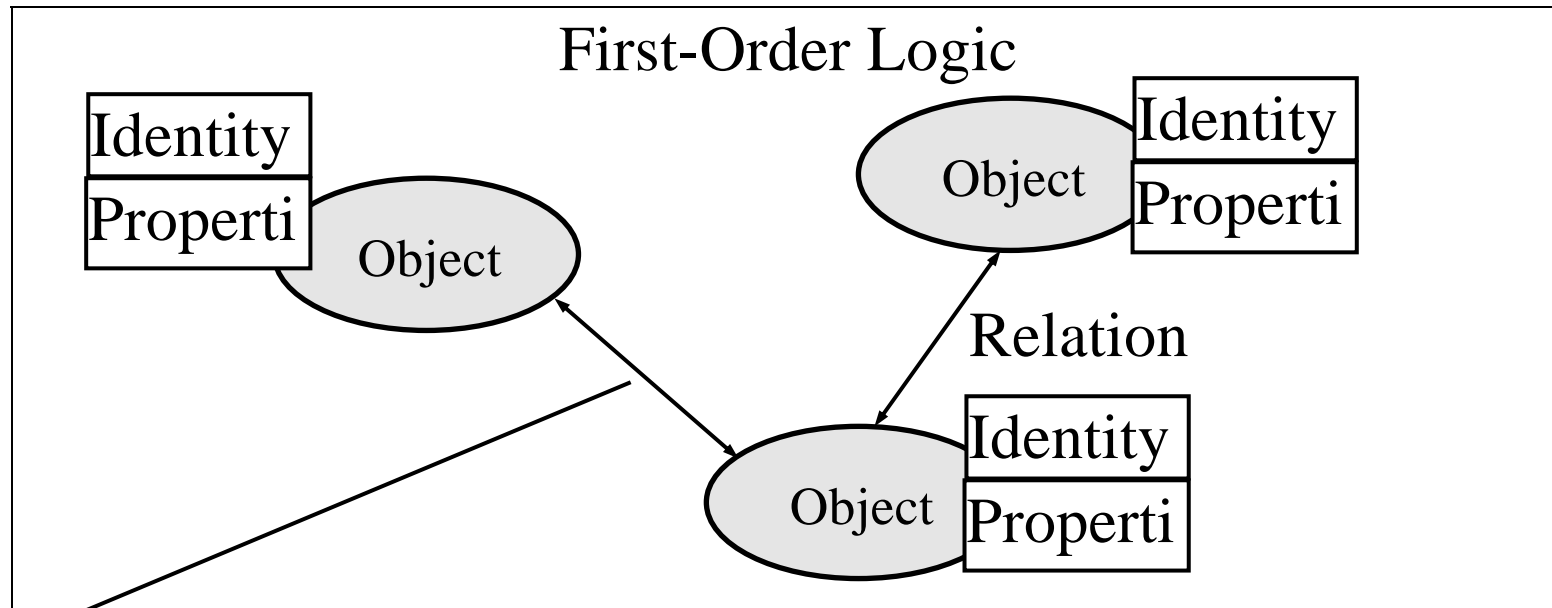


First-Order Logic



Relation in which there is only one “value” for a given “input”

For example:

Objects: people, house, numbers,...
Relations: brother of, bigger than, inside,...
Properties: red, round, bogus,...
Functions: father of, best friend, cosine of

Facts that are decomposed into:

“One plus two equals three”

Objects: one, two, three, one plus two.

Function: plus.

Property: additive.

Relation: equals.

- ✧ First-order logic can express facts about all of the objects in the universe.
- ✧ First-order logic does not make an ontological commitment to time, events.
- ✧ First-order logic gives its users the freedom to describe things (objects) in a way that is appropriate for the domain.
- ✧ First-order logic is by far the most studied and best understood scheme yet devised.

Syntax and Semantics

- ✧ First-order logic uses terms to represent objects.
- ✧ Constant symbols, variables, and function symbols are used to build terms.
- ✧ Quantifier and predicate symbols are then applied to terms to build sentences.
- ✧ Sentences are used to represent facts.
- ✧ A complete grammar is as shown in (Figure 1)

<Sentence> := <AtomicSentence>
 | **<Sentence> <Connective> <Sentence>**
 | **<Quantifier> <Variable>,... <Sentence>**
 | **\neg <Sentence>**
 | **(<Sentence>)**

<Atomic Sentence> := <Predicate>(<Term>,...)
 | **<Term> = <Term>**

<Term> := <Function>(<Term>,...)
 | **<Constant> | <Variable>**

<Connective> := \wedge | \vee | \Leftrightarrow | \Rightarrow
<Quantifier> := \forall | \exists
<Constant> := Martin | 59302 | Cat | X | ...
<Variable> := a | x | s | ...
<Predicate> := Before | Likes | Raining | Fails | ...
<Function> := Father | Hairof | 304grade for | ...

Figure 1. BNF for first-order logic

✧ **Constant symbols:** *A, B, C, John, ...*

- ✓ An interpretation must specify which object in the world is referred to by each constant symbol, and each constant symbol names exactly **one** object.
- ✓ Some object can have several names.

✧ **Predicate symbols:** *Round, Brother, ...*

- ✓ An interpretation specifies that a predicate symbol refers to a particular relation in the model.
- ✓ In any given model, the relation is defined by the set of **tuples** of objects that satisfy it.
- ✓ A **tuple** is a collection of objects arranged in a fixed order; they are written with angle brackets surrounding the objects.
- ✓ For example:
 { <King John, Richard the Lionheart>, <Richard the Lionheart, King John> }.

✧ **Function symbols:** *Cosine*, *FatherOf*, ...

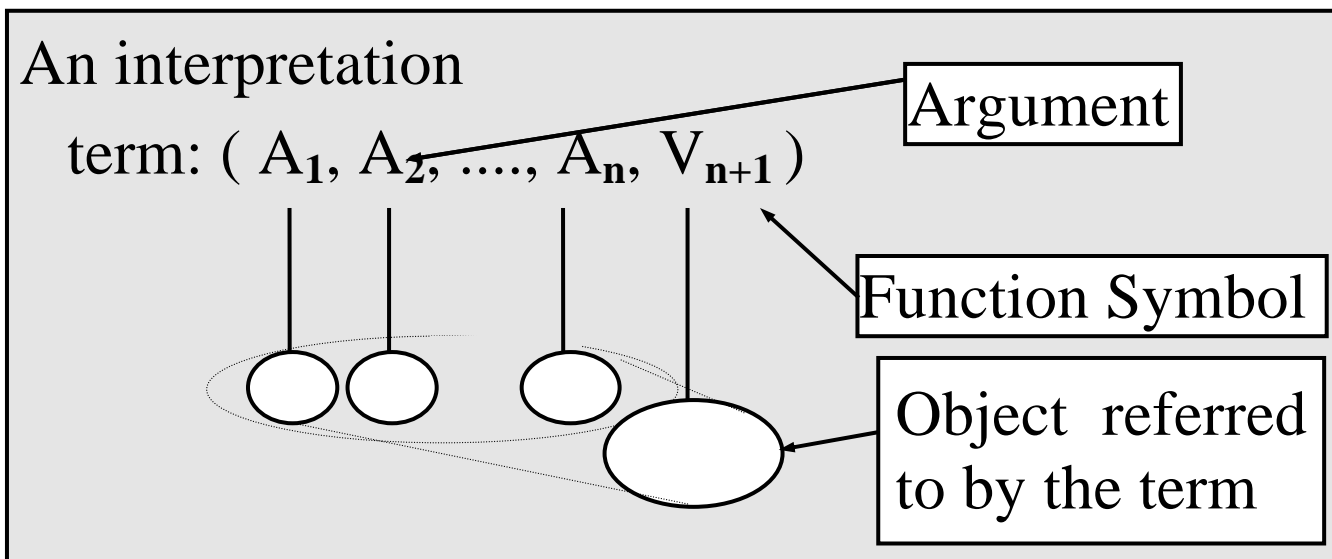
- ✓ Some relations are *functional* -- any given object is related to exactly another object by the relation.
- ✓ In any given model, the mapping is just a set of ***n+1* tuples** with a special property:

The last element of each tuple is the value of the function for the first *n* elements, and each combination of the first *n* elements appears in exactly one tuple.

- ✓ Function symbols are used to refer to particular objects without using their names.
- ✓ For example: A table of *cosine*.

Terms

- ✧ A **term** is a logical expression that refers to an object -- e.g. *John*.
- ✧ A **complex term** is formed by a function symbol followed by a parenthesized list of terms as *arguments* to the function symbol.
- ✧ For example: *LeftLegOf(John)* (instead of using a constant symbol *LeftLegOfJohn*).
- ✧ The **formal semantics**:



- ✧ For example:

{ <King John, King John's left leg>,
 <Richard the Lionheart, Richard's left leg > }.

Atomic Sentence

- ✧ An **atomic sentence** is formed from a predicate symbol followed by a parenthesized list of terms. For example:

Brother(Richard, John).

- ✧ Atomic sentences can have arguments that are complex terms. For example:

Married(FatherOf(Richard), MotherOf(John))

- ✧ An atomic sentence is **true** if the *relation* referred to by the predicate symbol *holds* between the objects referred to by the arguments.

Complex Sentence

- ✧ Complex sentences are constructed using **logical connectives** and atomic sentences.
- ✧ The semantics of the logic connectives is identical to that in the propositional logic.
- ✧ For examples:

$\text{Brother}(\text{John}, \text{Richard}) \wedge \text{Brother}(\text{Richard}, \text{John}).$

$\text{Older}(\text{John}, 30) \Rightarrow \neg \text{Younger}(\text{John}, 30).$

Quantifiers

- ✧ **Quantifiers** can be used to express properties of entire collections of objects, rather than having to enumerate the objects by name.
- ✧ **Universal quantification** (\forall): Used to express general rules about *every(all)* object such as “All cats are mammals”

$$\forall x \text{ Cat}(x) \Rightarrow \text{Mammal}(x) \quad \Rightarrow \text{preferred}$$

$\forall x \text{ } P$ is **true** if *all* the sentences obtained by substituting the name of an object for the variable x are **true**. Thus, the above is equivalent to :

$$\begin{aligned} &\text{Cat}(\text{Spot}) \Rightarrow \text{Mammal}(\text{Spot}) \wedge \\ &\text{Cat}(\text{Rebecca}) \Rightarrow \text{Mammal}(\text{Rebecca}) \wedge \\ &\text{Cat}(\text{Felix}) \Rightarrow \text{Mammal}(\text{Felix}) \wedge \\ &\text{Cat}(\text{Richard}) \Rightarrow \text{Mammal}(\text{Richard}) \wedge \\ &\text{Cat}(\text{John}) \Rightarrow \text{Mammal}(\text{John}) \wedge \\ &\quad \vdots \end{aligned}$$

A term with no variables is called a **ground** term.

✧ **Existential quantification** (\exists): Used to make a statement about *some* object in the universe without naming it, such as “Spot has a sister who is a cat”.

$$\exists x \text{ Sister}(x, \text{Spot}) \wedge \text{Cat}(x) \wedge \text{preferred}$$

$\exists x P$ is **true** if *at least one* of the all sentences obtained by substituting the name of an object for the variable x is true . Thus, the above is equivalent to :

$$\begin{aligned} &(\text{Sister}(\text{Spot}, \text{Spot}) \wedge \text{Cat}(\text{Spot})) \vee \\ &(\text{Sister}(\text{Rebecca}, \text{Spot}) \wedge \text{Cat}(\text{Rebecca})) \vee \\ &(\text{Sister}(\text{Felix}, \text{Spot}) \wedge \text{Cat}(\text{Felix})) \vee \\ &(\text{Sister}(\text{Richard}, \text{Spot}) \wedge \text{Cat}(\text{Richard})) \vee \\ &(\text{Sister}(\text{John}, \text{Spot}) \wedge \text{Cat}(\text{John})) \vee \\ &\vdots \end{aligned}$$

✧ **Nonsensible combinations of connectives and quantifier:**

$$\forall x \text{ Cat}(x) \wedge \text{Mammal}(x)$$

$$\begin{aligned} &\text{Cat}(\text{Spot}) \wedge \text{Mammal}(\text{Spot}) \wedge \\ &\text{Cat}(\text{Rebecca}) \wedge \text{Mammal}(\text{Rebecca}) \wedge \\ &\text{Cat}(\text{Felix}) \wedge \text{Mammal}(\text{Felix}) \wedge \\ &\text{Cat}(\text{Richard}) \wedge \text{Mammal}(\text{Richard}) \wedge \\ &\text{Cat}(\text{John}) \wedge \text{Mammal}(\text{John}) \wedge \end{aligned}$$

$$\vdots$$

$$\exists x \text{ Sister}(x, \text{Spot}) \Rightarrow \text{Cat}(x)$$

$$\begin{aligned} &(\text{Sister}(\text{Spot}, \text{Spot}) \Rightarrow \text{Cat}(\text{Spot})) \vee \\ &(\text{Sister}(\text{Rebecca}, \text{Spot}) \Rightarrow \text{Cat}(\text{Rebecca})) \vee \\ &(\text{Sister}(\text{Felix}, \text{Spot}) \Rightarrow \text{Cat}(\text{Felix})) \vee \\ &(\text{Sister}(\text{Richard}, \text{Spot}) \Rightarrow \text{Cat}(\text{Richard})) \vee \\ &(\text{Sister}(\text{John}, \text{Spot}) \Rightarrow \text{Cat}(\text{John})) \vee \end{aligned}$$

$$\vdots$$

✧Nested quantifiers:

$$\forall x, y \text{ Parent}(x, y) \Rightarrow \text{Child}(y, x).$$

$$\forall x \forall y \text{ Parent}(x, y) \Rightarrow \text{Child}(y, x).$$

$$\forall x, y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(y, x).$$

$$\forall x \exists y \text{ Loves}(x, y)$$

$$\exists y \forall x \text{ Loves}(x, y)$$

The order of quantification is very important.

$\forall x (\exists y P(x, y))$: Every object in the universe has a particular property, namely, the property that is related to *some* object by the relation P.

$\exists x (\forall y P(x, y))$: There is *some* object in the universe that has a particular property, namely, the property that is related to *every* object by the relation P.

scope rule

$$\forall x [Cat(x) \vee (\exists x \textit{Brother}(\textit{Richard},x))]$$

The variable belongs to the innermost quantifier that mentions it; then it will not subject to any other quantification.

well-formed formula

Every variable must be introduced by a quantifier before it is used.

$\forall x P(y)$ is incorrect.

A sentence that have all its variables properly introduced is called a **well-formed formula** or **wff**.

✧ Connections between \forall and \exists

$$\forall x \neg Likes(x, Parsnips) \equiv \neg \exists x Likes(x, Parsnips)$$

$$\forall x Likes(x, IceCream) \equiv \neg \exists x \neg Likes(x, IceCream)$$

\forall is a conjunction, and \exists is a disjunction, they obey De Morgan's law:

$\forall \mathbf{x} \neg \mathbf{P}$	\equiv	$\neg \exists \mathbf{x} \mathbf{P}$	$\neg \mathbf{P} \wedge \neg \mathbf{Q}$	\equiv	$\neg (\mathbf{P} \vee \mathbf{Q})$
$\neg \forall \mathbf{x} \mathbf{P}$	\equiv	$\exists \mathbf{x} \neg \mathbf{P}$	$\neg (\mathbf{P} \wedge \mathbf{Q})$	\equiv	$\neg \mathbf{P} \vee \neg \mathbf{Q}$
$\forall \mathbf{x} \mathbf{P}$	\equiv	$\neg \exists \mathbf{x} \neg \mathbf{P}$	$\mathbf{P} \wedge \mathbf{Q}$	\equiv	$\neg (\neg \mathbf{P} \vee \neg \mathbf{Q})$
$\exists \mathbf{x} \mathbf{P}$	\equiv	$\neg \forall \mathbf{x} \neg \mathbf{P}$	$\mathbf{P} \vee \mathbf{Q}$	\equiv	$\neg (\neg \mathbf{P} \wedge \neg \mathbf{Q})$

Equality

✧ The **equality symbol** can be used to make statements to the effect that two terms refer to the same object. For example:

$$Father(John) = Henry.$$

✧ Thus, the equality is a predicate symbol to refer to the **identity relation**, which is the set of all pairs of same objects, such as:

$$\{ \langle \text{Spot}, \text{Spot} \rangle, \langle \text{Rebecca}, \text{Rebecca} \rangle, \\ \langle \text{King John}, \text{King John} \rangle, \langle \text{Henry II}, \text{Henry II} \rangle, \dots \}$$

✧ The ***Father(John)=Henry*** can be proved by looking at the *functional relation* for ***Father*** to find out if its value is ***Henry***.

✧ The equality symbol can also be used with negation to insist that two terms are not the same object.

For example:

$$\exists x, y \text{ Sister}(\text{Spot}, x) \wedge \text{Sister}(\text{Spot}, y) \wedge \neg(x=y).$$

insists that Spot has *at least two sisters*,
in contrast to

$$\exists x, y \text{ Sister}(\text{Spot}, x) \wedge \text{Sister}(\text{Spot}, y)$$

Extensions and Notational Variations

Three types of alternatives to first-order logic: **higher-order logic**, **first-order logic with abbreviations**, and **notational variations**.

First-order logic can quantify only over *objects*, the first-order entities that actually exist in the world.

Higher-order Logic

Allows quantify over **relations** and **functions** as well as over objects.

Equivalent of two objects:

$$\forall x, y \quad (x=y) \Leftrightarrow (\forall p \, p(x) \Leftrightarrow p(y))$$

Equivalent of two functions:

$$\forall f, g \quad (f=g) \Leftrightarrow (\forall x \, f(x) \Leftrightarrow g(x))$$

✧ More expressive but less understood of how to reason effectively with sentences.

Functional and Predicate expressions using the λ operator.

The λ is traditionally used to term a term into a function. For example, the lambda expression in the LISP.

λ -expression : $\lambda x,y x^2-y^2$

evaluation : $(\lambda x,y x^2-y^2)(25,24) = 25^2-24^2=49$

$\lambda x,y \text{ Gender}(x) \neq \text{Gender}(y) \wedge \text{Address}(x) = \text{Address}(y)$

Means “Two people who are of different gender but of the same address”.

✧ The λ -expression does not increase the formal expressive power of first-order logic.

The Uniqueness Quantifier $\exists!$.

This abbreviation is a concise way to express that *an unique object satisfying some predicate exists*.

$\exists! x \text{ King}(x)$ means

“There exists a *unique* object x satisfying $\text{King}(x)$ ”

$\equiv \exists x \text{ King}(x) \wedge \forall y \text{ King}(y) \Rightarrow x=y$

A more complex example:

$\forall c \text{ Country}(c) \Rightarrow \exists! r \text{ Ruler}(r, c)$

“For every country , there exists exactly one ruler”

The Uniqueness Operator ι (iota)

The operator can be used to refer to the unique object as $\iota x P(x)$ directly.

To say “*the unique ruler of Freedonia is dead*” by
 $Dead(\iota r Ruler(r, Freedonia))$ is an abbreviation of
 $\equiv \exists! r Ruler(r, Freedonia) \wedge Dead(r)$

Notational Variations

Other notations have been developed for using first-order logic in the fields other than the AI:

Syntax item	This book	Others
Negation (not)	$\neg P$	$\sim P$ \overline{P}
Conjunction (and)	$P \wedge Q$	$P \& Q$ $P \cdot Q$ PQ P, Q
Disjunction (or)	$P \vee Q$	$P \mid Q$ $P; Q$ $P + Q$
Implication (if)	$P \Rightarrow Q$	$P \rightarrow Q$ $P \supset Q$
Equivalence (iff)	$P \Leftrightarrow Q$	$P \equiv Q$ $P \leftrightarrow Q$
Universal (all)	$\forall x P(x)$	$(\forall x)P(x) \wedge xP(x)$ $P(x)$
Existential (exist)	$\exists x P(x)$	$(\exists x)P(x) \vee xP(x)$ $P(\text{Skolem}_i)$
Relation	$R(x, y)$	$(R \ x \ y)$ Rxy xRy

✧ In Prolog:

- ✓ It uses uppercase letters for variables and lowercase for constants.
- ✓ The order of the implication is reversed as $Q :- P$.
- ✓ Uses ‘,’ for conjunction, e.g. $R :- P, Q$.
- ✓ **cat(X) :- furry(X), meows(X), has(X, claws).**

✧ In Lisp

- ✓ Variables are distinguished by the prefix \$, or ?.
- ✓ Connectives, just like the predicate and function symbols, come first.
- ✓ Sentences and nonconstant terms are surrounded by parentheses.
- ✓ **(forall ?x**
(implies (and (furry ?x) (meows ?x) (has ?x claws))
(cat ?x)))

Using First-Order logic

Using first-order logic to represent objects and their relationships in some simple **domains** - a section of the world.

A kinship domain

The domain of family relationships, or kinship.

$$\forall m, c \text{ Mother}(c)=m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c).$$

$$\forall w, h \text{ Husband}(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h, w).$$

$$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x).$$

$$\forall p, c \text{ parent}(p, c) \Leftrightarrow \text{Chile}(c, p).$$

$$\forall g, c \text{ Grandparent}(g, c) \Leftrightarrow \exists p \text{ Parent}(g, p) \wedge \text{Parent}(p, c).$$

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p \text{ Parent}(p, x) \wedge \text{Parent}(p, y).$$

Axiom, Definitions, and Theorems

✧ **In mathematic**

Axioms : capture the basic facts about a diomain.

Definitions : define other concepts using axioms.

Theorems : Proved by using axioms and
definitions.

Independent Axiom : a axiom that can not be derived from all
the other axioms.

✧ In Artificial Intelligence

1. What is the minimum set of axioms for fully specifying a domain?

In the kinship domain, *Child*, *Spouse*, *Male*, and *Female* would be reasonable candidates for basic predicate for defining other predicates.

2. Are there too many basic sentences?

For example : Is it necessary to have:

$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$ because $\text{Sibling}(\text{John}, \text{Richard})$ can be inferred as:

$$\begin{aligned}
 & (\exists p \text{ Parent}(p, \text{John}) \wedge \text{Parent}(p, \text{Richard}) \Leftrightarrow \\
 & \quad \exists p \text{ Parent}(p, \text{Richard}) \wedge \text{Parent}(p, \text{John})) \\
 & \Rightarrow \text{Sibling}(\text{Richard}, \text{John}).
 \end{aligned}$$

However, in **AI** it is common to include redundant axioms to improve **efficiency**.

In AI, an axiom of the form $\forall x, y P(x,y) \Leftrightarrow$ is often called a **definition** of P.

It is sometimes impossible to have a complete definition for a predicate, e.g.

$$\forall x \text{ Person}(x) \Leftrightarrow \dots$$

Instead, the followings are often used:

$$\forall x \text{ Person}(x) \Rightarrow \dots \quad \text{or}$$

$$\forall x \dots \Rightarrow \text{Person}(x)$$

The Domain of Sets

Use mathematic sets domain as an example.

EmptySet is a constant.

Member and *Subset* are predicates.

Intersection, *Union*, and *Adjoin* are functions

Set is a predicate that is true only of sets.

✧ The basic axioms:

1. The only sets are the empty set and those made by adjoining something to it.

$$\forall s \text{ Set}(s) \Leftrightarrow (s = \text{EmptySet}) \vee (\exists x, s_2 \text{ Set}(s_2) \wedge s = \text{Adjoin}(x, s_2))$$

2. The empty set has no elements adjoined into it.

$$\neg \exists x, s \text{ Adjoin}(x, s) = \text{EmptySet}$$

3. Adjoining an element already in a set has no effect.

$$\forall x, s \text{ Member}(x, s) \Leftrightarrow s = \text{Adjoin}(x, s)$$

4. The only members of a set are the elements that were adjoined into it.

$$\begin{aligned} \forall x, s \text{ Member}(x, s) \Leftrightarrow \\ \exists y, s_2 (s = \text{Adjoin}(y, s_2) \wedge (x=y \vee \text{Member}(x, s))) \end{aligned}$$

5. A set is a subset of another if and only if all of the first set's members are members of the second set.

$$\begin{aligned} \forall s_1, s_2 \text{ Subset}(s_1, s_2) \Leftrightarrow \\ (\forall x \text{ Member}(x, s_1) \Rightarrow \text{Member}(x, s_2)) \end{aligned}$$

6. Two sets are equal if and only if each is a subset of the other.

$$\forall s_1, s_2 (s_1 = s_2) \Leftrightarrow (\textit{Subset}(s_1, s_2) \wedge \textit{Subset}(s_2, s_1))$$

7. An object is a member of the intersection of two sets if and only if it is a member of each of the sets.

$$\forall x, s_1, s_2 \textit{Member}(x, \textit{Intersection}(s_1, s_2)) \Leftrightarrow \\ \textit{Member}(x, s_1) \wedge \textit{Member}(x, s_2)$$

8. An object is a member of the union of the two sets if and only if it is a member of either set.

$$\forall x, s_1, s_2 \textit{Member}(x, \textit{Union}(s_1, s_2)) \Leftrightarrow \\ \textit{Member}(x, s_1) \vee \textit{Member}(x, s_2)$$

✧ **Exercise:** Write Axioms for the list domain.

Nil : empty List

Cons, *Append*, *First*, and *Rest* : functions.

Find and *List?* predicates (as *Member* and *Set*)

Special Notations for Sets, Lists, and Arithmetic

Extend the first-order logic to include *set* and *list* notation by using some **syntactic sugar** - abbreviations in *syntax* without changing the *semantics*.

For Sets

$$\emptyset = \text{EmptySet}$$

$$\{x\} = \text{Adjoin}(x, \text{EmptySet})$$

$$\{x,y\} = \text{Adjoin}(x, \text{Adjoin}(y, \text{EmptySet}))$$

$$\{x,y/s\} = \text{Adjoin}(x, \text{Adjoin}(y,s))$$

$$r \cup s = \text{Union}(r,s)$$

$$r \cap s = \text{Intersection}(r,s)$$

$$x \in s = \text{Member}(x,s)$$

$$r \subseteq s = \text{Subset}(r,s)$$

For Lists

$$[] = \text{Nil}$$

$$[x] = \text{Cons}(x, \text{Nil})$$

$$[x,y] = \text{Cons}(x, \text{Cons}(y, \text{Nil}))$$

$$[x,y/l] = \text{Cons}(x, \text{Cons}(y,l))$$

Asking Questions and Getting Answers

First assume the two utilities **TELL** and **ASK** which are used in FOL.

✧ Adding **Assertions**:

$\text{TELL}(\text{KB}, (\forall m, c \text{ Mother}(c)=m \Leftrightarrow \text{Female}(m) \wedge \text{Parent}(m, c))$

✧ Making **Queries** or **Goals**:

After adding the **assertion**

$\text{TELL}(\text{KB}, (\text{Female}(\text{Maxi}) \wedge \text{Parent}(\text{Maxi}, \text{Spot}) \wedge$
 $\text{Parent}(\text{Spot}, \text{Boots})))$

The following **query** can be made

$\text{ASK}(\text{KB}, \text{Grandparent}(\text{Maxi}, \text{Boots}))$

✧ **Substitution** or **binding** list as answers:

$\text{ASK}(\text{KB}, \exists x \text{ Child}(x, \text{Spot}))$

should have an answer like:

$\{x/\text{Boots}, \dots\}$



Logic Agents For The Wumpus World

Three agent architectures are considered : **Reflex agents**, **Model-based agents**, and **Goal-based agents**.

Percept sequence (with *time* record):

Percept([Stench, Breeze, Glitter, None, None], 5)

Possible actions: *Turn(Right), Turn(Left), Forward,*
Shoot, Grab, Release, Climb.

Query Format: $\exists a \text{ Action}(a, 5)$

with *binding list* such as $\{a/\text{Grab}\}$

```
function KB-AGENT(percept) returns an action  
  static: KB, a knowledge base  
           t, a counter, initially 0, indicating time  
  
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))  
  action  $\leftarrow$  ASK(KB, MAKE-ACTION-QUERY(t))  
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))  
  t  $\leftarrow$  t + 1  
  return action
```

Figure 2: A generic knowledge-based agent.



A Simple Reflex Agent

The simplest possible kind of agent, which has rules directly connecting percepts to actions. The rules resemble reflexes or instincts.

For example: *See a glitter, grab the gold*

$$\forall s,b,u,c,t \text{ Percept}([s,b,Glitter,u,c],t) \Rightarrow \text{Action}(Grab,t).$$

More useful and flexible mediating forms for the connection between percepts and actions:

$$\forall s,b,u,c,t \text{ Percept}([s,b,Glitter,u,c],t) \Rightarrow \text{AtGold}(t).$$
$$\forall s,b,u,c,t \text{ Percept}([s,Breeze,g,u,c],t) \Rightarrow \text{Breeze}(t).$$
$$\forall s,b,u,c,t \text{ Percept}([Stench,b,g,u,c],t) \Rightarrow \text{Stench}(t).$$

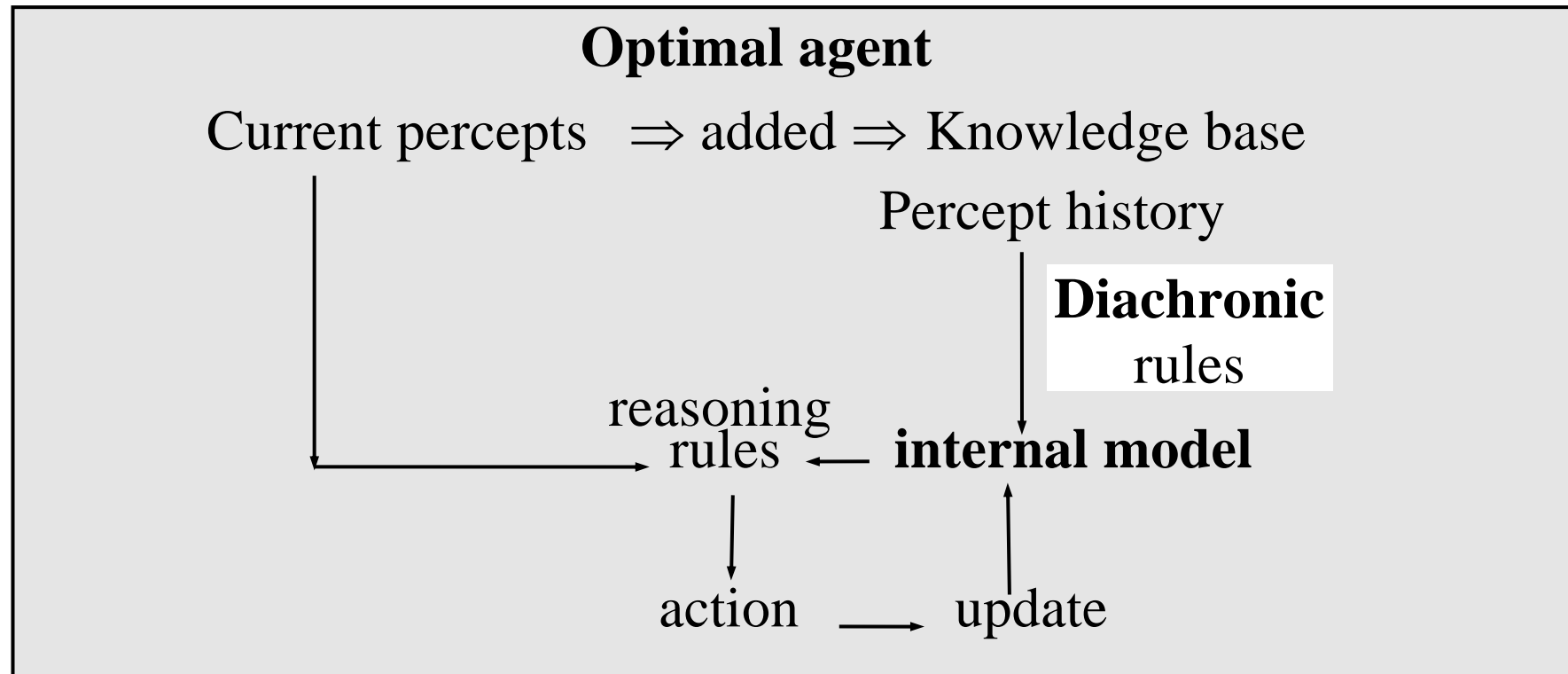
And the connection becomes

$$\forall t \text{ AtGold}(t) \Rightarrow \text{Action}(Grab,t)$$

Limitations of Simple Reflex Agents

- ✧ A pure reflex agent cannot know for sure *when* to perform an action! It does not have the representation of the world. For example, *it does not know if it is carrying the gold*.
- ✧ Reflex agents are also unable to avoid *infinite loops*. They might do exactly what they did before when re-entering a place and receiving the same percepts.

+ Representing Change in The World



Any system that makes decisions on the basis of past percepts can be rewritten to use instead a set of sentences about the *current world state*.

Rules describing the way in which the world changes are called **diachronic** rules.

✧ Two possibilities to deal with change:

1. To change the knowledge base by replacing the old sentence with the latest new sentence.

All the knowledge about the past is lost, thus, can answer the question about the latest situation.

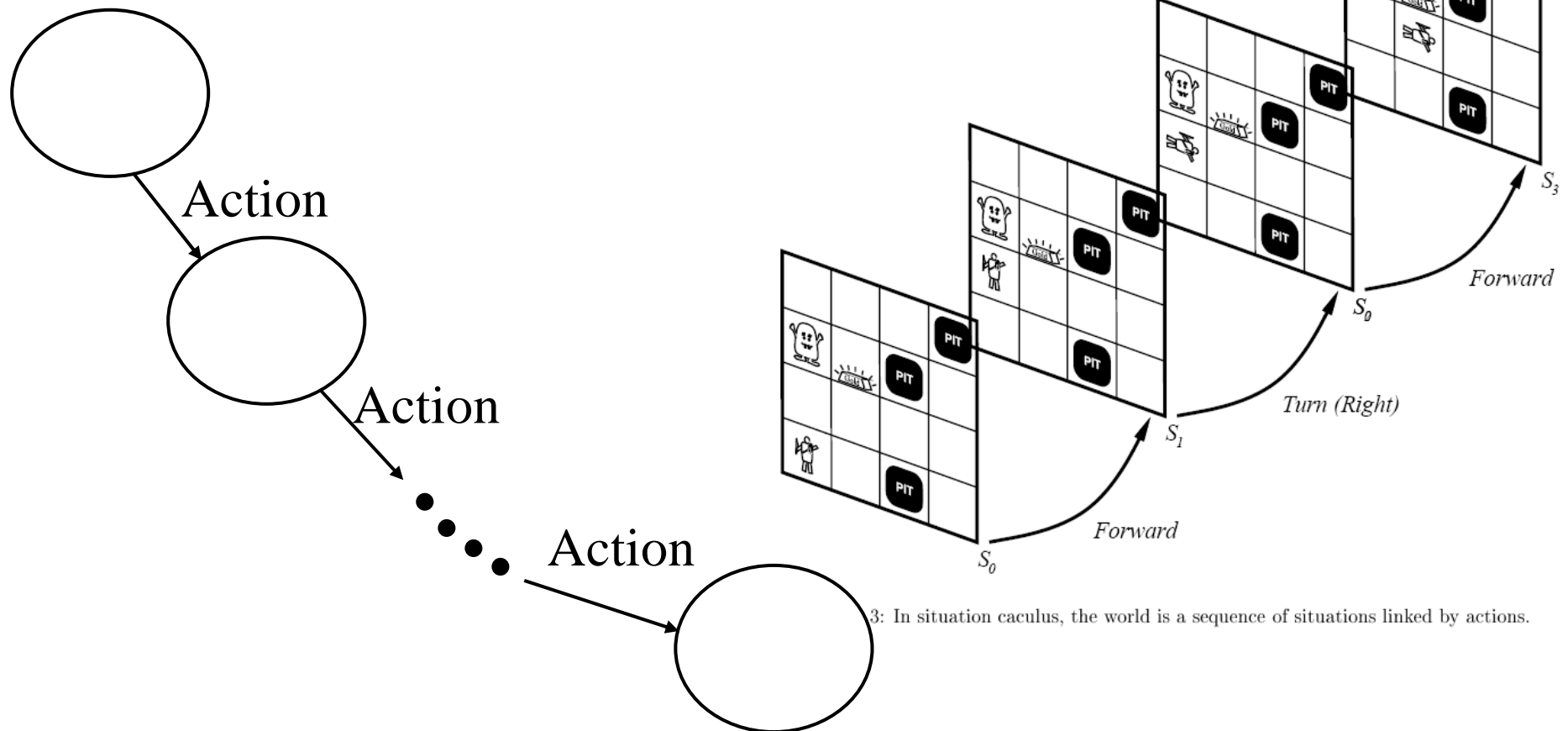
2. To **search** through a space of past and possible states, where each state is represented by a different knowledge base.

Can explore hypothetical situations by switching attention between several knowledge bases.

- ✧ More oftenly, it is required to represent different situations and actions in the same knowledge base.
- ✧ In principle, *representing situations and actions is no different from representing more concrete objects, or concrete relations.*

Situation Calculus

A particular way of describing the changes in FOL; Situations are generated from previous situations by actions.



3: In situation calculus, the world is a sequence of situations linked by actions.

A world is a sequence of situations, snapshots of the world

✧ Relations and properties that can change over time are handled by giving an extra **situation argument** to each corresponding predicates.

✧ For example, use

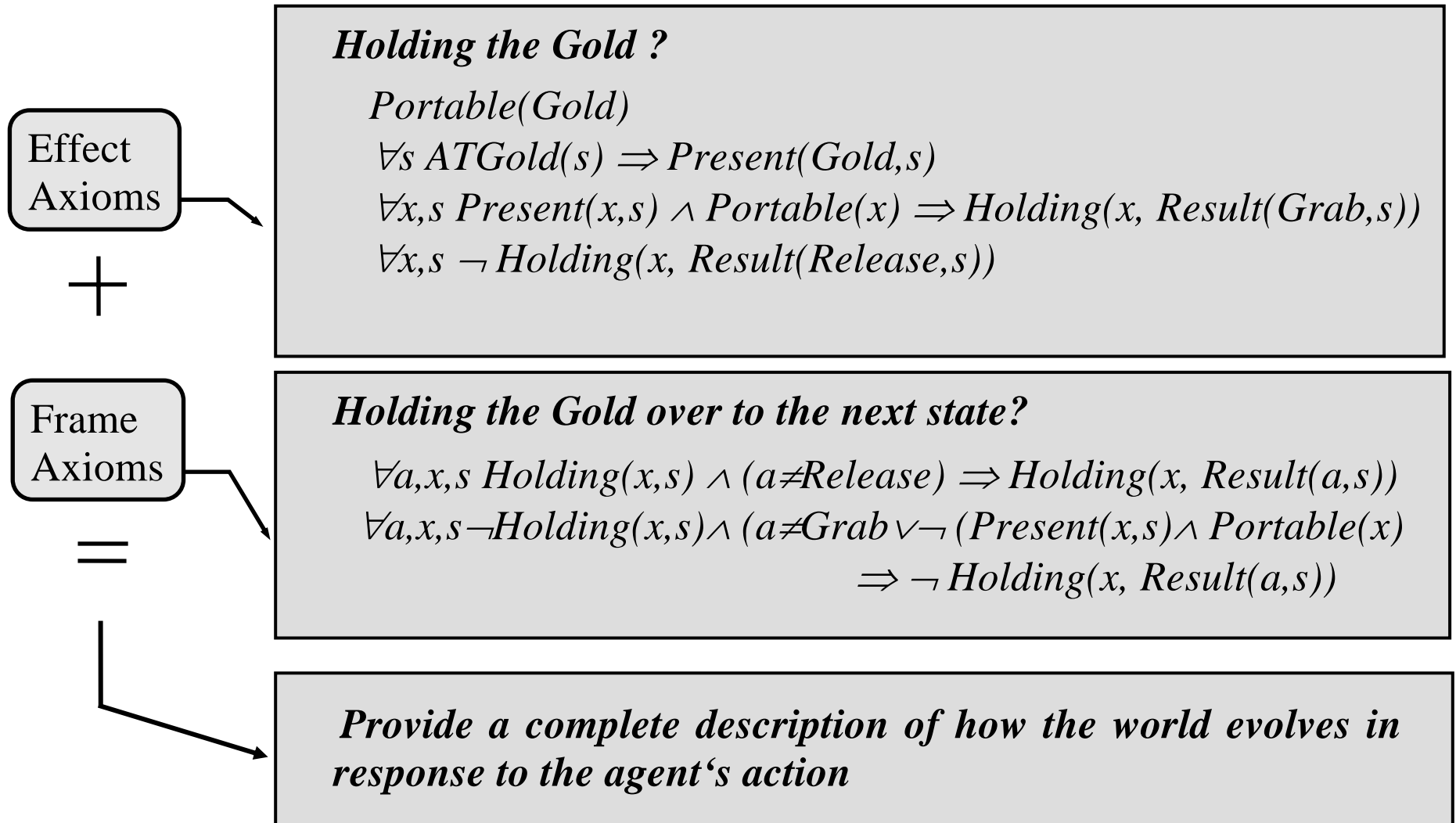
$At(Agent, [1,1], S_0) \wedge At(agent, [1,2], S_1)$ instead of
 $At(Agent\ location)$

✧ The function ***Result(action, situation)*** is used to denote the situation that results from performing an action in some initial situation. For example:

$Result(Forward, S_0) = S_1$
 $Result(Turn(Right), S_1) = S_2$
 $Result(Forward, S_2) = S_3$

✧ Actions are described by stating their effects, i.e. *specifying the*

properties of the situation that results from doing the action. For example:



- ✧ The combined axiom (**successor-state axiom**) for describing the evolution of the world:

true afterwards \Leftrightarrow
[an action made it true
 \vee true already and no action made it false]

For example:

$\forall a, x, s \text{ Holding}(x, \text{Result}(a, s)) \Leftrightarrow$
[(a=Grab \wedge Present(x,s) \wedge Portable(x))
 \vee (Holding(x, s) \wedge a \neq Release)]

- ✧ One successor-state axiom must be specified for each predicate that may change its value over time.

Keeping Track of Location

In the wumpus world it is important for an agent to remember *where it has been* and *what it saw now* to make right decision.

✧ Except for the initial location $At(\textit{Agent}, [1,1], S_0)$, the agent also needs to know:

1. What direction it is facing:

$$\textit{Orientation}(\textit{Agent}, S_0) = 0$$

2. How locations are arranged(a simple “map”):

The map

$x,y \text{ LocationToward}([x,y], 0) = [x+1, y]$

$x,y \text{ LocationToward}([x,y], 90) = [x, y+1]$

$x,y \text{ LocationToward}([x,y], 180) = [x-1, y]$

$x,y \text{ LocationToward}([x,y], 270) = [x, y-1]$

Square ahead of an agent

$\forall p,l,s \text{ At}(p,l,s) \Rightarrow \text{LocationAhead}(p,s) =$
 $\text{LocationToward}(l, \text{Orientation}(p,s))$

Adjacent Squares

$\forall l_1, l_2 \text{ Adjacent}(l_1, l_2) \Leftrightarrow$
 $\exists d, l_1 = \text{LocationToward}(l_2, d)$

3. Whatever is known about the contents of the locations (geographical details on the map):

$$\forall \mathbf{x}, \mathbf{y} \text{ Wall}([\mathbf{x}, \mathbf{y}]) \Leftrightarrow (\mathbf{x}=\mathbf{0} \vee \mathbf{x}=\mathbf{5} \vee \mathbf{y}=\mathbf{0} \vee \mathbf{y}=\mathbf{5})$$

4. What the actions do to the locations:


$$\begin{aligned} \forall a, d, p, s \text{ At}(p, l, \text{Result}(a, s)) \Leftrightarrow \\ [(a = \text{Forward} \wedge l = \text{LocationAhead}(p, s) \wedge \neg \text{Wall}(l) \\ \vee (\text{at}(p, l, s) \wedge a \neq \text{Forward}))] \end{aligned}$$

5. What the actions do to the orientations:

$$\begin{aligned} \forall a, d, p, s \text{ Orientation}(p, \text{Result}(a, s)) = d \Leftrightarrow \\ [(a = \text{Turn(Right)} \wedge d = \text{Mod}(\text{Orientation}(p, s) - 90, 360)) \\ \vee (a = \text{Turn(Left)} \wedge d = \text{Mod}(\text{Orientation}(p, s) + 90, 360)) \\ \vee (\text{Orientation}(p, s) = d \wedge \neg (a = \text{Turn(Right)} \vee a = \text{Turn(Left)}))] \end{aligned}$$

Deducing Hidden Properties Of The World

Abstracted or hidden qualities (properties) of a world can also be associated with the locations.

$$\forall l, s \text{ At}(\text{Agent}, l, s) \wedge \text{Breeze}(s) \Rightarrow \text{Breezy}(l)$$
$$\forall l, s \text{ At}(\text{Agent}, l, s) \wedge \text{Stench}(s) \Rightarrow \text{Smelly}(l)$$


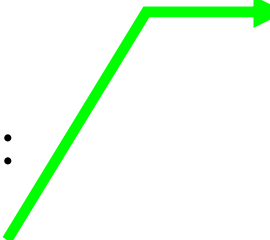
These properties are used to deduced the right situation of the world, and then the correct actions for the agent.

For example “where the pits are”, “where the wumpus is”, and “which squares are safe to move to?”

Synchronic Rules

Relate properties of a world state to other properties of the **same** world state to capture the necessary information for the desired deductions.

✧ Synchronic rules:



These rules are used by **model-based reasoning** systems (agents)

1. Causal rules:

Reflect the assumed direction of causality in the world: some hidden property of the world causes certain percepts to be generated.

$$\forall l_1, l_2, s \text{ AT}(\textit{Wumpus}, l_1, s) \wedge \textit{Adjacent}(l_1, l_2) \Rightarrow \textit{Smelly}(l_2)$$

$$\forall l_1, l_2, s \text{ AT}(\textit{Pit}, l_1, s) \wedge \textit{Adjacent}(l_1, l_2) \Rightarrow \textit{Breezy}(l_2)$$

2. Diagnostic rules:

Diagnostic reasoning

Infer the presence of hidden properties directly from the **percept-derived information**, such as:

$$\forall l, s \text{ At}(\text{Agent}, l, s) \wedge \text{Breeze}(s) \Rightarrow \text{Breezy}(l)$$

$$\forall l, s \text{ At}(\text{Agent}, l, s) \wedge \text{Stench}(s) \Rightarrow \text{Smelly}(l)$$

Generally, diagnostic rules can draw a **weak** conclusion only, For example:

$$\forall l_1, s \text{ Smelly}(l_1) \Rightarrow$$

$$\exists l_2 \text{ At}(\text{Wumpus}, l_2, s) \wedge (l_2 = l_1 \vee \text{Adjacent}(l_1, l_2))$$

It is tricky to derive the **strongest** possible conclusions from the diagnostic rules:

$$\forall x,y,g,u,c,s \text{ Percept}([None, None, g, u, c], t) \wedge \\ At(Agent, x, s) \wedge Adjacent(x, y) \\ \Rightarrow OK(y)$$

But sometimes a square can be OK even when smells and breezes around.

Perhaps the best way to represent safety is by using model-based rule: $\forall x, t (\neg At(Wumpus, x, t) \wedge \neg Pit(x)) \Leftrightarrow OK(x)$

✧ If the axioms **correctly** and **completely** describe the way the world works and the way that percepts are produced, then *the inference procedure will correctly infer the **strongest possible description** of the world state given the available percepts.*

Preferences Among Actions

- ✧ It is usually not enough only to write rules recommending actions on the basis of certain conditions in the world.
- ✧ Actions are generally classified by the **desirability**, and the inference engine should choose whichever the action that has the highest desirability.

This practice will make the rules **not modular**: *that is changing in the agent's beliefs about some aspects of the world would require changes in rules dealing with other aspects also.*

A more modular way is to separate facts about actions from facts about goals, which means the agent can be reprogrammed simply by asking it to achieve something different.

✧ Examples:

$$\forall a,s \text{ Great}(a,s) \Rightarrow \text{Action}(a,s)$$

$$\forall a,s \text{ Good}(a,s) \wedge (\neg \exists b \text{ Great}(b,s)) \Rightarrow \text{Action}(a,s)$$

$$\forall a,s \text{ Medium}(a,s) \wedge (\neg \exists b \text{ Great}(b,s) \vee \text{Good}(b,s)) \Rightarrow \text{Action}(a,s)$$

$$\forall a,s \text{ Risky}(a,s) \wedge (\neg \exists b \text{ Great}(b,s) \vee \text{Good}(b,s) \vee \text{Medium}(a,s)) \Rightarrow \text{Action}(a,s)$$

Great actions: picking up the gold when found, climbing out of the cave with the gold.

Good actions: moving to a square that is OK and has not yet been visited.

Medium actions: moving to a square that is OK and has already been visited.

Risky actions: moving to a square that is not known to be deadly, but is not known to be OK either.

Deadly actions: moving to a square that is known to contain a pit or a live wumpus.



Toward A Goal-Based Agent

✧ For such agents, their actions are sometimes changed radically because of the changes of the goal.

For example,

$$\forall s \text{ Holding}(\text{Gold}, s) \Rightarrow \text{GoalLocation}([1,1], s)$$

✧ The presence of an **explicit goal** allows the agent to work out a sequence of actions that will achieve the goal.

✧ Three ways to find such a sequence.

1. **Inference:** To write axioms that will allow us to ASK the KB for a sequence of actions that is guaranteed to achieve the goal safely. Not feasible for a larger world.
2. **Search:** To use a best-first search procedure to find a path to the goal. This requires translating the knowledge into a set of operators, and accompanying state representation, for the search algorithm.
3. **Planning:** Involving using special-purpose reasoning systems designed to reason about actions.