Artificial Intelligence

Logical AI

Christopher Simpkins

Logic and AI

In AI, **knowledge-based** agents use a process of **reasoning** over an internal **representation** of knowledge to decide what actions to take.

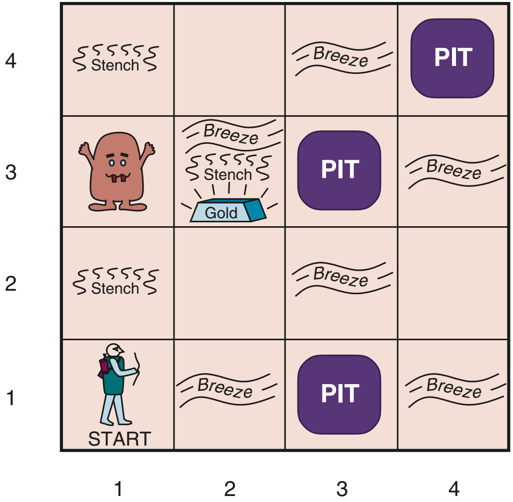
Knowledge-Based Agents

function KB-AGENT(*percept*) **returns** an *action*
  persistent: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  $t \leftarrow t + 1$
  return *action*

KENNESAW STATE UNIVERSITY

The Wumpus World



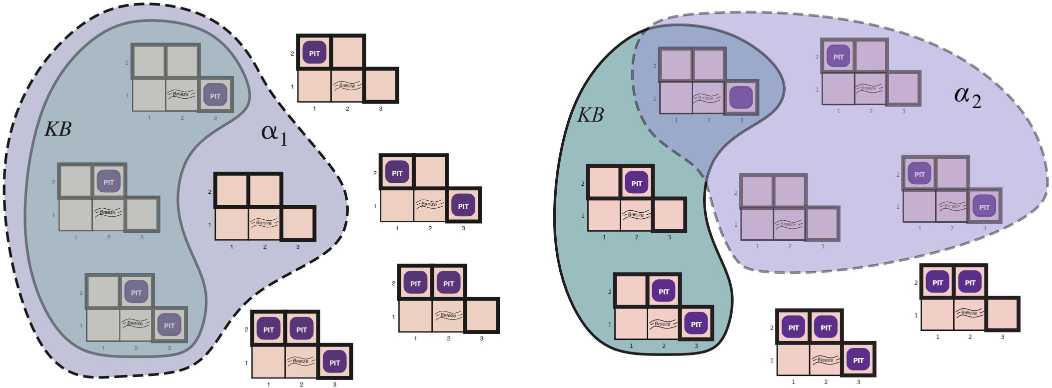
First Steps Wumpus WOrld

(a) (b)

Later Steps Wumpus World

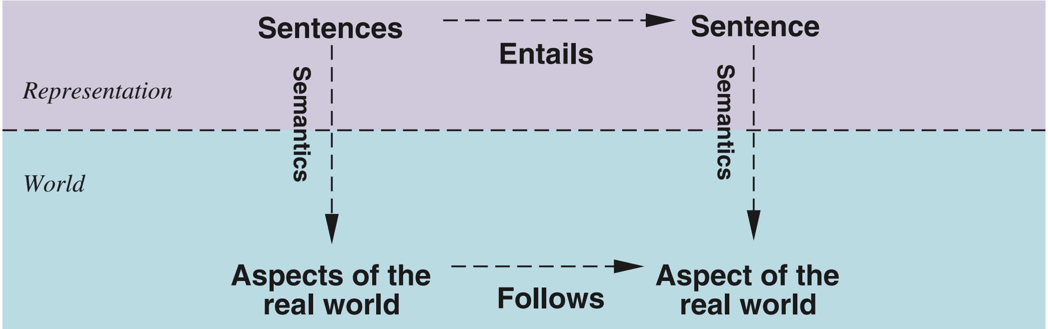
(a) (b)

Logic



(a) (b)

Logic



Sentences
Entails
Sentence
Representation
Semantics
Semantics
World
Aspects of the real world
Follows
Aspect of the real world

Propositional Logic

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & (\, \textit{Sentence} \,) \\
& \mid & \neg\, \textit{Sentence} \\
& \mid & \textit{Sentence} \land \textit{Sentence} \\
& \mid & \textit{Sentence} \lor \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Rightarrow\; \textit{Sentence} \\
& \mid & \textit{Sentence} \;\Leftrightarrow\; \textit{Sentence} \\
& & \\
\textsc{Operator Precedence} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

KENNESAW STATE UNIVERSITY

Propositional Logic

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	$\underline{\textit{true}}$
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	$\underline{\textit{true}}$
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	$\underline{\textit{true}}$
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Propositional Theorem Proving

function TT-ENTAILS?(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $symbols \leftarrow$ a list of the proposition symbols in KB and α
  return TT-CHECK-ALL($KB, \alpha, symbols, \{\,\}$)

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** *true* or *false*
  if EMPTY?($symbols$) **then**
    if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)
    else return *true*   // when KB is false, always return true
  else
    $P \leftarrow$ FIRST($symbols$)
    $rest \leftarrow$ REST($symbols$)
    return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
      and
      TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

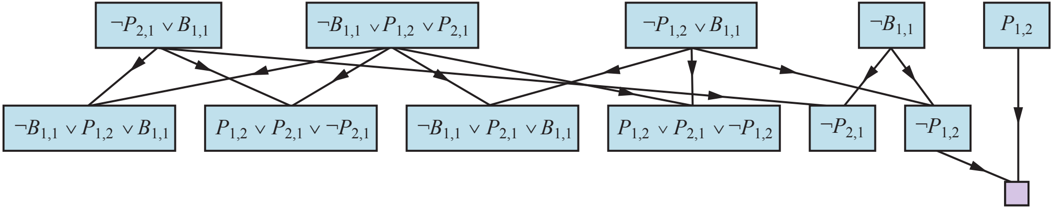
$$
\begin{aligned}
CNFSentence &\rightarrow Clause_1 \wedge \cdots \wedge Clause_n \\
Clause &\rightarrow Literal_1 \vee \cdots \vee Literal_m \\
Fact &\rightarrow Symbol \\
Literal &\rightarrow Symbol \mid \neg Symbol \\
Symbol &\rightarrow P \mid Q \mid R \mid \ldots \\
HornClauseForm &\rightarrow DefiniteClauseForm \mid GoalClauseForm \\
DefiniteClauseForm &\rightarrow Fact \mid (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow Symbol \\
GoalClauseForm &\rightarrow (Symbol_1 \wedge \cdots \wedge Symbol_l) \Rightarrow False
\end{aligned}
$$

Propositional Theorem Proving

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: *KB*, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \land \neg\alpha$
  new $\leftarrow \{\,\}$
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents $\leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new $\leftarrow$ *new* $\cup$ *resolvents*
    if *new* $\subseteq$ *clauses* **then return** *false*
    clauses $\leftarrow$ *clauses* $\cup$ *new*

Propositional Theorem Proving



Propositional Theorem Proving

function PL-FC-ENTAILS?(*KB*, *q*) **returns** *true* or *false*
  inputs: *KB*, the knowledge base, a set of propositional definite clauses
    q, the query, a proposition symbol
  count ← a table, where *count*[*c*] is initially the number of symbols in clause *c*'s premise
  inferred ← a table, where *inferred*[*s*] is initially *false* for all symbols
  queue ← a queue of symbols, initially symbols known to be true in *KB*

  while *queue* is not empty **do**
    p ← POP(*queue*)
    if *p* = *q* **then return** *true*
    if *inferred*[*p*] = *false* **then**
      inferred[*p*] ← *true*
      for each clause *c* in *KB* where *p* is in *c*.PREMISE **do**
        decrement *count*[*c*]
        if *count*[*c*] = 0 **then** add *c*.CONCLUSION to *queue*
  return *false*

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

$$P \Rightarrow Q$$

$$L \wedge M \Rightarrow P$$

$$B \wedge L \Rightarrow M$$

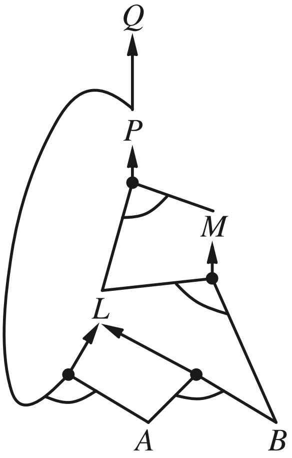
$$A \wedge P \Rightarrow L$$

$$A \wedge B \Rightarrow L$$

$$A$$

$$B$$

(a)



(b)

Propositional Model Checking

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, { })

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

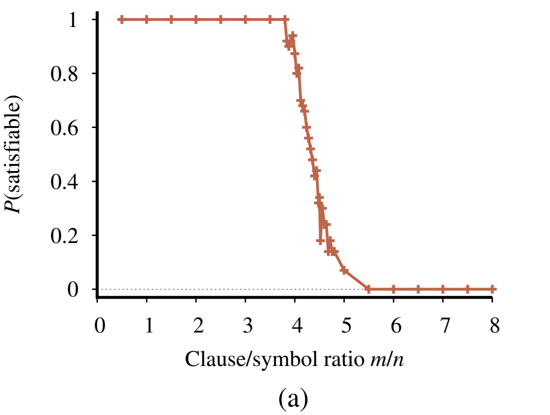
  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
        DPLL(*clauses*, *rest*, *model* ∪ {P=*false*})

KENNESAW STATE UNIVERSITY

Propositional Model Checking

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move, typically around 0.5
          max_flips, number of value flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for each i = 1 to max_flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      if RANDOM(0, 1) ≤ p then
          flip the value in model of a randomly selected symbol from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

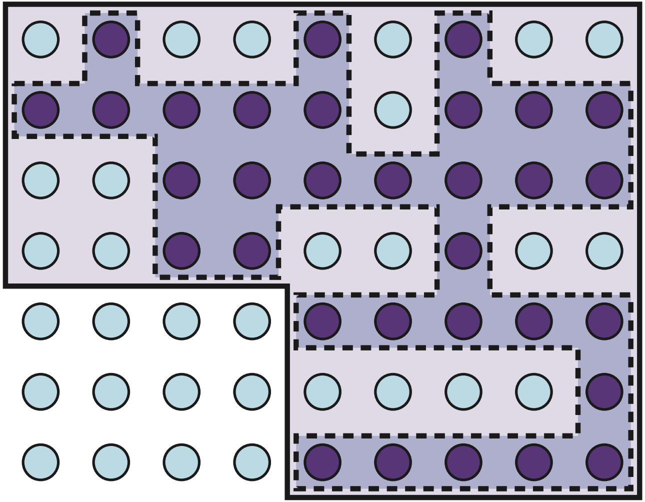
Propositional Model Checking



Agents Based on Propositional Logic

```
function HYBRID-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench,breeze,glitter,bump,scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, initially 0, indicating time
              plan, an action sequence, initially empty

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x,y] : ASK(KB, OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
    plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
    unvisited ← {[x,y] : ASK(KB, L^{t'}_{x,y}) = false for all t' ≤ t}
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, HaveArrow^t) = true then
    possible_wumpus ← {[x,y] : ASK(KB, ¬ W_{x,y}) = false}
    plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then        // no choice but to take a risk
    not_unsafe ← {[x,y] : ASK(KB, ¬ OK^t_{x,y}) = false}
    plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
    plan ← PLAN-ROUTE(current, {[1, 1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action

function PLAN-ROUTE(current,goals,allowed) returns an action sequence
  inputs: current, the agent's current position
          goals, a set of squares; try to plan a route to one of them
          allowed, a set of squares that can form part of the route

  problem ← ROUTE-PROBLEM(current, goals,allowed)
  return SEARCH(problem)        // Any search algorithm from Chapter 3
```

Agents Based on Propositional Logic



Agents Based on Propositional Logic

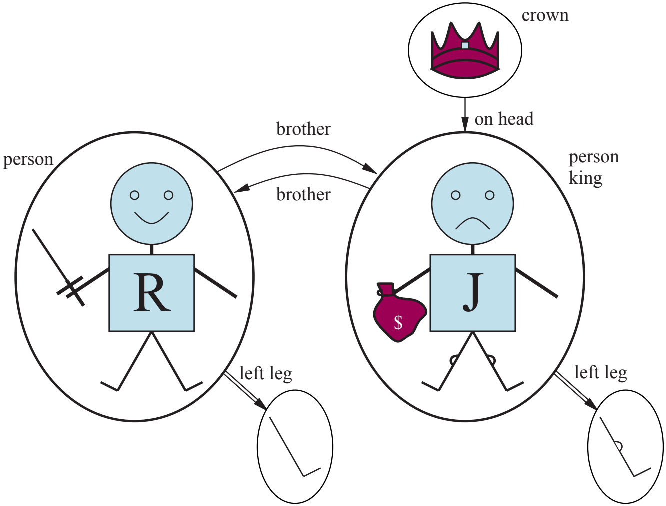
```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
   inputs: init, transition, goal, constitute a description of the problem
           T_max, an upper limit for plan length

   for t = 0 to T_max do
      cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
      model ← SAT-SOLVER(cnf)
      if model is not null then
         return EXTRACT-SOLUTION(model)
   return failure
```

KENNESAW STATE UNIVERSITY

Representation

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

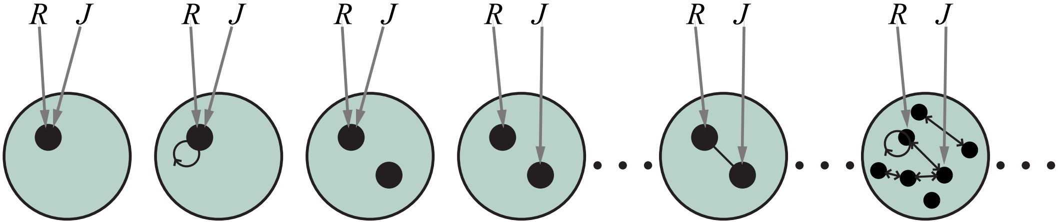
Syntax and Semantics of First-Order Logic



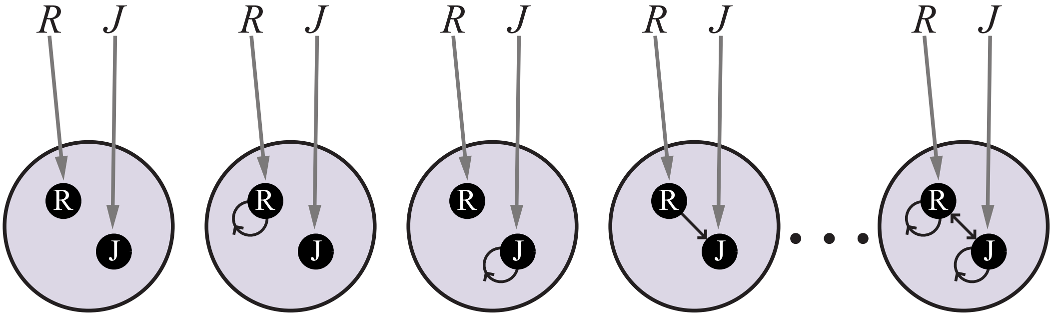
Syntax and Semantics of First-Order Logic

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate} \mid \mathit{Predicate}(\mathit{Term}, \ldots) \mid \mathit{Term} = \mathit{Term} \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Rightarrow\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Leftrightarrow\ \mathit{Sentence} \\
 & \mid & \mathit{Quantifier}\ \mathit{Variable}, \ldots\ \mathit{Sentence} \\
 & & \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term}, \ldots) \\
 & \mid & \mathit{Constant} \\
 & \mid & \mathit{Variable} \\
 & & \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid X_1 \mid \mathit{John} \mid \cdots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\mathit{Predicate} & \rightarrow & \mathit{True} \mid \mathit{False} \mid \mathit{After} \mid \mathit{Loves} \mid \mathit{Raining} \mid \cdots \\
\mathit{Function} & \rightarrow & \mathit{Mother} \mid \mathit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Syntax and Semantics of First-Order Logic



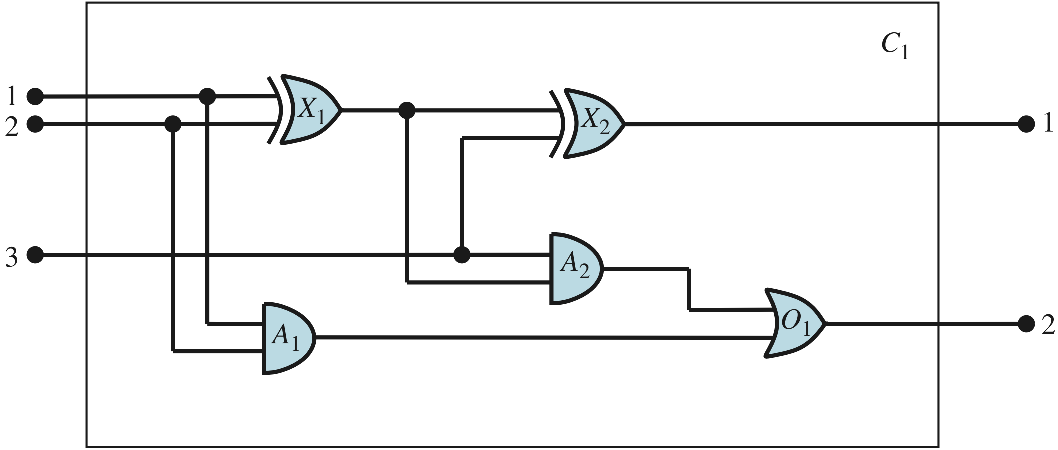
Syntax and Semantics of First-Order Logic



Using First-Order Logic

Foo

Knowledge Engineering in First-Order Logic



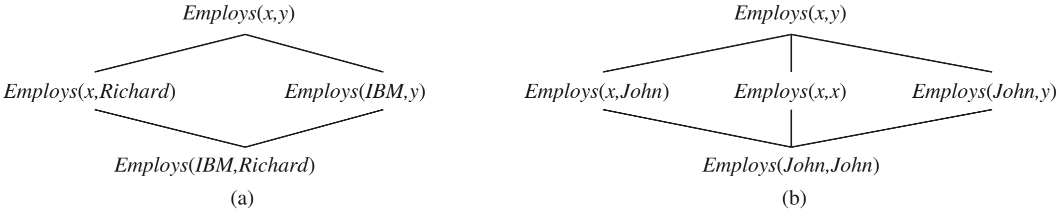
Propositional vs. First-Order Logc

Foo

Unification and First-Order Inference

```
function UNIFY(x, y, θ=empty) returns a substitution to make x and y identical, or failure
   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS(x), ARGS(y), UNIFY(OP(x), OP(y), θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST(x), REST(y), UNIFY(FIRST(x), FIRST(y), θ))
   else return failure

function UNIFY-VAR(var, x, θ) returns a substitution
   if {var/val} ∈ θ for some val then return UNIFY(val, x, θ)
   else if {x/val} ∈ θ for some val then return UNIFY(var, val, θ)
   else if OCCUR-CHECK?(var, x) then return failure
   else return add {var/x} to θ
```

Unification and First-Order Inference



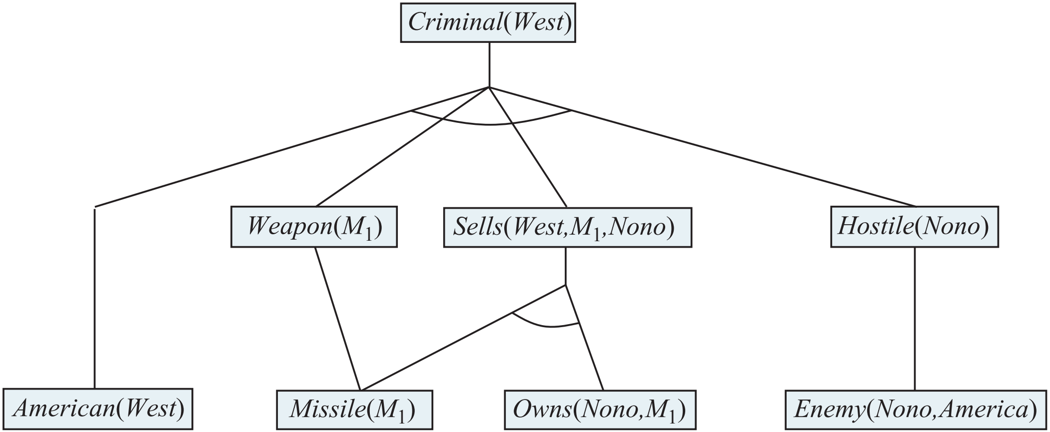
Forward Chaining

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: *KB*, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence

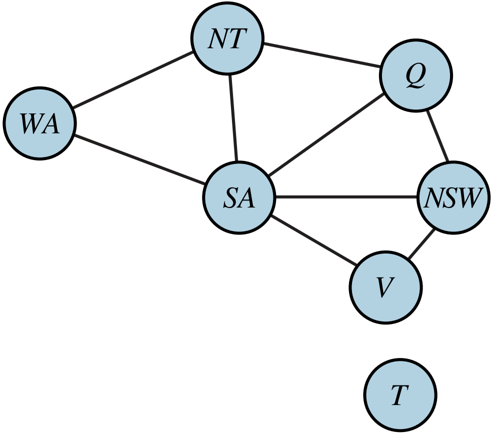
  while *true* **do**
    new $\leftarrow \{\}$ *// The set of new sentences inferred on each iteration*
    for each *rule* **in** *KB* **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in *KB*
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in *KB* or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *failure* **then return** ϕ
    if *new* $= \{\}$ **then return** *false*
    add *new* to *KB*

KENNESAW STATE UNIVERSITY

Forward Chaining



Forward Chaining



(a)

$$
\begin{aligned}
& \mathit{Diff}(wa, nt) \wedge \mathit{Diff}(wa, sa) \wedge \\
& \quad \mathit{Diff}(nt, q) \wedge \mathit{Diff}(nt, sa) \wedge \\
& \quad \mathit{Diff}(q, nsw) \wedge \mathit{Diff}(q, sa) \wedge \\
& \quad \mathit{Diff}(nsw, v) \wedge \mathit{Diff}(nsw, sa) \wedge \\
& \quad \mathit{Diff}(v, sa) \Rightarrow \mathit{Colorable}()
\end{aligned}
$$

$$\mathit{Diff}(Red, Blue) \quad \mathit{Diff}(Red, Green)$$

$$\mathit{Diff}(Green, Red) \quad \mathit{Diff}(Green, Blue)$$

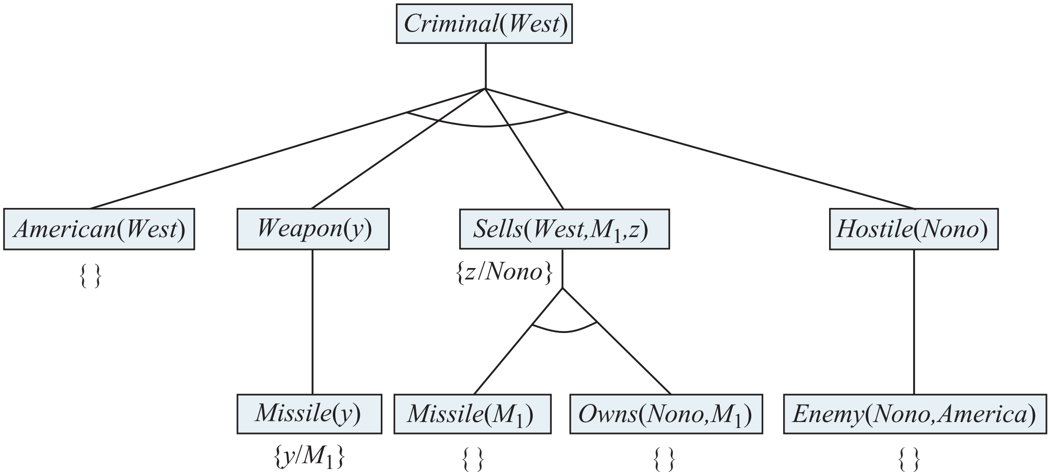
$$\mathit{Diff}(Blue, Red) \quad \mathit{Diff}(Blue, Green)$$

(b)

Backward Chaining

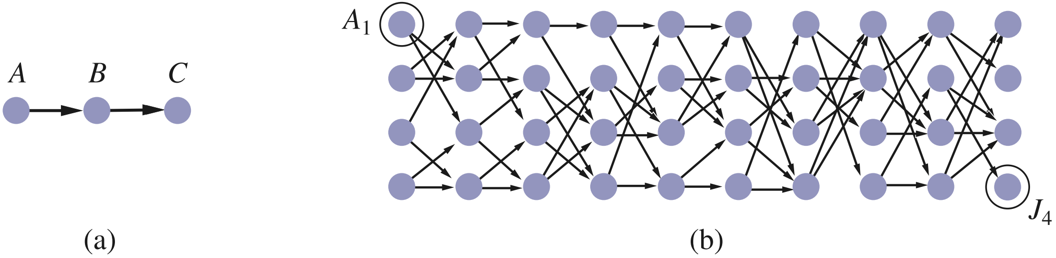
```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

function FOL-BC-OR(KB, goal, θ) returns a substitution
  for each rule in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs ⇒ rhs) ← STANDARDIZE-VARIABLES(rule)
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ′

function FOL-BC-AND(KB, goals, θ) returns a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else
    first,rest ← FIRST(goals), REST(goals)
    for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
      for each θ″ in FOL-BC-AND(KB, rest, θ′) do
        yield θ″
```

Backward Chaining

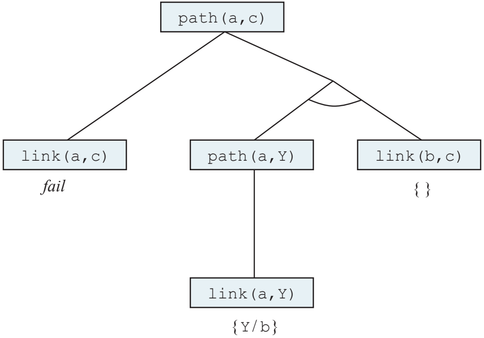


KENNESAW STATE UNIVERSITY

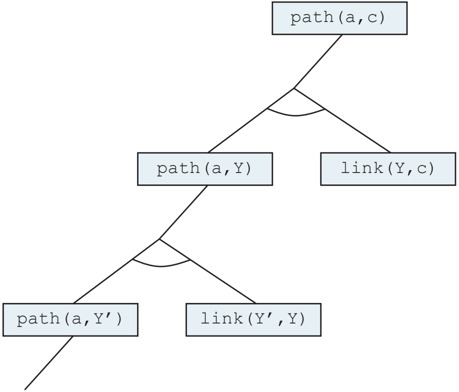
Logic Programming



Logic Programming

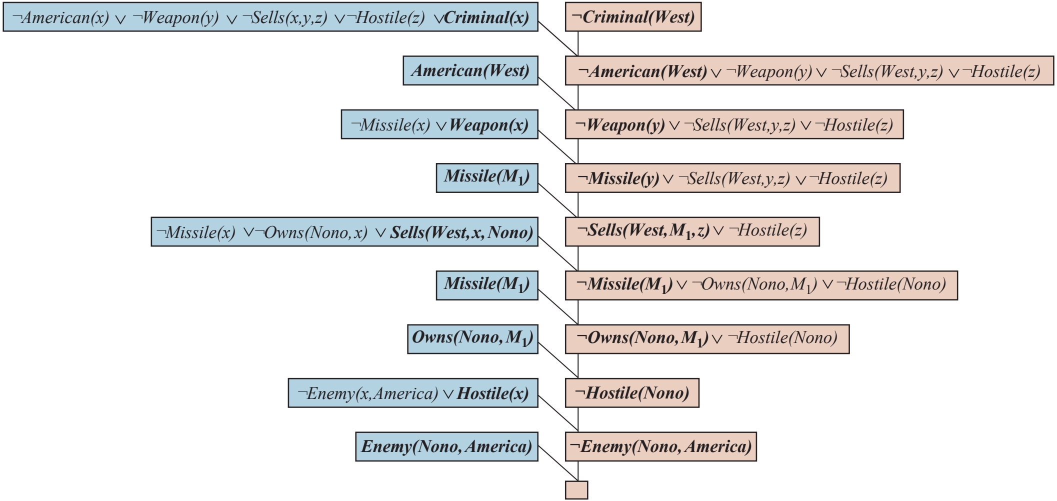


(a)

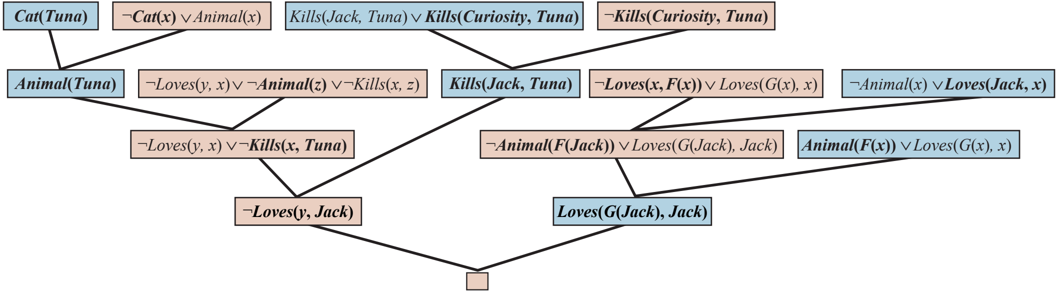


(b)

Resolution


¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x,y,z) ∨ ¬Hostile(z) ∨ Criminal(x)
¬Criminal(West)
American(West)
¬American(West) ∨ ¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
¬Missile(x) ∨ Weapon(x)
¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
Missile(M_1)
¬Missile(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
¬Missile(x) ∨ ¬Owns(Nono,x) ∨ Sells(West,x,Nono)
¬Sells(West,M_1,z) ∨ ¬Hostile(z)
Missile(M_1)
¬Missile(M_1) ∨ ¬Owns(Nono,M_1) ∨ ¬Hostile(Nono)
Owns(Nono,M_1)
¬Owns(Nono,M_1) ∨ ¬Hostile(Nono)
¬Enemy(x,America) ∨ Hostile(x)
¬Hostile(Nono)
Enemy(Nono,America)
¬Enemy(Nono,America)

Resolution


Cat(Tuna)
¬Cat(x) ∨ Animal(x)
Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
¬Kills(Curiosity, Tuna)
Animal(Tuna)
¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)
Kills(Jack, Tuna)
¬Loves(x, F(x)) ∨ Loves(G(x), x)
¬Animal(x) ∨ Loves(Jack, x)
¬Loves(y, x) ∨ ¬Kills(x, Tuna)
¬Animal(F(Jack)) ∨ Loves(G(Jack), Jack)
Animal(F(x)) ∨ Loves(G(x), x)
¬Loves(y, Jack)
Loves(G(Jack), Jack)

Completeness

Any set of sentences S is representable in clausal form

Assume S is unsatisfiable, and in clausal form

Herbrand's theorem

Some set S' of ground instances is unsatisfiable

Ground resolution theorem

Resolution can find a contradiction in S'

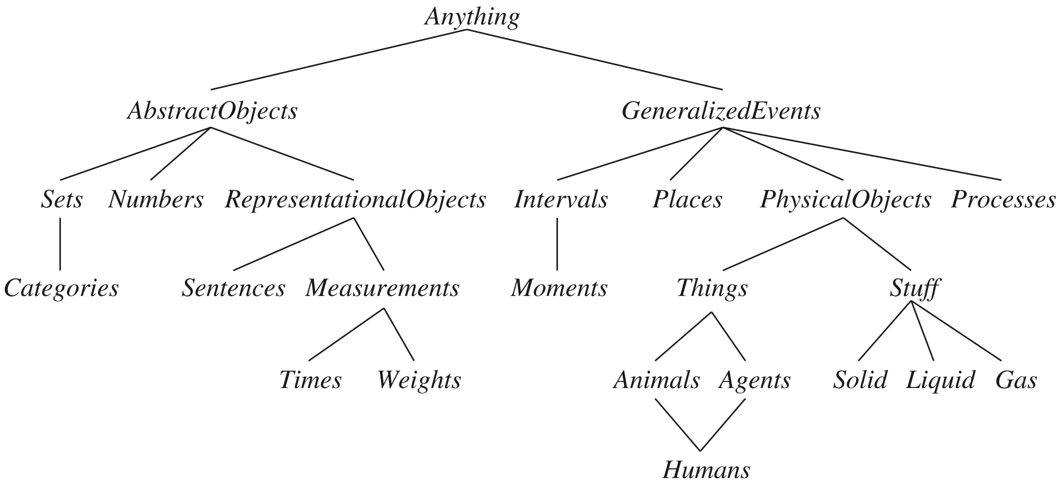
Lifting lemma

There is a resolution proof for the contradiction in S'

G{"o}del's Incompleteness Theorem

Foo

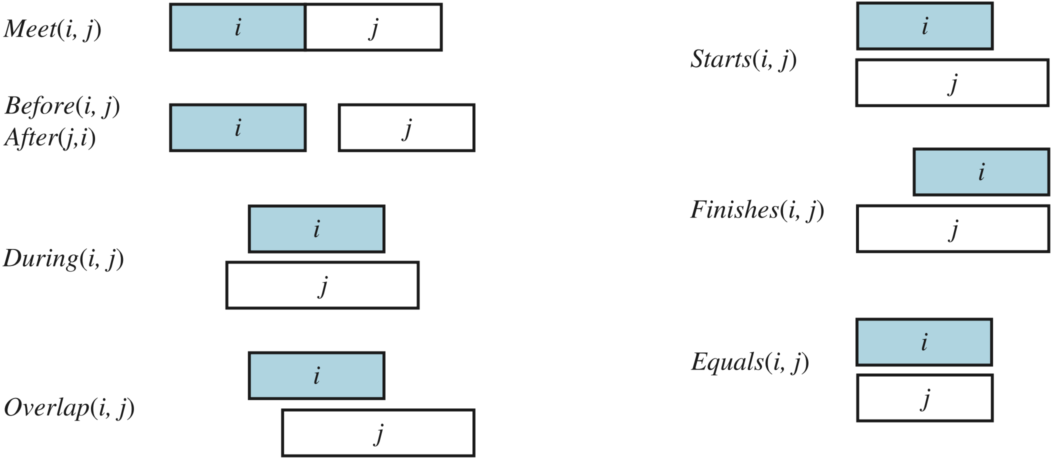
Ontological Engineering



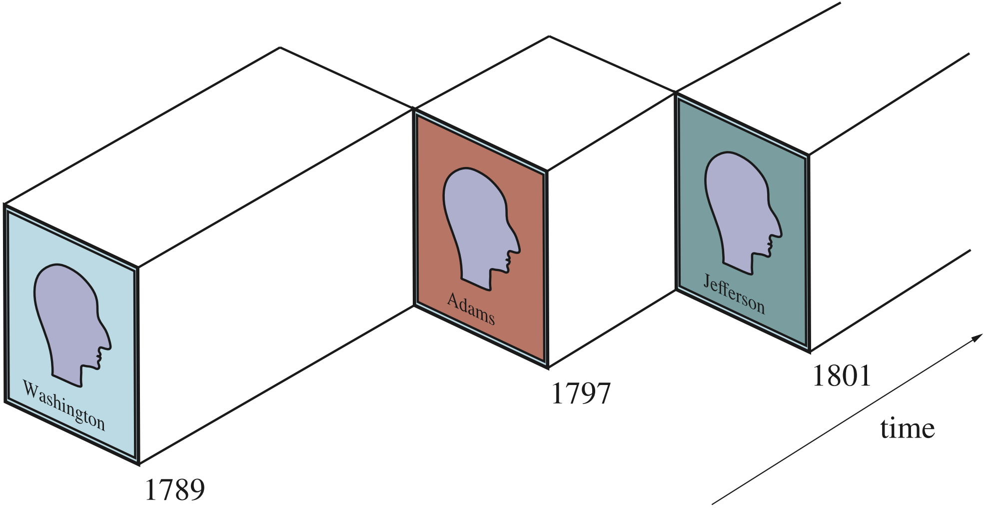
Categories and Objects

Foo

Events


Meet(i, j)
i
j
Before(i, j)
After(j,i)
i
j
During(i, j)
i
j
Overlap(i, j)
i
j
Starts(i, j)
i
j
Finishes(i, j)
i
j
Equals(i, j)
i
j

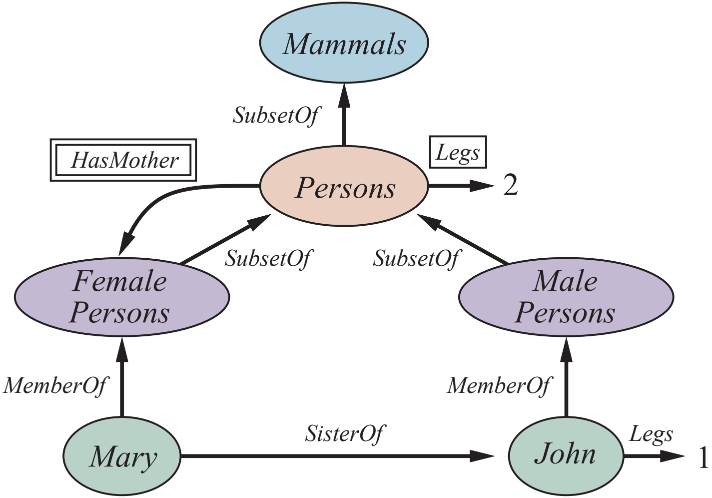
Fluents



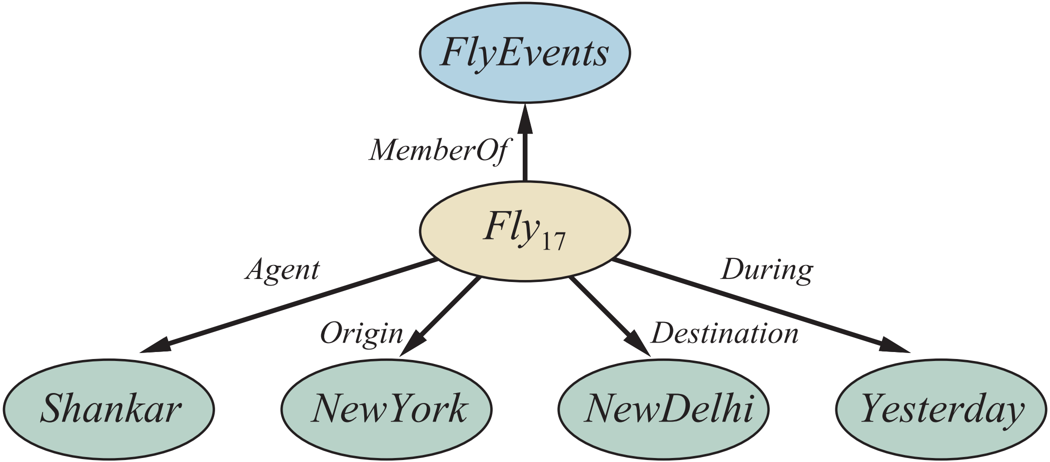
Mental Objects and Modal Logic

Foo

Reasoning Systems for Categories



Reasoning Systems for Categories



Description Logics

$$
\begin{aligned}
\mathit{Concept} &\rightarrow \textbf{Thing} \mid \mathit{ConceptName} \\
&\mid \quad \textbf{And}(\mathit{Concept}, \ldots) \\
&\mid \quad \textbf{All}(\mathit{RoleName}, \mathit{Concept}) \\
&\mid \quad \textbf{AtLeast}(\mathit{Integer}, \mathit{RoleName}) \\
&\mid \quad \textbf{AtMost}(\mathit{Integer}, \mathit{RoleName}) \\
&\mid \quad \textbf{Fills}(\mathit{RoleName}, \mathit{IndividualName}, \ldots) \\
&\mid \quad \textbf{SameAs}(\mathit{Path}, \mathit{Path}) \\
&\mid \quad \textbf{OneOf}(\mathit{IndividualName}, \ldots) \\
\mathit{Path} &\rightarrow [\mathit{RoleName}, \ldots] \\
\mathit{ConceptName} &\rightarrow \mathit{Adult} \mid \mathit{Female} \mid \mathit{Male} \mid \ldots \\
\mathit{RoleName} &\rightarrow \mathit{Spouse} \mid \mathit{Daughter} \mid \mathit{Son} \mid \ldots
\end{aligned}
$$

KENNESAW STATE UNIVERSITY

Reasoning with Default Information

Truth maintenance systems