

Mary was given a book by John.

Mary took a book from John.

Mary received a book from John.

> **Conceptual Primitives**

- basic meaning elements that underlie the words that we use.
- can be combined to represent complex meanings.
- an interlingual representation.
  Goal: to represent meaning so that general rules can be applied, without duplicating information.
  Theory: a small number of primitive actions can represent any sentence.

> **Conceptual Primitives for Actions**

Eleven primitives can account for most actions in the physical world:

- ATRANS : to change an abstract relationship of a physical object.
- ATTEND : to direct a sense organ or focus an organ towards a stimulus.
- INGEST : to take something inside an animate object.
- EXPEL : to take something from inside an animate object and force it out.
- GRASP : to physically grasp an object
- MBUILD : to create or combine thoughts.
- MTRANS : to transfer information mentally.

- MOVE : to move a body part
- PROPEL : to apply a force to
- PTRANS : to change the location of a physical object.
- SPEAK : to produce a sound.

Conceptual Categories

- PP : Picture producers (PPs) denote physical objects serving in various roles in a conceptualization. In representing the sentence “Amy took a breath,”, “Amy” and “breath” (or “air”) are PPs. Actors must be an animate PP, or a natural force.
- ACT : ACTs are conceptual primitives representing things that can be done by an actor to an object, or events that can happen to an object. One of eleven primitive actions.
- LOC : Location.
- T : Time.
- AA (action aider) : modifications of features of an ACT.
  e.g., speed factor in PROPEL.
- PA : attributes of an object, of the form STATE(VALUE).
  e.g., COLOR(red).

Conceptual Roles

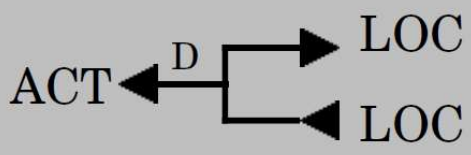
- Conceptualization: The basic unit of the conceptual level of understanding.
- Actor: The performer of an ACT.
- ACT: An action done to an object.
- Object: A thing that is acted upon.
- Recipient: The reciever of an object as the result of an ACT.
- Direction: The location that an ACT is directed toward.
- State: The state that an object is in.

Conceptual Syntax Rules

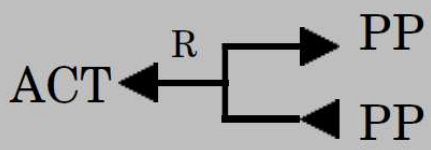
PPs can perform actions.

PPs can be described by an attribute.

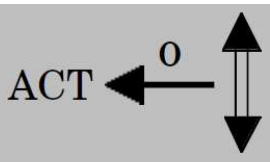
ACTs can have objects.



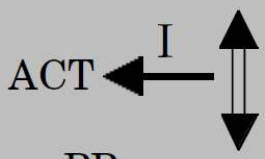
ACTs can have directions.



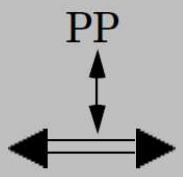
ACTs can have recipients.



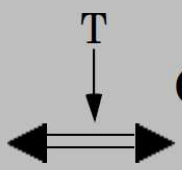
Objects can be conceptualizations.



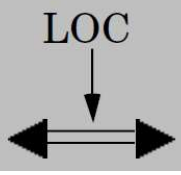
Instruments can be conceptualizations.



PPs can be described by conceptualizations.

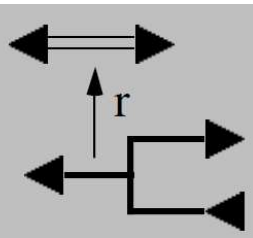


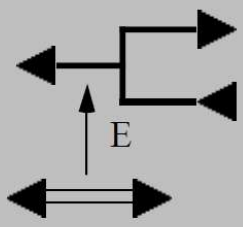
Conceptualizations can have times.



Conceptualizations can have locations.

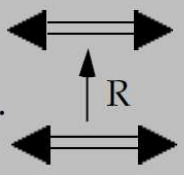
Conceptualizations can result in state changes.





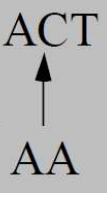
States or state changes can enable conceptualizations to occur.

Mental ACTs can serve as reasons for conceptualizations.



PP ⟷ PP One PP is equivalent to another.

ACTs can be varied along certain dimensions.



Conceptual Tenses

past	p
future	f
negation	/
start of a transition	ts
end of a transition	tf
conditional	c
continuous	k
interrogative	?
timeless	∞
present	nil

States

states of objects are described by scales with numerical values.

HEALTH	(range -10 to 10)
dead	-10
gravely ill	-9
sick	-9 to -1
under the weather	-2
all right	0
tip top	+7
perfect health	+10

FEAR	(range -10 to 0)
terrified	-9
scared	-5
anxious	-2
calm	0

MENTAL STATE	(range -10 to +10)
catatonic	-9
depressed	-5
upset	-3
sad	-2
ok	0
pleased	+2
happy	+5
ecstatic	+10

CONSCIOUSNESS	(range 0 to +10)
unconscious	0
asleep	5
awake	10
"higher drug consciousness"	>10

Some words can be combinations of scales:

shocked = SURPRISE (6)
DISGUST (-5)

calm = SURPRISE (0)
DISGUST (0)
FEAR (0)
ANGER (0)
CONSCIOUSNESS (> 0)

... some states take absolute values

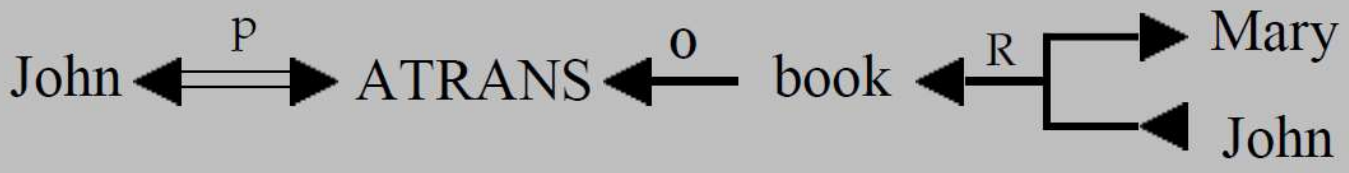
LENGTH
COLOR
MASS
SPEED

... some states are just relationships between objects

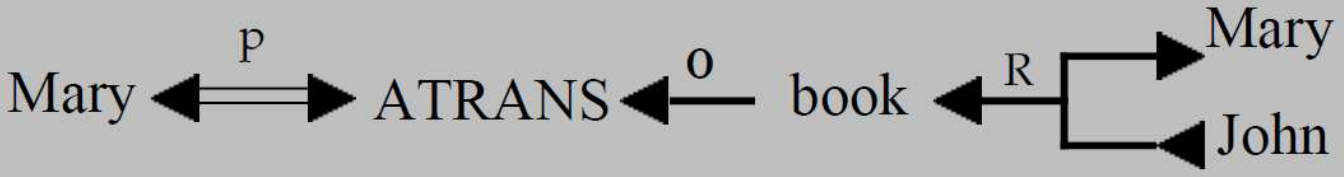
CONTROL
PART (inalienable possession)
POSS (possession)
OWNERSHIP
CONTAIN
PROXIMITY

Examples

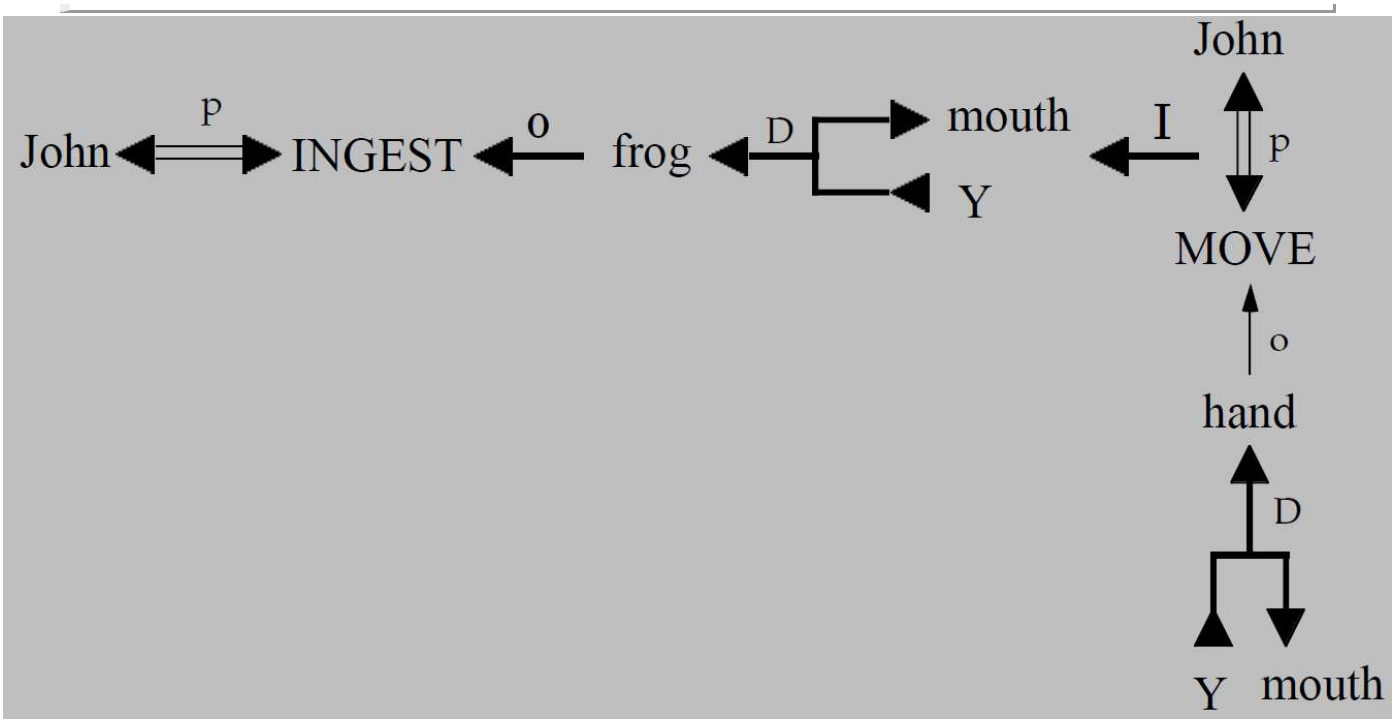
John gave Mary a book.



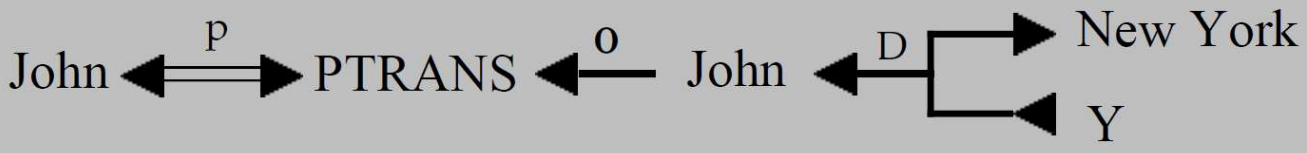
Mary took a book from John.



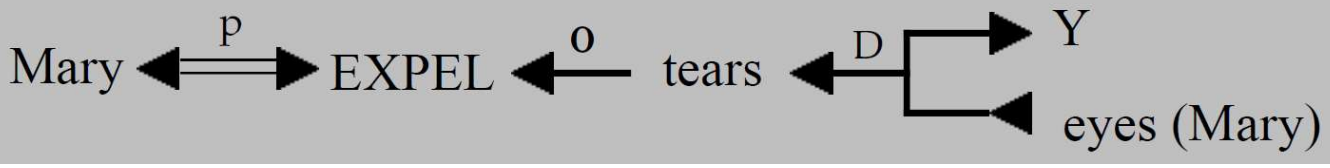
John ate a frog.



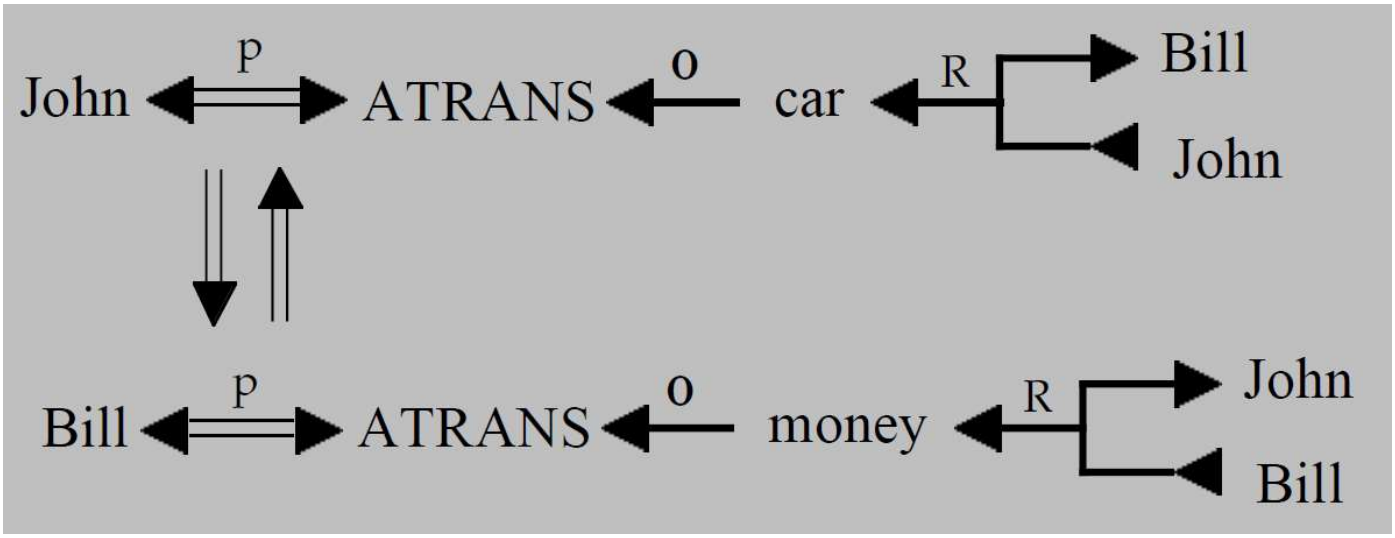
John went to New York.



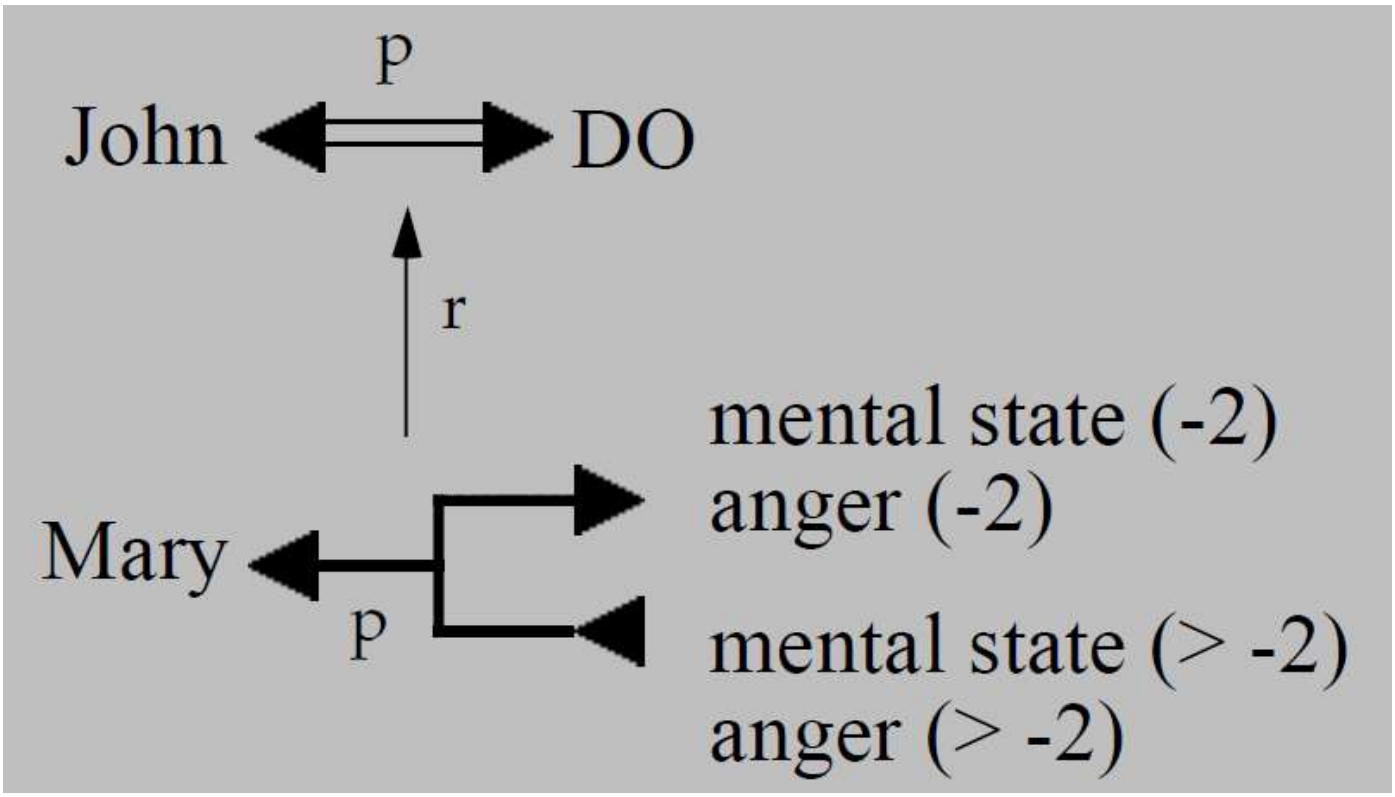
Mary cried.



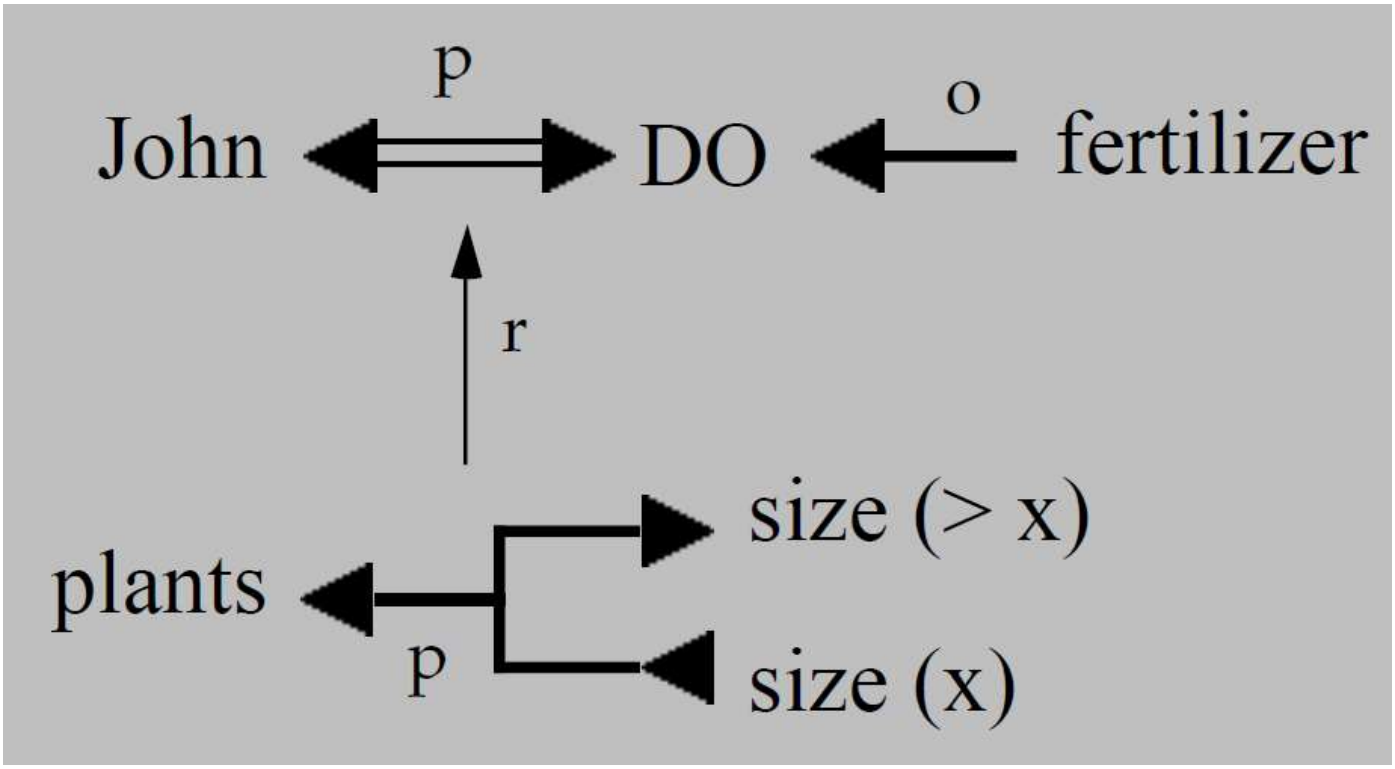
John sold his car to Bill



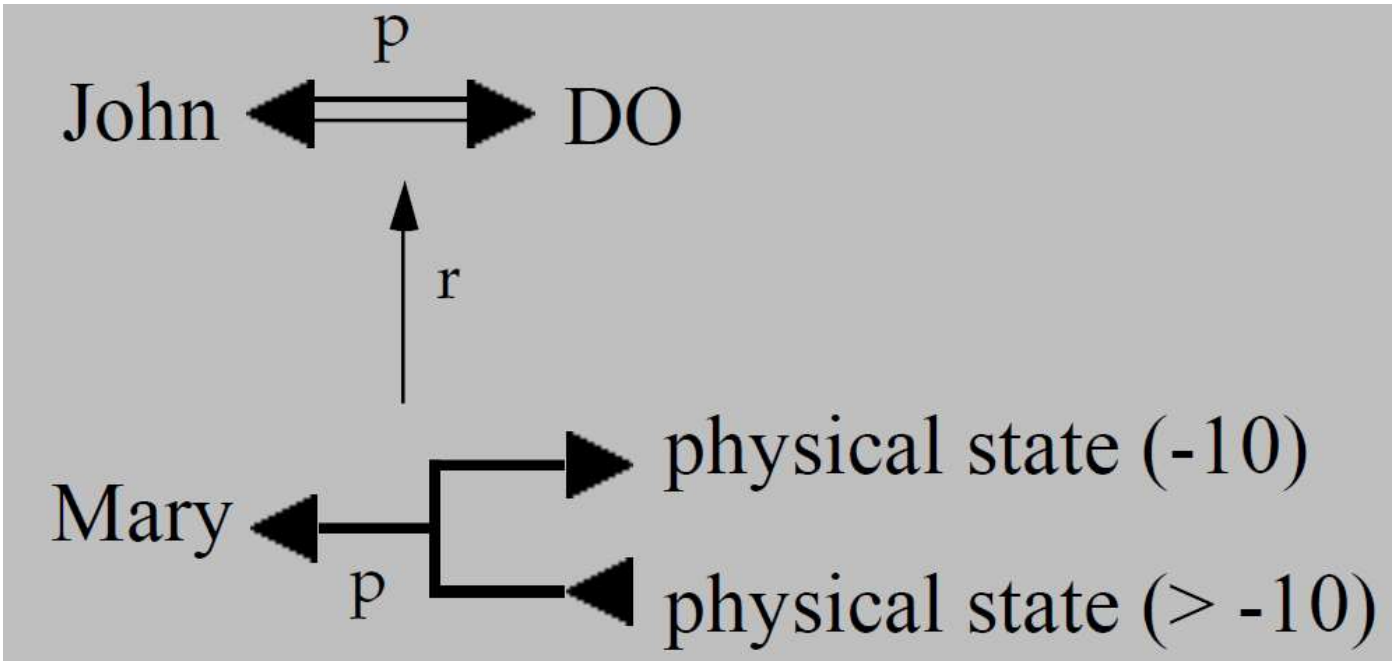
John annoyed Mary



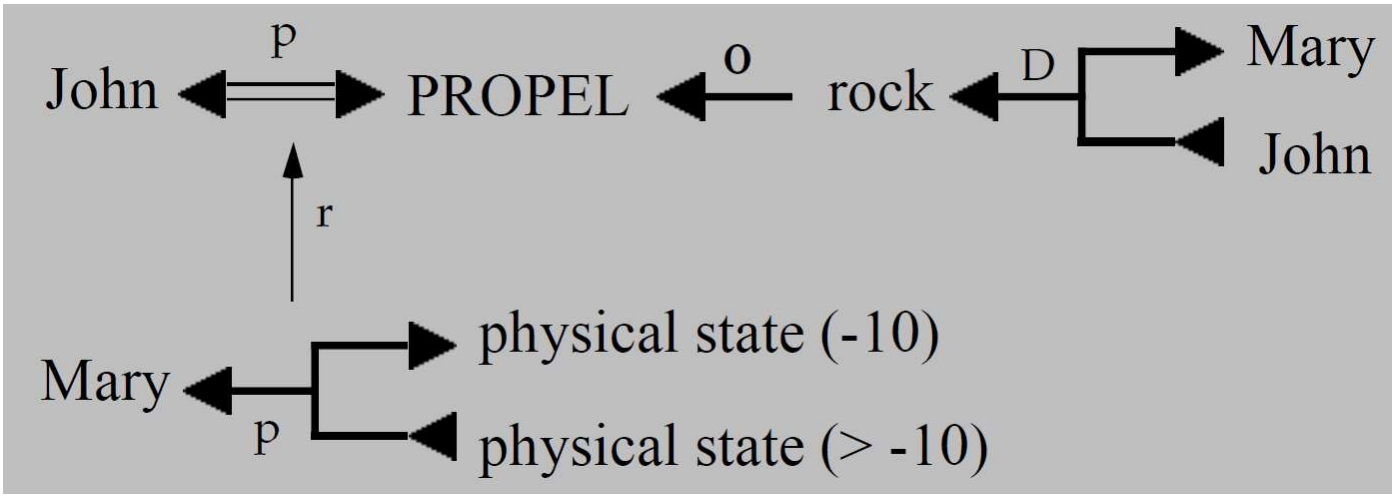
John grew the plants with fertilizer



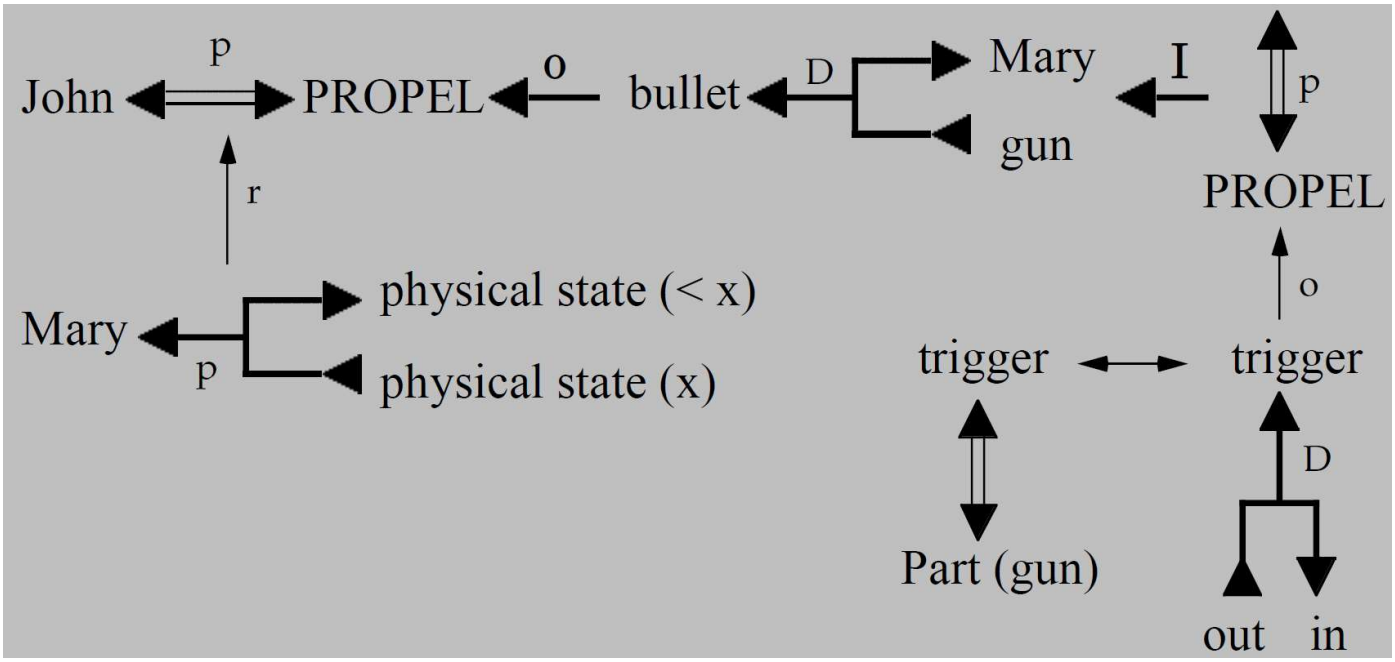
John killed Mary



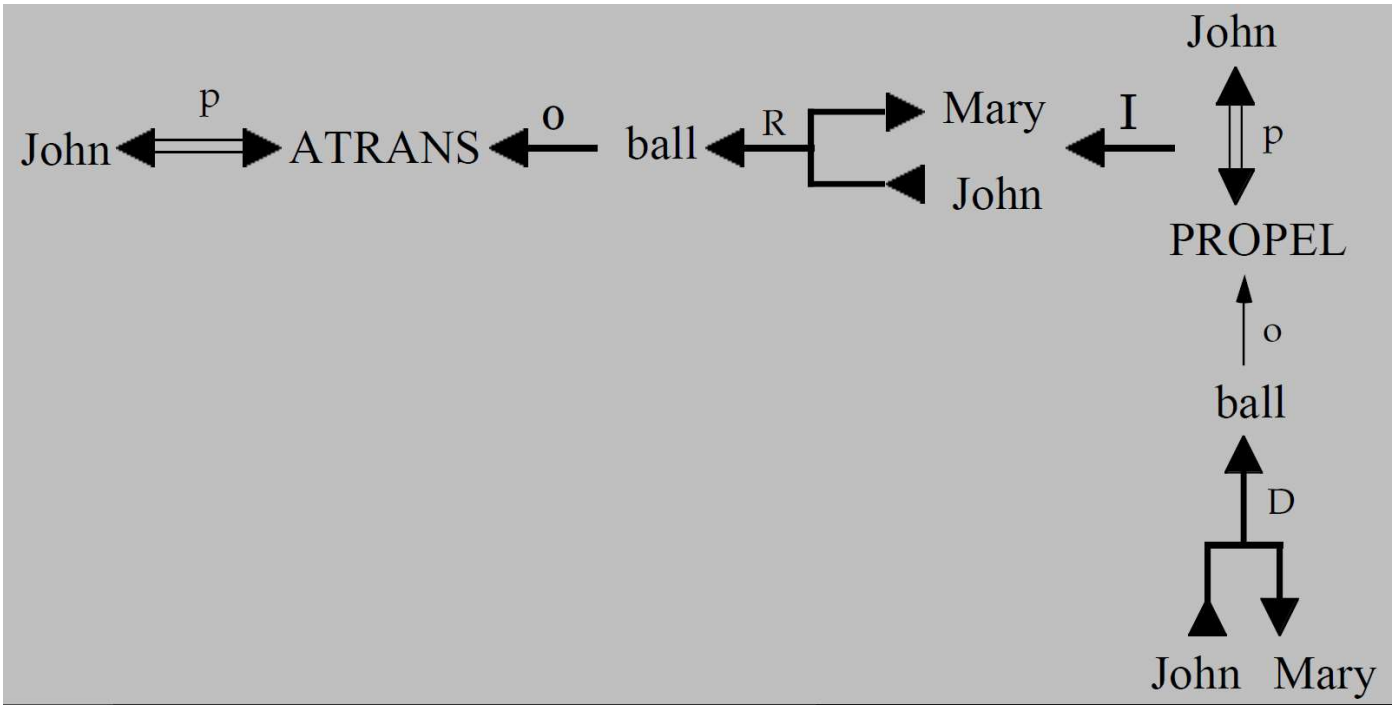
John killed Mary by throwing a rock at her



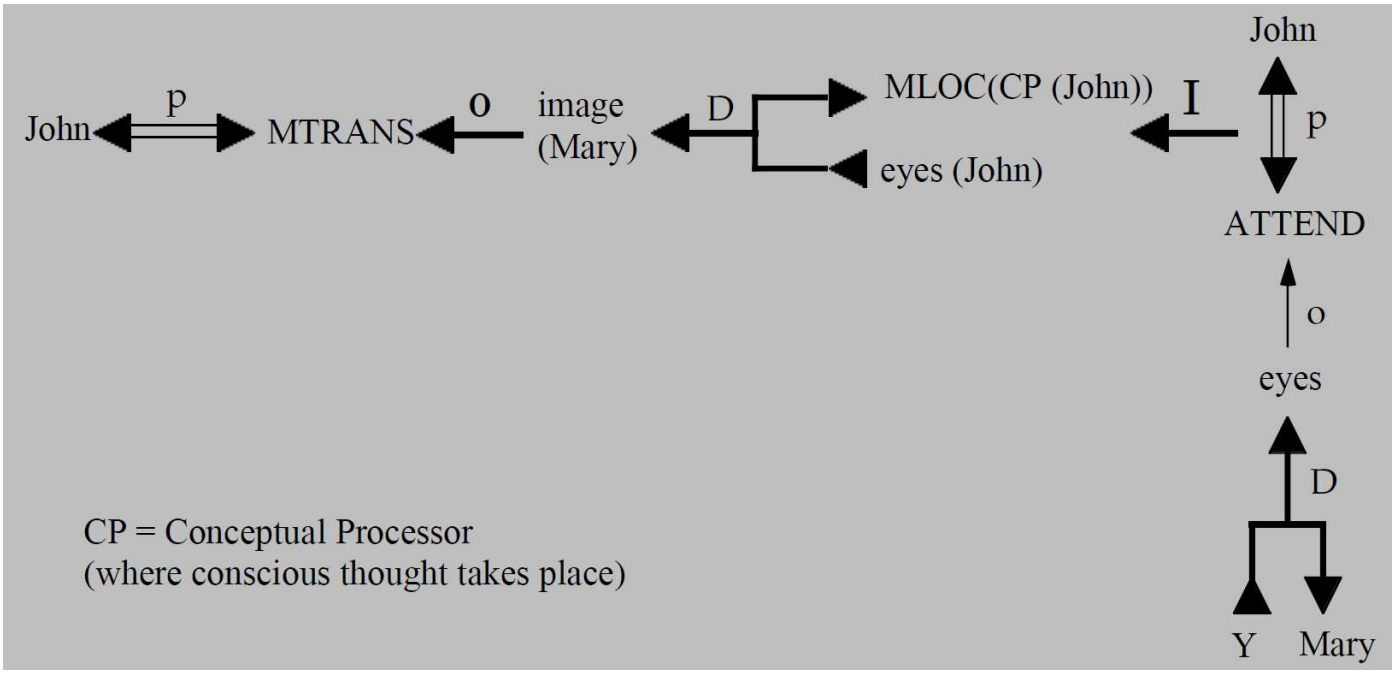
John shot Mary



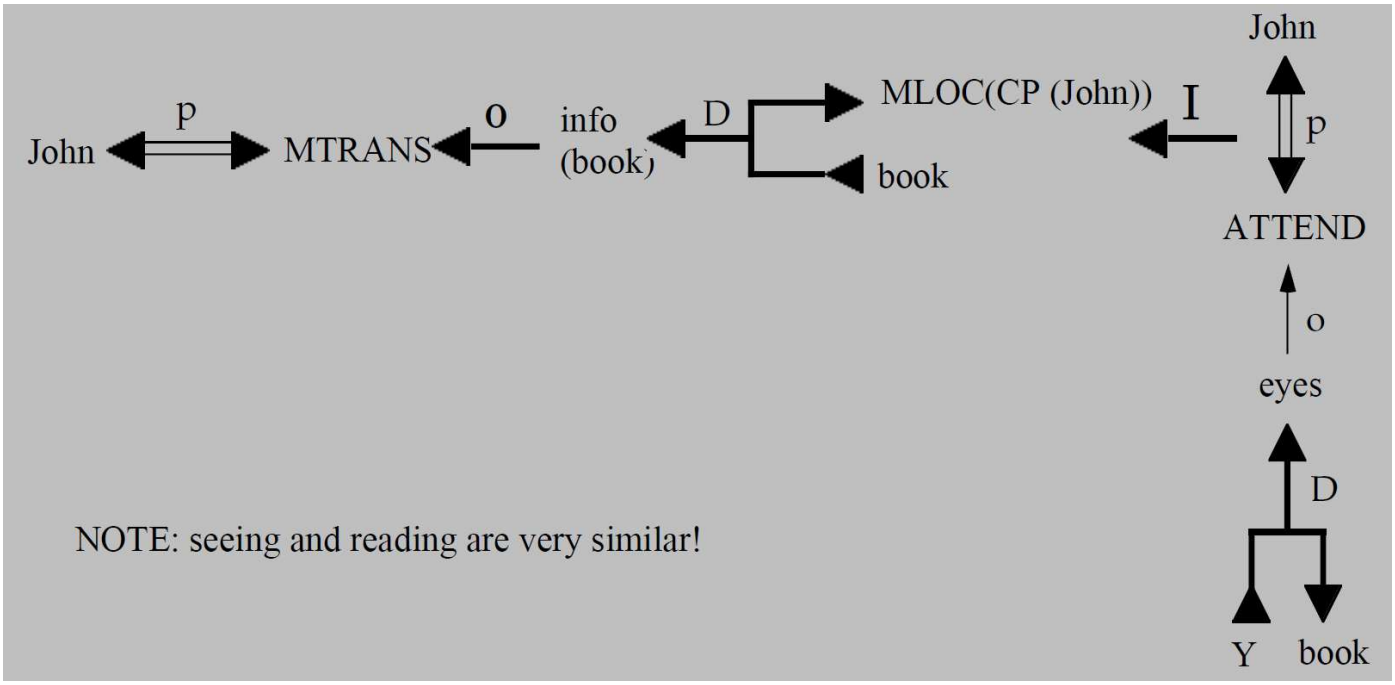
John threw a ball to Mary



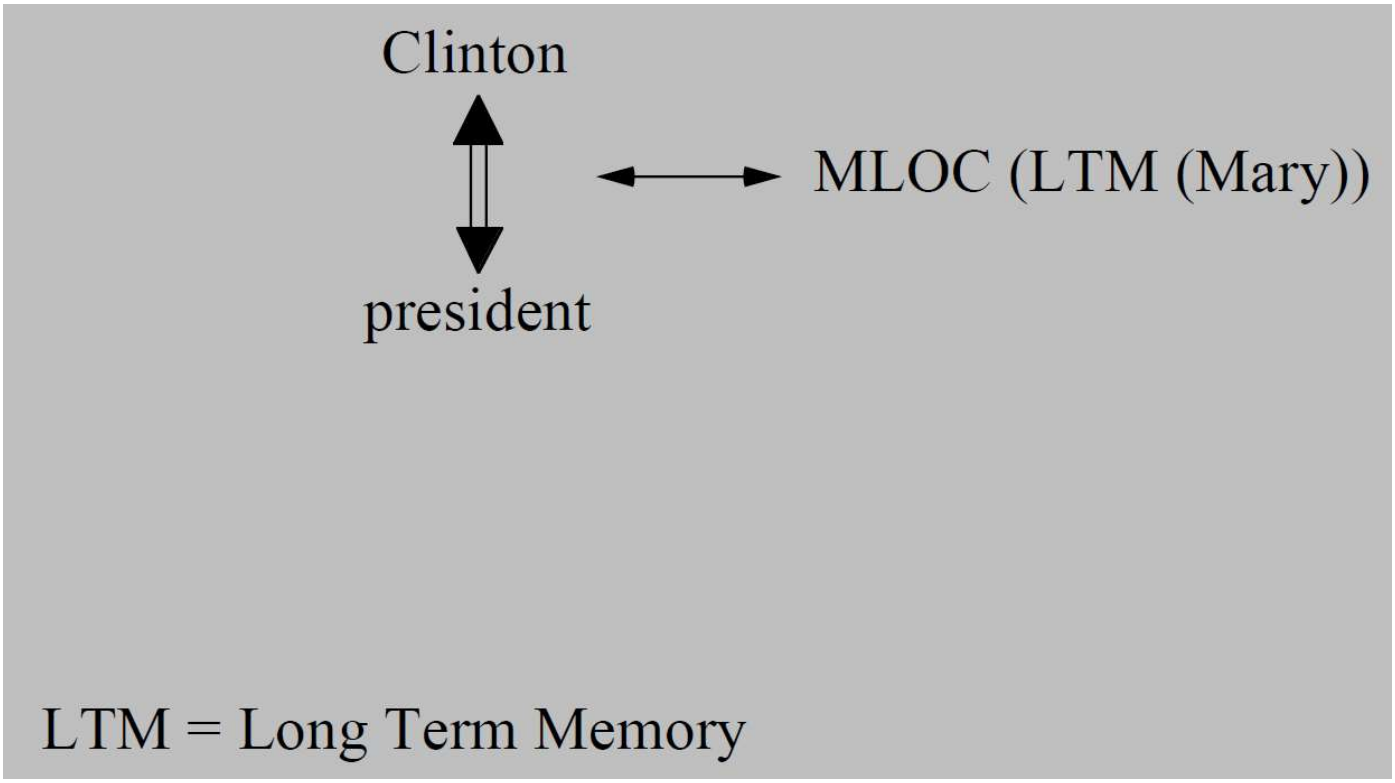
John saw Mary



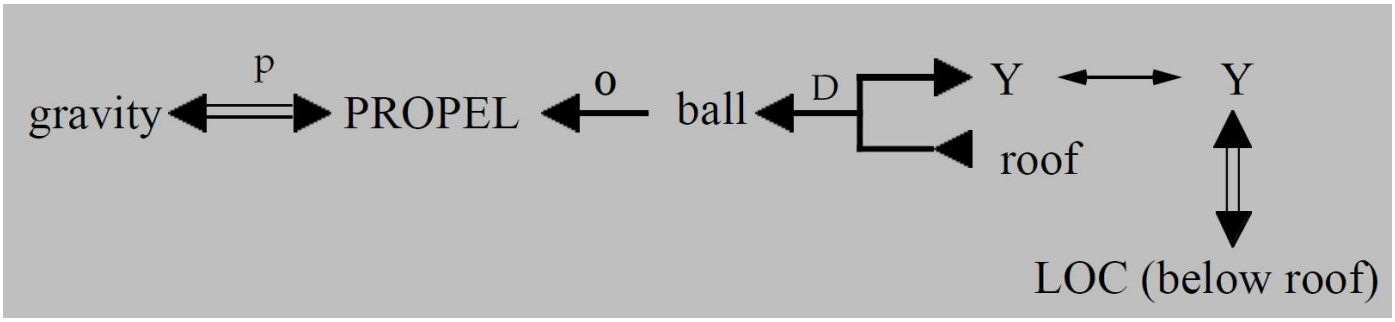
John read a book



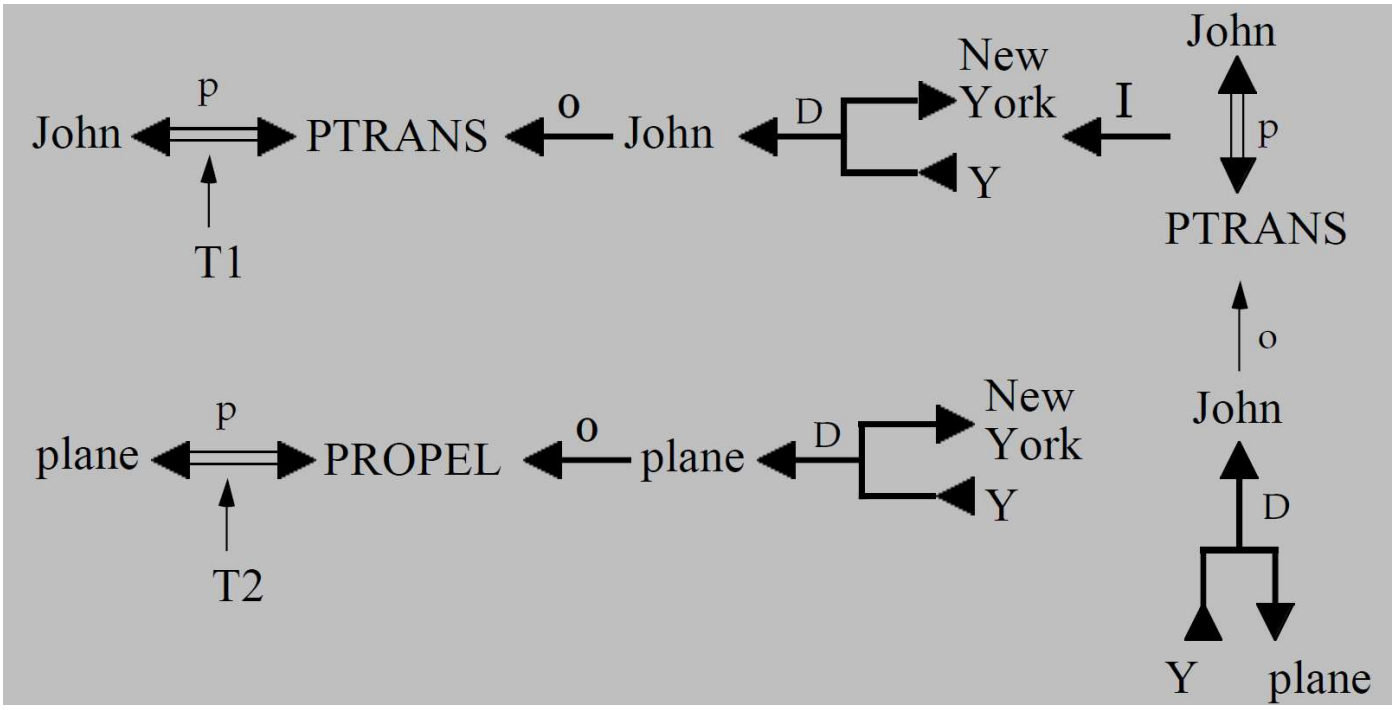
Mary knows that Clinton is president



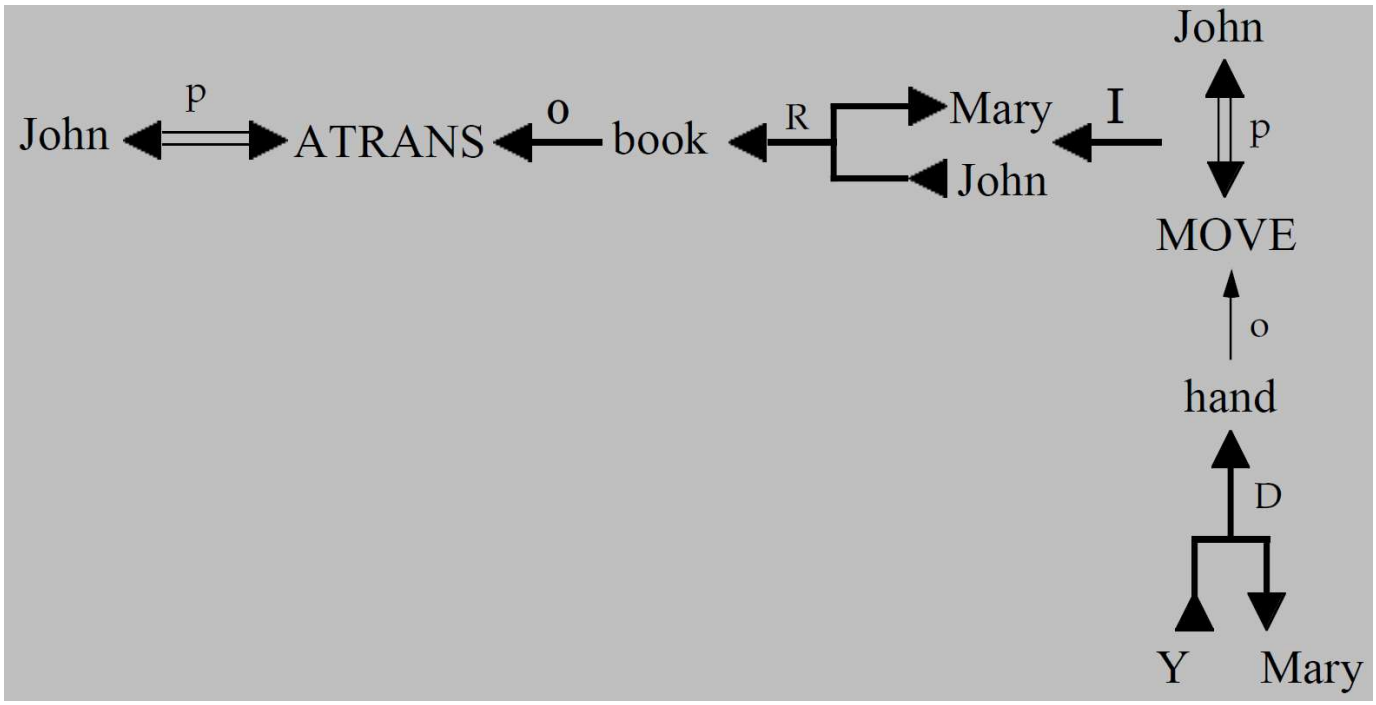
The ball fell from the roof.



John flew to New York



John gave Mary a book by handing it to her



Scripts

Entry conditions: conditions that must be true for the script to be called.

Results: conditions that become true once the script terminates.

Props: “Things” that support the content of the script.

Roles: the actions that the participants perform.

Scenes: a presentation of a temporal aspect of a script.

A RESTAURANT script

Script: RESTAURANT

Track: coffee shop

Props: Tables, Menu, F = food, Check, Money

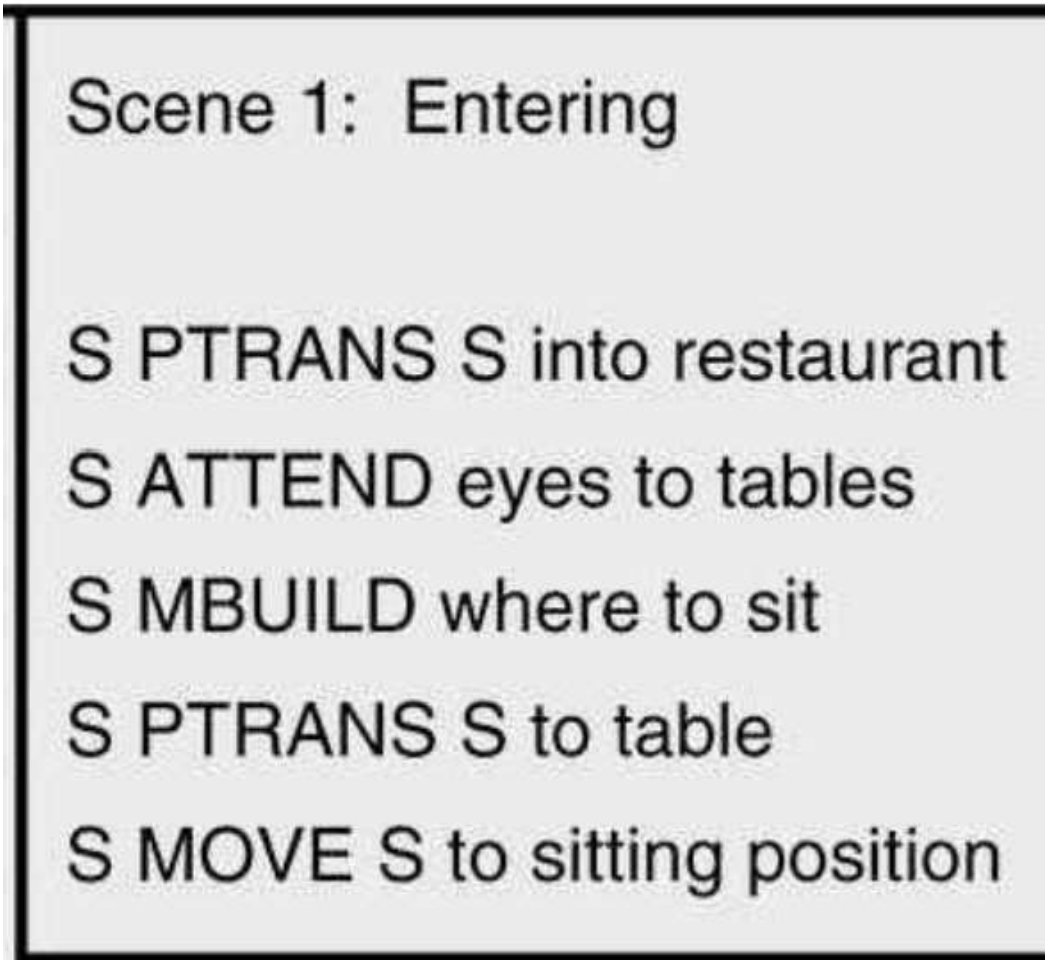
Roles: S = Customer, W = Waiter, C = Cook, M = Cashier, O = Owner

Entry conditions: S is hungry, S has money

Results: S has less money, O has more money, S is not hungry, S is pleased (optional)

Script: Goint to a restaurant Props: Food Tables Menu Money Roles: Owner Customer Waiter Cashier	Scene 1: Entering the restaurant Customer enters the restaurant Scans the tables Chooses the best one Decides to sit there Goes there Occupies the seat
	Scene 2: Ordering the food Customer asks for menu Waiter brings it Customer glances it Chooses what to eat Orders that item
Entry Conditions: Customer is hungry Customer has money Owner has food	Scene 3: Eating the food Waiter brings the food Customer eats it
Results: Customer is not hungry Owner has more money Customer has less money Owner has less food	Scene 4: Paying the bill Customer asks for the bill Waiter brings it Customer pays for it Waiter hands the cash to the cashier Waiter brings the balance amount Customer tips him Customer moves out of the restaurant

Fig. 6.9 Pseudo-form of a restaurant script



Scene 2: Ordering

(Menu on table) (W brings menu)
S PTRANS menu to S

(S asks for menu)
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'need menu' to W
W PTRANS W to menu

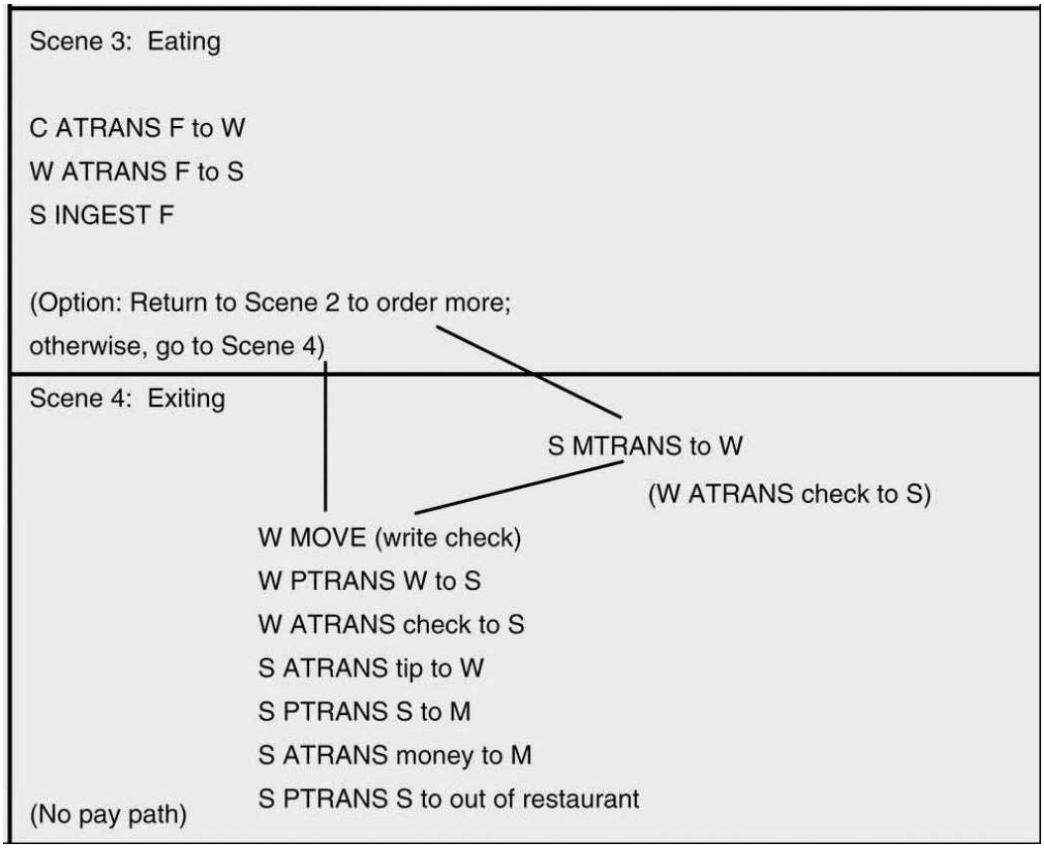
W PTRANS W to table
W ATRANS menu to S

S MTRANS food list to CP (S)
*S MBUILD choice of F
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'I want F' to W

W PTRANS W to C
W MTRANS (ATRANS F) to C

C MTRANS 'no F' to W
W PTRANS W to S
W MTRANS 'no F' to S
(go back to *) or
(go to Scene 4 at no pay path)

C DO (prepare F script)
to Scene 3



Advantages of Scripts:

- Ability to predict events.
- A single coherent interpretation may be build up from a collection of observations.

Disadvantages:

- Less general than frames.
- May not be suitable to represent all kinds of knowledge.

Planning

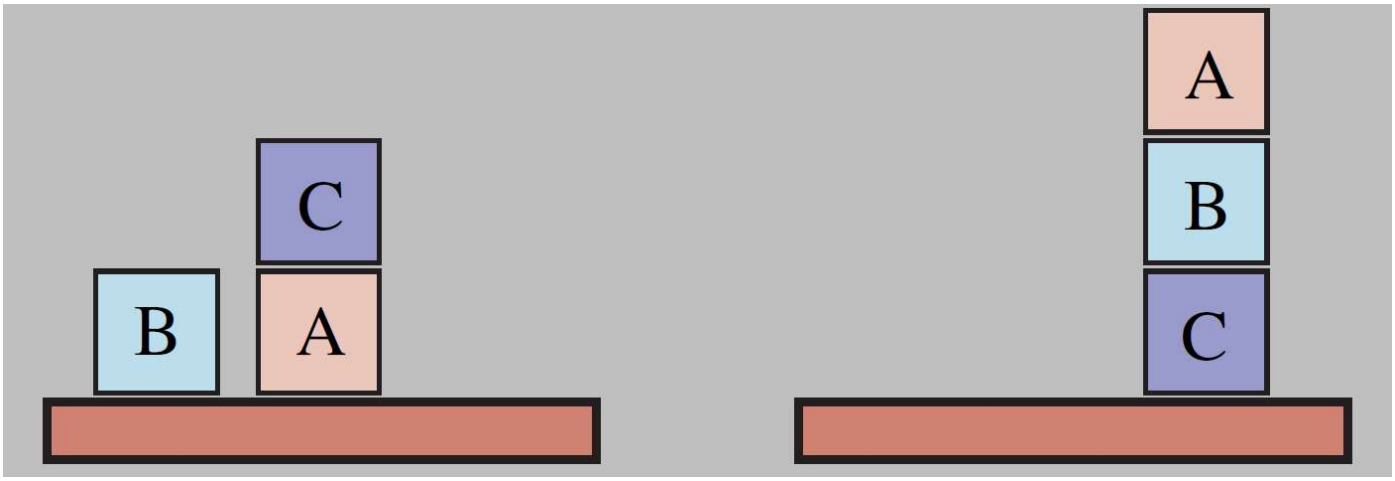
Classical planning is defined as the task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.

What is the Blocks World?

The world consists of:

- A flat surface such as a tabletop
- An adequate set of identical blocks which are identified by letters.

- The blocks can be stacked one on one to form towers of apparently unlimited height.
- The stacking is achieved using a robot arm which has fundamental operations and states which can be assessed using logic and combined using logical operations.
- The robot can hold one block at a time and only one block can be moved at a time.



Problem Description:

$$
\begin{aligned}
&Init(On(A,Table) \land On(B,Table) \land On(C,A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table)) \\
&Goal(On(A,B) \land On(B,C)) \\
&Action(Move(b,x,y), \\
&\quad \text{PRECOND: } On(b,x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT: } On(b,y) \land Clear(x) \land \neg On(b,x) \land \neg Clear(y)) \\
&Action(MoveToTable(b,x), \\
&\quad \text{PRECOND: } On(b,x) \land Clear(b) \land Block(b) \land Block(x), \\
&\quad \text{EFFECT: } On(b,Table) \land Clear(x) \land \neg On(b,x))
\end{aligned}
$$

One solution is the sequence

```
MoveToTable(C,A), Move(B,Table,C), Move(A,Table,B)
```

Possibe Actions of Robot Arm

`STACK(X,Y)` : Stacking Block X on Block Y
`UNSTACK(X,Y)` : Picking up Block X which is on top of Block Y
`PICKUP(X)` : Picking up Block X which is on top of the table
`PUTDOWN(X)` : Put Block X on the table

All the four operations have certain preconditions which need to be satisfied to perform the same. These preconditions are represented in the form of predicates.

The effect of these operations is represented using two lists ADD and DELETE. DELETE List contains the predicates which will cease to be true once the operation is performed. ADD List on the other hand contains the predicates which will become true once the operation is performed.

The Precondition, Add and Delete List for each operation have been listed below.

OPERATORS	PRECONDITION	DELETE	ADD
STACK(X,Y)	CLEAR(Y)∧ HOLDING(X)	CLEAR(Y) HOLDING(X)	ARMEMPTY ON(X,Y)
UNSTACK(X,Y)	ARMEMPTY∧ ON(X,Y)∧ CLEAR(X)	ARMEMPTY∧ ON(X,Y)	HOLDING(X) ∧CLEAR(Y)
PICKUP(X)	CLEAR(X)∧ ONTABLE(X)∧ ARMEMPTY	ONTABLE(X)∧ ARMEMPTY	HOLDING(X)
PUTDOWN(X)	HOLDING(X)	HOLDING(X)	ONTABLE(X)∧ ARMEMPTY

List of possible predicates:

`ON(X,Y)` - block X is on block Y.

`ONTABLE(X)` - block X is on the table.

`CLEAR(X)` - block X has nothing on it.

`HOLDING(X)` - the arm holds block X.

`ARMEMPTY` - the arm holds nothing.

Using logic but not logical notation we can say that

- If the arm is holding a block it is not empty
- If block A is on the table it is not on any other block

- If block A is on block B,block B is not clear.

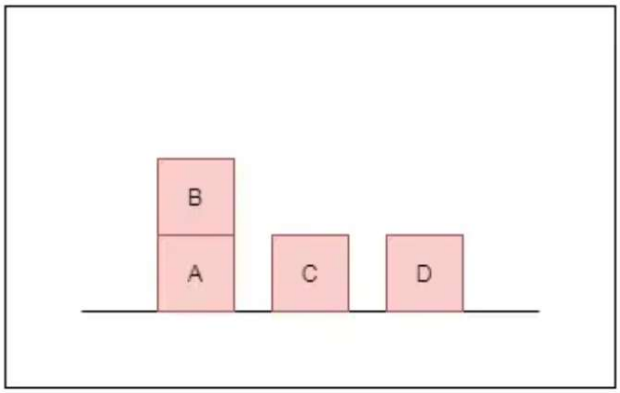
Components of a Planning System

Simple problem solving tasks basically involve the following tasks:

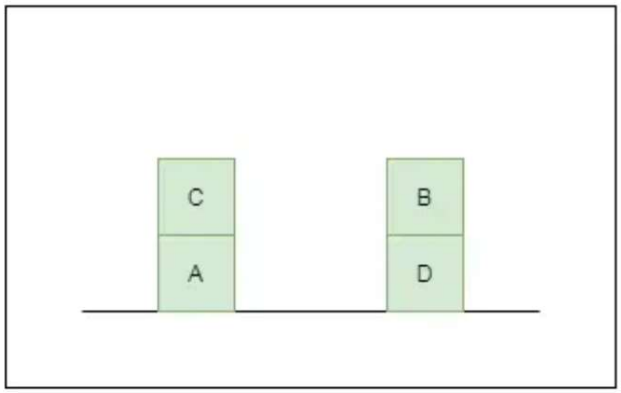
- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.
- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

Goal Stack Planning Algorithm :

1. Start by pushing the original goal on the stack. Repeat this until the stack becomes empty. If stack top is a compound goal, then push its unsatisfied subgoals on the stack.
2. If stack top is a single unsatisfied goal then, replace it by an action and push the action's precondition on the stack to satisfy the condition.
3. If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.
4. If stack top is a satisfied goal, pop it from the stack.



Initial State



Goal State

Initial State:

$$ON(B,A) \wedge ONTABLE(A) \wedge ONTABLE(C) \wedge ONTABLE(D) \wedge CLEAR(B) \wedge CLEAR(C) \wedge CLEAR(D) \wedge ARMEMPTY$$

Goal State:

$$ON(C,A) \land ON(B,D) \land ONTABLE(A) \land ONTABLE(D) \land CLEAR(B) \land CLEAR(C) \land ARMEMPTY$$

Step1: The following conditions are already true in Initial state

$$ONTABLE(A) \land ONTABLE(D) \land CLEAR(B) \land CLEAR(C) \land ARMEMPTY$$

Step2: The only goals to be proved are

$$ON(C,A) \land ON(B,D)$$

Step3: Push the compound goal and sub goals on to Stack

$$ON(C,A)$$

$$ON(B,D)$$

$$ON(C,A) \land ON(B,D)$$

To solve $ON(C,A)$ we an apply the action $STACK(C,A)$.

Step4: Replace $ON(C,A)$ with $STACK(C,A)$ in the Stack.

$$STACK(C,A)$$

$$ON(B,D)$$

$$ON(C,A) \land ON(B,D)$$

$STACK(C,A)$ can be applied if its pre-conditions are true.

Step5: Add pre-conditions of $STACK(C,A)$ on to the Stack.

$$CLEAR(A)$$

Step5: Add pre-conditions of STACK(C,A) on to the Stack.

HOLDING(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

The top of the Stack contains CLEAR(A) which is not true. Make the state true.

Step6: Replace CLEAR(A) by UNSTACK(B,A) and also add its pre-conditions

ON(B,A)

CLEAR(B)

ARMEMPTY

ON(B,A) ∧ CLEAR(B) ∧ ARMEMPTY

UNSTACK(B,A)

HOLDING(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

Step7: ON(B,A), CLEAR(B) and ARMEMPTY are all true. So pop these sub goals along with their compound goal.

State of Goal Stack after Step7

State of Goal Stack after Step7

UNSTACK(B,A)

HOLDING(C)

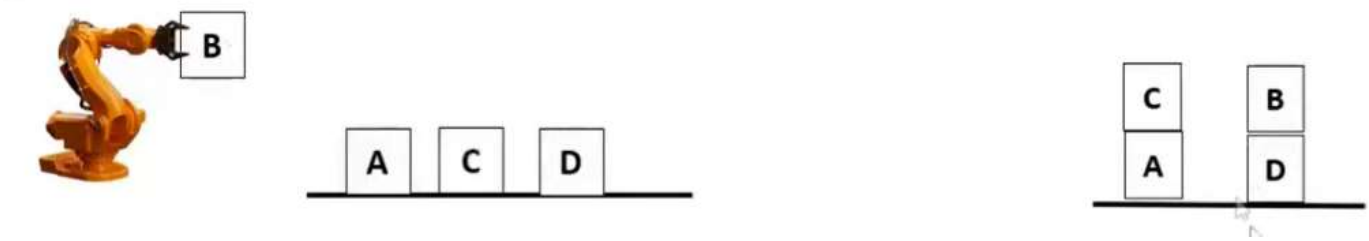
CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

Step8: Pop the action `UNSTACK(B,A)`, execute it and add it to a Queue of sequence of actions.



State of Goal Stack after Step8

HOLDING(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

The top of the Stack contains `HOLDING(C)` which is not true. To make the state true use the action `STACK(B,D)`.

Step9: Push `STACK(B,D)` and its pre-conditions on to the Goal Stack

Step9: Push STACK(B,D) and its pre-conditions on to the Goal Stack

HOLDING(B)

CLEAR(D)

CLEAR(D) ∧ HOLDING(B)

STACK(B,D)

HOLDING(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

As the pre-condition CLEAR(D) ∧ HOLDING(B) is true, pop the sub goals and the action STACK(B,D) . Execute the action STACK(B,D) and add it to the Queue.

```
Queue = UNSTACK(B,A) , STACK(B,D)
```

State of the Stack after executing the action STACK(B,D)

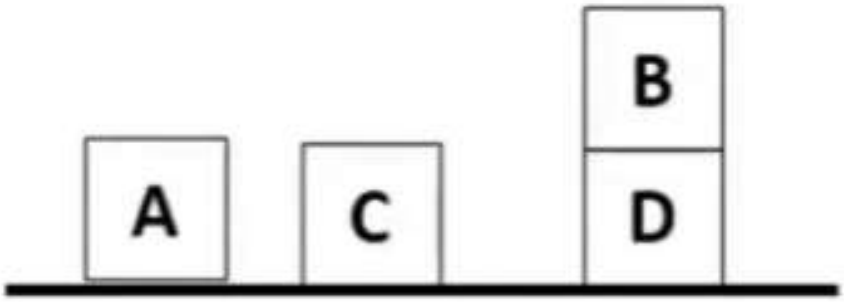
HOLDING(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)



The top of the Stack HOLDING(C) is not true. To make it true use the action PICKUP(C) .

Step10: Push the action PICKUP(C) and its pre-conditions on to the Stack.

ONTABLE(C) ∧ CLEAR(C) ∧ ARMEMPTY

PICKUP(C)

CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

ON(C,A) ∧ ON(B,D)

As the pre-condition ONTABLE(C) ∧ CLEAR(C) ∧ ARMEMPTY is true, pop the sub goals and the action PICKUP(C) . Execute the action PICKUP(C) and add it to the Queue.

Queue = UNSTACK(B,A) , STACK(B,D) , PICKUP(C)

Contents of the Goal Stack

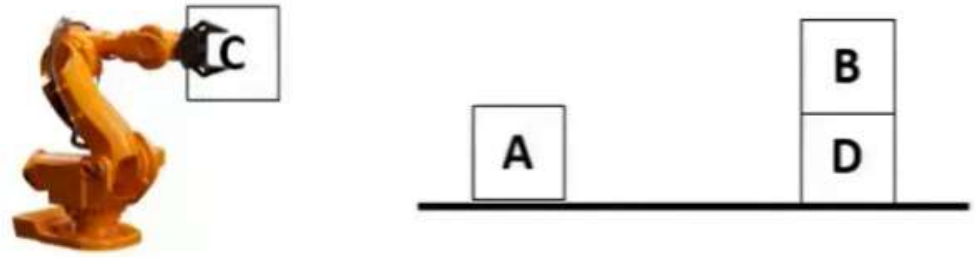
CLEAR(A) ∧ HOLDING(C)

STACK(C,A)

ON(B,D)

Contents of the Goal Stack

$$ON(C,A) \wedge ON(B,D)$$



As the pre-condition $CLEAR(A) \wedge HOLDING(C)$ is true, pop the sub goals and the action $STACK(C,A)$. Execute the action $STACK(C,A)$ and add it to the Queue.

> Queue = UNSTACK(B,A) , STACK(B,D) , PICKUP(C) , STACK(C,A)

Solution: Using Goal Stack algorithm the plan for the given problem is

> UNSTACK(B,A) , STACK(B,D) , PICKUP(C) , STACK(C,A)

Hierarchical Planning

Principle

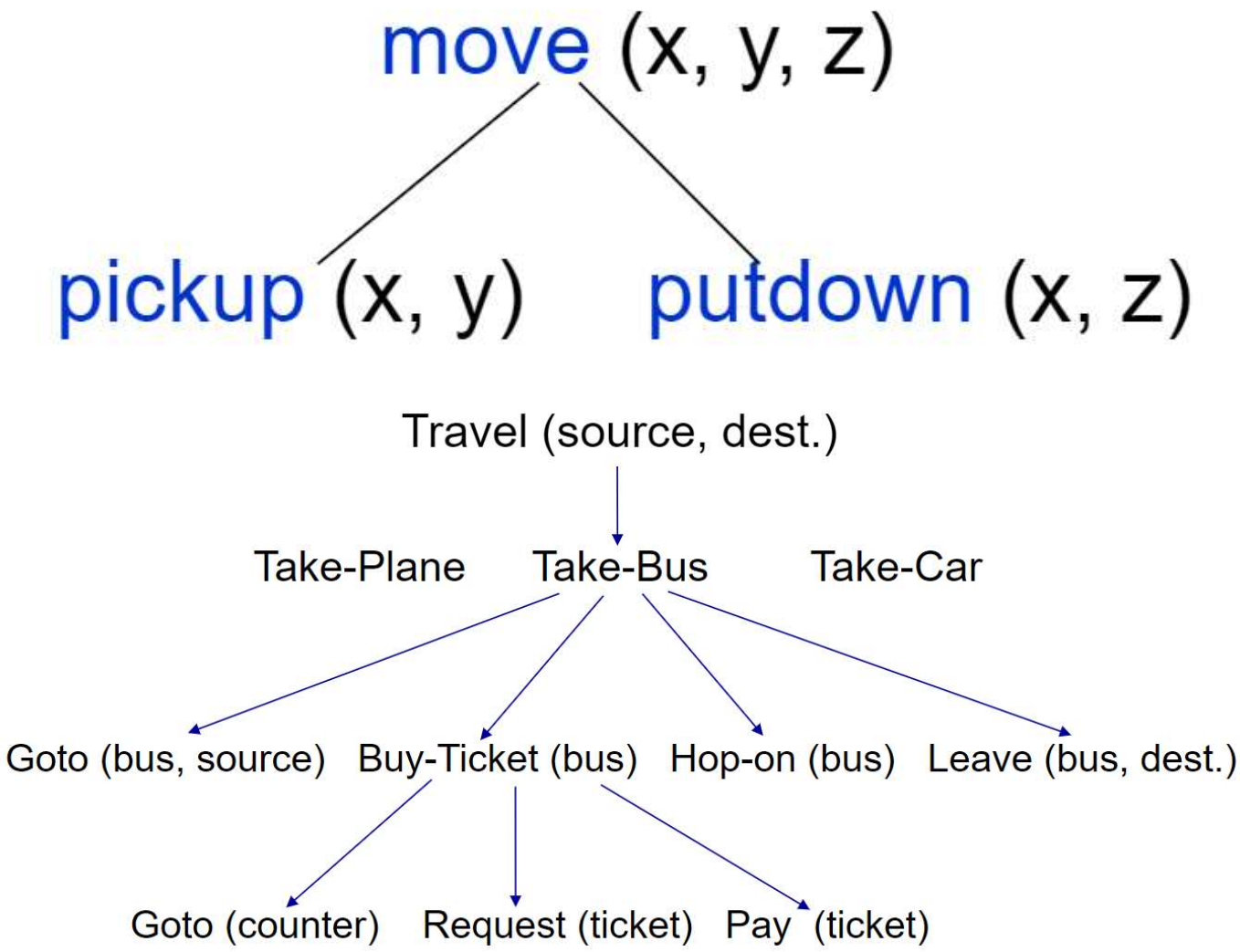
- hierarchical organization of 'actions'
- complex and less complex (or: abstract) actions
- lowest level reflects directly executable actions

Procedure

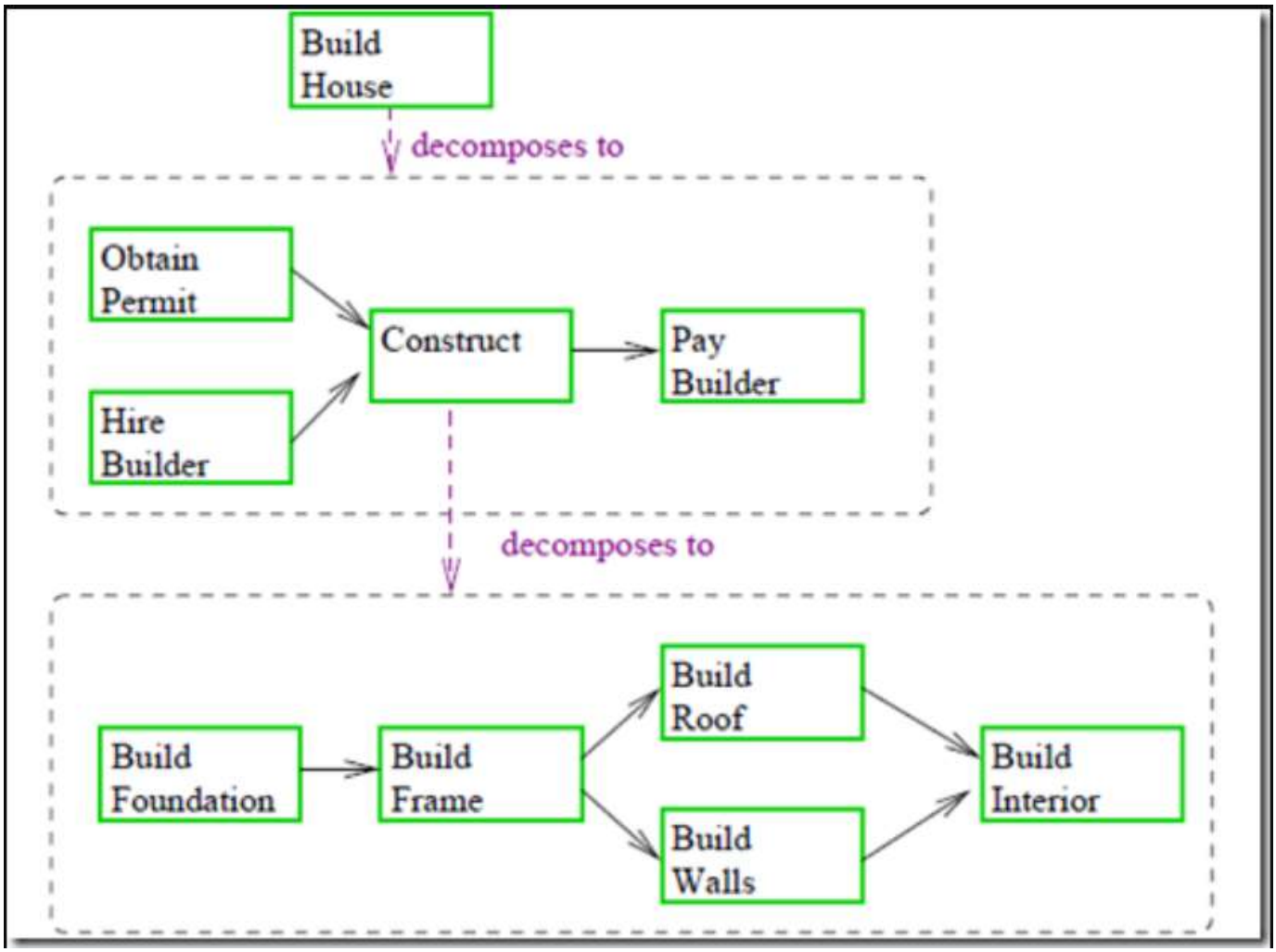
- planning starts with complex action on top
- plan constructed through action decomposition
- substitute complex action with plan of less complex actions
- overall plan must generate effect of complex action

Plan Decomposition

- Plans are organized in a hierarchy.
- Links between nodes at different levels in the hierarchy denote a decomposition of a " complex action " into more primitive actions (operator expansion).



For example, in case of building a house, hierarchical planning is used as shown below.


Build House
decomposes to
Obtain Permit
Hire Builder
Construct
Pay Builder
decomposes to
Build Foundation
Build Frame
Build Roof
Build Walls
Build Interior