Knowledge-Based Agents

- An agent that uses prior or acquired knowledge to achieve its goals
- Knowledge Base: Representations of the agent's environment
- ASK and TELL information
- Agent that has KB in it, KB has a knowledge base and inference engine, the inference engine proves things for your agent(Using situation calculus or STRIP operator) and comes up with a plan

Logics

- Logics are formal languages for representing information such as conclusions that can be drawn
- Syntax defines sentences in the language
- Semantics define meaning in sentences

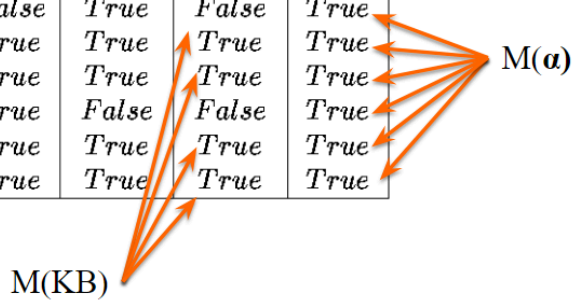
Entailment

- KB entails alpha if alpha is true in all worlds where KB is true (All of KB is in alpha)

E.g., the KB containing "the Giants won" and "the Reds won" entails "Either the Giants won or the Reds won"

KB |= α? Is M(KB) a subset of M(**α**)?

A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
$False$	$False$	$False$	$False$	$True$	$False$	$False$
$False$	$False$	$True$	$True$	$False$	$False$	$False$
$False$	$True$	$False$	$False$	$True$	$False$	$True$
$False$	$True$	$True$	$True$	$True$	$True$	$True$
$True$	$False$	$False$	$True$	$True$	$True$	$True$
$True$	$False$	$True$	$True$	$False$	$False$	$True$
$True$	$True$	$False$	$True$	$True$	$True$	$True$
$True$	$True$	$True$	$True$	$True$	$True$	$True$



Inference

- Sentence alpha can be derived from KB by procedure(s)
- Complete and Sound

Propositional Logic

- Proposition models are True/False

  Normal Form
  - Conjunctive Normal Form: Product of sums of simple variables or negated simple symbols (Ands)
  - Disjunctive Normal Form: Sum of products of simple variables or negated simple symbols (Ors)
  - Horn Form: Conjunction of horn clauses (<=1 positive literal)
    - The left side has lots of information, and the right side only has 1

- Only ands on the left side, right side only 1 object

- We convert sentences to Horn form as they are entered into the KB
- Using Existential Elimination and And Elimination

- e.g., ∃x Owns(Nono, x) ∧ Missile(x) becomes

  Owns(Nono, M)
  Missile(M)

- Can OR a bunch of statements (Derive expressions from functions)

- Tautologies: Always true

Validity and Satisfiability

- Valid: True in all models
- Satisfiable: True in some model
- Unsatisfiable: True in no models

Proof Checking

- Model Checking (Truth Tables), Search
- Inference Rules

◇ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

◇ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◇ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◇ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◇ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◇ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \quad \neg\beta}{\alpha}$$

◇ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Logical agents apply inference to a knowledge base
to derive new information and make decisions

Basic concepts of logic:
- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Propositional logic suffices for some of these tasks

Truth table method is sound and complete for propositional logic

- Knowledge Base (KB): contains a set of sentences expressed using a Knowledge Representation Language (KRL)
  - TELL: operator to add a sentence to the KB
  - ASK: to query the KB
- Logics are KRLs where conclusions can be drawn
  - Syntax
  - Semantics
- Entailment: KB |= a iff "a is true in all worlds where KB is true"
- Inference: KB $|-_i$ a = sentence a can be derived from KB using a procedure *i*
  - Sound: whenever KB $|-_i$ a, then KB |= a is true
  - Complete: whenever KB |= a, then KB $|-_i$ a

First Order Logic

- Prepositional logic is limited and it is difficult to represent simple worlds
- First Order Logic has Objects, Relations/Predicates (Return T/F), Functions (Returns Object), Properties

- "One plus two equals three"

  Objects: one, two, three, one plus two
  Relations: equals
  Properties: --
  Functions: plus

  ("one plus two" is the name of the object, obtained by applying function plus to one and two; three is another name for this object)

You can fool all of the people some of the time.

```
∀x (person(x) => ∃t (time(t) ^ can-fool(x,t)))
```

All purple mushrooms are poisonous.

```
∀x (mushroom(x) ^ purple(x) => poisonous(x))
```

Every gardener likes the sun.

```
∀ x gardener(x) => likes(x,Sun)
```

You can fool some of the people all of the time.

```
∃ x ∀ t (person(x) ^ time(t)) => can-fool(x,t)
```

Situation Calculus

- Hold the current state or situation as well

First-order logic:

- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:

- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Basic manipulation rules

¬(¬A) = A Double negation

¬(A ^ B) = (¬A) V (¬B) Negated “and”

¬(A V B) = (¬A) ^ (¬B) Negated “or”

A ^ (B V C) = (A ^ B) V (A ^ C) Distributivity of ^ on V

A V (B ^ C) = (A V B) ^ (A V C) Distributivity of V on ^

A => B = (¬A) V B by definition

¬(A => B) = A ^ (¬B) using negated or

A ⇔ B = (A => B) ^ (B => A) by definition

¬(A ⇔ B) = (A ^ (¬B))V(B ^ (¬A)) using negated and & or

…

Inference in FOL

- Ground Term: Term that does not contain a variable

First Order Logic Proof

E.g., Modus Ponens (MP)

$$\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \qquad \frac{At(Joe, UCB) \quad At(Joe, UCB) \Rightarrow OK(Joe)}{OK(Joe)}$$

E.g., And-Introduction (AI)

$$\frac{\alpha \quad \beta}{\alpha \wedge \beta} \qquad \frac{OK(Joe) \quad CSMajor(Joe)}{OK(Joe) \wedge CSMajor(Joe)}$$

E.g., Universal Elimination (UE)

$$\frac{\forall x \ \ \alpha}{\alpha\{x/\tau\}} \qquad \frac{\forall x \ \ At(x, UCB) \Rightarrow OK(x)}{At(Pat, UCB) \Rightarrow OK(Pat)}$$

τ must be a ground term (i.e., no variables)

- **Universal Elimination (UE):**
  for any sentence α, variable x and ground term τ,
  $\frac{\forall x \ \alpha}{\alpha\{x/\tau\}}$ e.g., from ∀x Likes(x, Candy) and {x/Joe} we can infer Likes(Joe, Candy)
- **Existential Elimination (EE):**
  for any sentence α, variable x and a new constant symbol k not in KB,
  $\frac{\exists x \ \alpha}{\alpha\{x/k\}}$ e.g., from ∃x Kill(x, Victim) we can infer Kill(Murderer, Victim), if Murderer is a new symbol
- **Existential Introduction (EI):**
  for any sentence α, variable x not in α and ground term g in α,
  $\frac{\alpha}{\exists x \ \alpha\{g/x\}}$ e.g., from Likes(Joe, Candy) we can infer ∃x Likes(x, Candy)

Unification

p	q	
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{y/John, x/OJ\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$

VARIABLE term

1 – unify(P(a,X), P(a,b)) σ = {X/b}

2 – unify(P(a,X), P(Y,b)) σ = {Y/a, X/b}

3 – unify(P(a,X), P(Y,f(a)) σ = {Y/a, X/f(a)}

4 – unify(P(a,X), P(X,b)) σ = failure

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\sigma} \quad \text{where } p_i'\sigma = p_i\sigma \text{ for all } i$$

E.g. p_1' = Faster(Bob,Pat)
p_2' = Faster(Pat,Steve)
$p_1 \wedge p_2 \Rightarrow q = Faster(x,y) \wedge Faster(y,z) \Rightarrow Faster(x,z)$
$\sigma = \{x/Bob, y/Pat, z/Steve\}$
$q\sigma = Faster(Bob, Steve)$

GMP used with KB of definite clauses (*exactly* one positive literal):
either a single atomic sentence or
(conjunction of atomic sentences) ⇒ (atomic sentence)
All variables assumed universally quantified

- Why is GMP an efficient inference rule?

  - It takes bigger steps, combining several small inferences into one

  - It takes sensible steps: uses eliminations that are guaranteed to help (rather than random UEs)

  - It uses a precompilation step which converts the KB to canonical form (Horn sentences)

Forward Chaining

- Apply rules as you get more data (Apply rules whenever you have the chance to get more facts)

Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x) (1)
Use(friends, x) ∧ Runs(x, N64 games) ⇒ **Provide(Reality Man, friends, x)** (2)
Software(x) ∧ Runs(x, N64 games) ⇒ **Emulator(x)** (3)

Programmer(Reality Man) (4)
People(friends) (5)
Software(U64) (6)
Use(friends, U64) (7)
Runs(U64, N64 games) (8)
Provide(Reality Man, friends, U64) (9)
Emulator(U64) (10)
Criminal(Reality Man) (11)

Which results in the final conclusion: Criminal(Reality Man)

Backward Chaining

- Look at all rules in KB and see if the rule you are looking for could be created

- Question: Has Reality Man done anything criminal?
- We will use the same knowledge as in our forward-chaining version of this example:

  Programmer(x) ∧ Emulator(y) ∧ People(z) ∧ Provide(x,z,y)⇒ Criminal(x)

  Use(friends, x) ∧ Runs(x, N64 games) ⇒ Provide(Reality Man, friends, x)

  Software(x) ∧ Runs(x, N64 games) ⇒ Emulator(x)

  Programmer(Reality Man)

  People(friends)

  Software(U64)

  Use(friends, U64)

  Runs(U64, N64 games)

GMP is incomplete

Godels's Theorem

- Any sentence entailed by a list of sentences can be proven from that set

Resolution

Entailment in first-order logic is only semidecidable:
can find a proof of α if $KB \models \alpha$
cannot always prove that $KB \not\models \alpha$
Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

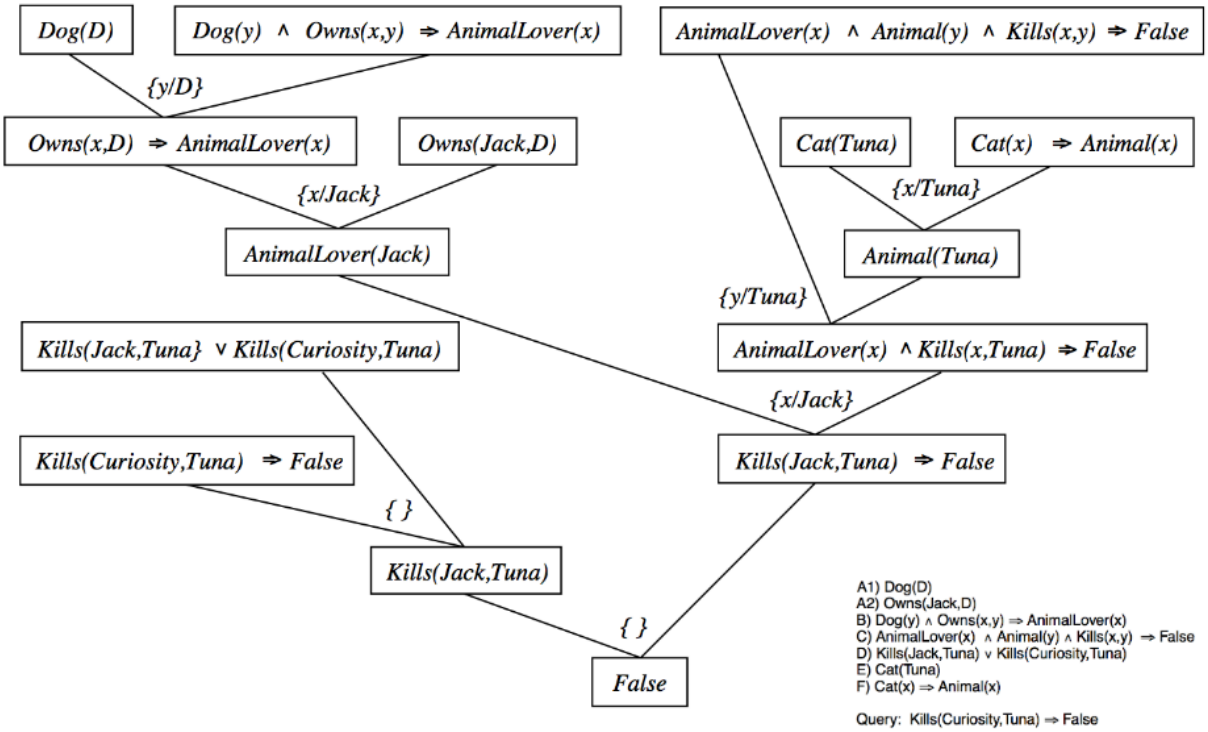
Resolution is a refutation procedure:
to prove $KB \models \alpha$, show that $KB \wedge \neg\alpha$ is unsatisfiable

Resolution uses KB, $\neg\alpha$ in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:

Inference continues until an empty clause is derived (contradiction)

Anything can be written in CNF form which is great for resolution



Logical Reasoning Systems

- Provers: Use resolution
- Languages: Uses backward chaining
- Production System: Based on implications and forward changing
- Frame systems and semantic networks: Graph structure
- Description Logic systems: Reason with object classes and relations among them

Basic Tasks

- TELL, ASK, restricted ASK, remove sentence from KB, FETCH, STORE
- Usually use hash table to store, index based on operators and arguments

Logic Programming

- Prolog programming language: List of logic statements
- Closed World Assumption (Anything that I don't know is False)

```
/* program P                        clause #      */

p(a).                               /* #1 */
p(X) :- q(X), r(X).                 /* #2 */
p(X) :- u(X).                       /* #3 */

q(X) :- s(X).                       /* #4 */


r(a).                               /* #5 */
r(b).                               /* #6 */


s(a).                               /* #7 */
s(b).                               /* #8 */
s(c).                               /* #9 */

u(d).                               /* #10 */
```

q(X) ^ r(X) -> p(X)

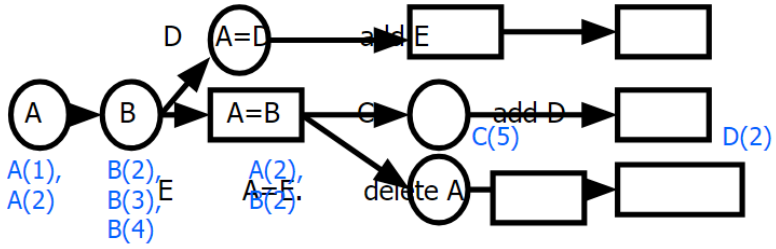
Logical Variable - like a substitute

<u>Theorem Provers</u>

- OTTER (Organized Techniques for Theorem Proving and Effective Research)

<u>Production Systems (Forward Chaining)</u>

- Uses rete algorithm



Circular nodes: fetches to WM; rectangular nodes: unifications

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

$A(x) \wedge B(y) \wedge D(x) \Rightarrow$ add $E(x)$

$A(x) \wedge B(x) \wedge E(x) \Rightarrow$ delete $A(x)$

{A(1), A(2), B(2), B(3), B(4), C(5)}

<u>Frame Systems and Semantic Networks</u>

- Different notation

Link Type	Semantics
$A \xrightarrow{\text{Subset}} B$	$A \subset B$
$A \xrightarrow{\text{Member}} B$	$A \in B$
$A \xrightarrow{R} B$	$R(A,B)$
$A \xrightarrow{[R]} B$	$\forall x\ [\ x \in A \Rightarrow R(x,y)\ ]$
$A \xrightarrow{[R]} B$	$\forall x\ \exists y\ [\ x \in A \Rightarrow y \in B \wedge R(x,y)\ ]$

Description Logic Systems

- Makes it easier to describe categories

- Concept expressions, e.g.,
  - Doctor ⊔ Lawyer
  - Rich ⊓ Happy
  - Cat ⊓ ∃sits-on.Mat
- Equivalent to FOL formulae with one free variable
  - $\text{Doctor}(x) \vee \text{Lawyer}(x)$
  - $\text{Rich}(x) \wedge \text{Happy}(x)$
  - $\exists y.(\text{Cat}(x) \wedge \text{sits-on}(x, y))$
- Special concepts
  - ⊤ (aka top, Thing, most general concept)
  - ⊥ (aka bottom, Nothing, inconsistent concept)

  used as abbreviations for
  - (A ⊔ ¬ A) for any concept A
  - (A ⊓ ¬ A) for any concept A

OWL exploits results of 20+ years of DL research

- Well defined (model theoretic) **semantics**
- **Formal properties** well understood (complexity, decidability)

I can't find an efficient algorithm, but neither can all these famous people.

[Garey & Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, 1979.]

- Known **reasoning algorithms**
- **Scalability** demonstrated by **implemented systems**

Planning

1. Open up action and goal representation to allow selection
2. Divide and conquer by subdividing
3. Relax requirement for sequential construction of solutions

- STRIPS language

states: conjunctions of function-free ground literals (I.e., predicates applied to constant symbols, possibly negated); e.g.,

At(Home) ∧ ¬Have(Milk) ∧ ¬Have(Bananas) ∧ ¬Have(Drill) …

goals: also conjunctions of literals; e.g.,

At(Home) ∧ Have(Milk) ∧ Have(Bananas) ∧ Have(Drill)

but can also contain variables (implicitly universally quant.); e.g.,

At(x) ∧ Sells(x, Milk)

- Planner: ask for sequence of actions that makes goal true if executed

- Theorem prover: ask whether query sentence is true given KB

STRIP operations

- Action
- Precondition
- Effect

Types of Planners

- Situation space planner: search through possible situations/states

- Progression (forward) Planner:
  start with the initial state, apply operators until the goal is reached
  Problem: high branching factor!

- Regression (backward) Planner:
  start from the goal state and apply operators until the start state reached
  Why desirable? usually many more operators are applicable to the initial state than to goal state.
  Difficulty: when want to achieve a conjunction of goals

Initial STRIPS algorithm: situation-space regression planner

Partial order planner: some steps are ordered, some are not

Total order planner: all steps ordered (thus, plan is a simple list of steps)

Linearization: process of deriving a totally ordered plan from a partially ordered plan.

- The plan is complete if every precondition is achieved

We formally define a plan as a data structure consisting of:

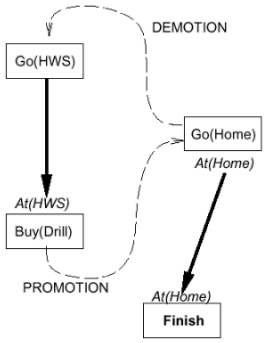
- Set of plan steps (each is an operator for the problem)
- Set of step ordering constraints

  e.g., A B $\prec$ means "A before B"
- Set of variable binding constraints

  e.g., v = x where v variable and x constant or other variable
- Set of causal links

  e.g., A B c means "A achieves c for B"

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(HWS)$:



Demotion: put before $Go(HWS)$

Promotion: put after $Buy(Drill)$

Keywords:

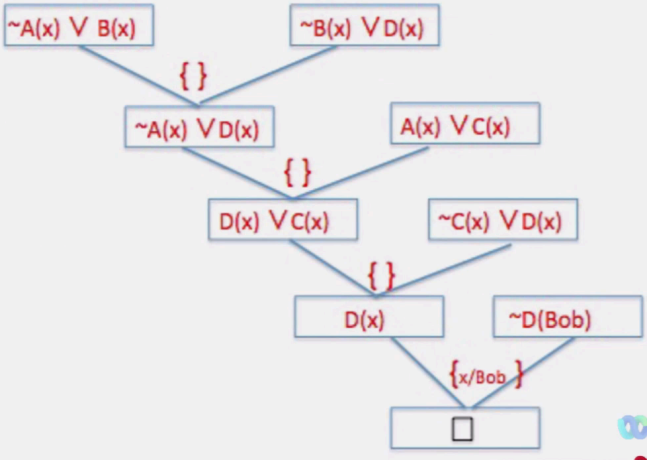
I	**Syntax**
D	**Semantics**
C	**Model**
E	**Entailment**
B	**Inference**
H	**Soundness**
J	**Completeness**
A	**Equivalence**
F	**Validity**
G	**Satisfiability**

Definitions:

A. sentences are true in the same models
B. determine whether sentence entailed by KB
C. a possible world that defines truth values for all sentences
D. truth of sentences with respect to models
E. necessary truth of one sentence given another
F. sentence is true in all models
G. sentence is true in some model
H. produce only entailed sentences
I. formal structure of sentences
J. can produce all entailed sentences

Use resolution and a proof by contradiction to prove D(Bob) from the following knowledge base:

~A(x) ∨ B(x)
A(x) ∨ C(x)
~B(x) ∨ D(x)
~C(x) ∨ D(x)
~D(Bob)



Given: R1: append([],Y,Y).

R2: append([X|L],Y, [X|Z]) :- append(L,Y, Z).

Data: L=[A], Y=[B,C], X=[[D]]:

Prove: Z = ?, [X|Z] = ?

[X|Z]=[[[D]], A,B,C]

Unify with R2: append([[[D]]|[A]], [B,C], [[[D]]|Z]) :- append([A], [B, C], Z).

Subquery: append([A], [B,C], Z)

Z=[A,B,C]

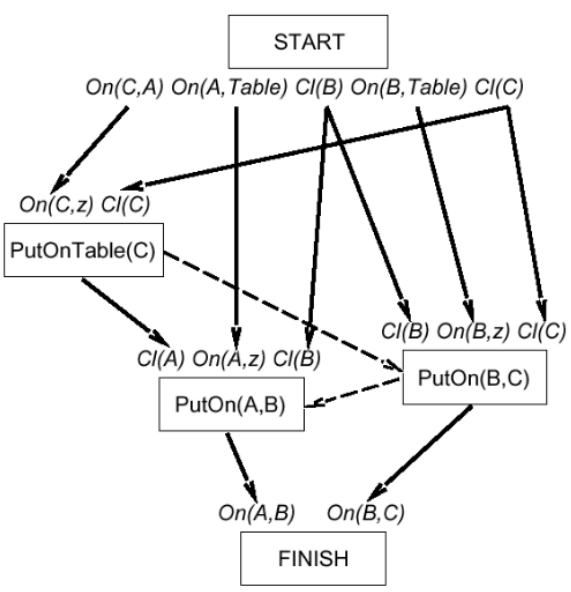
Unify with R2: append([A|[]], [B,C], [A|Z]) :- append([], [B,C], Z)

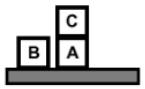
Subquery: append([], [B,C], Z)

Z=[B,C]

Unify with R1: append([], [B,C], [B,C])

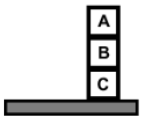
Z=[A, B, C] [X|Z]=[[[D]], A, B, C]





PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)

PutOn(B,C)
clobbers Cl(C)
=> order after
PutOnTable(C)



CNF Conversion

1. **Eliminate implications**
2. **Move ¬ inward**
3. **Standardize variables**
4. Skolemization **(removing existential quantifiers**
5. **Drop universal quantifiers**
6. **Distribute ∨ over ∧**