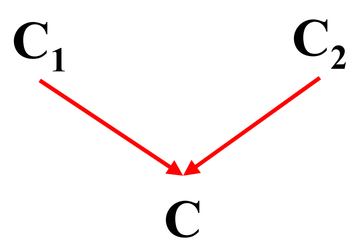
Logical Inference and Reasoning Agents

Foundations of Artificial Intelligence

Resolution Rule of Inference

- **Resolution provides a complete rule of inference for first order predicate calculus**
  - if used in conjunction with a refutation proof procedure (proof by contradiction)
  - requires that formulas be written in clausal form
- **Refutation procedure**
  - to prove that $KB \models \alpha$, show that $KB \wedge \neg\alpha$ is unsatisfiable
  - i.e., assume the contrary of α, and arrive at a contradiction
  - KB and $\neg\alpha$, must be in CNF (conjunction of clauses)
  - each step in the refutation procedure involves applying resolution to two clauses, in order to get a new clause



  - inference continues until the empty clause ⊠ is derived (a contradiction)

Resolution Rule of Inference

- **Basic Propositional Version:**

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

- **Full First-Order Version:**

$$\frac{(p_1 \vee \ldots \vee p_j \vee \ldots \vee p_m), (q_1 \vee \ldots \vee q_k \vee \ldots \vee q_n)}{(p_1 \vee \ldots \vee p_{j-1} \vee p_{j+1} \vee \ldots \vee p_m \vee q_1 \vee \ldots \vee q_{k-1} \vee q_{k+1} \vee \ldots \vee q_n)\sigma}$$

provided that p_j and $\neg q_k$ are *unifiable* via a *substitution* σ

- **Example:**

$$\neg rich(x) \vee unhappy(x) \qquad rich(bob)$$

$$unhappy(bob)$$

with substitution $\sigma = \{x/bob\}$

Conjunctive Normal Form - Revisited

- **Literal = possibly negated atomic sentence**
  - e.g., $\neg \boldsymbol{rich}(\boldsymbol{x})$, **or** $\boldsymbol{unhappy}(\boldsymbol{bob})$, **etc.**
- **Clause = disjunction of literals**
  - e.g., $\neg \boldsymbol{rich}(\boldsymbol{x}) \vee \boldsymbol{unhappy}(\boldsymbol{x})$
- **The *KB* is a conjunction of clauses**
- **Any first-order logic *KB* can be converted into CNF:**
  - 1. Replace $\boldsymbol{P} \Rightarrow \boldsymbol{Q}$ with $\neg \boldsymbol{P} \vee \boldsymbol{Q}$
  - 2. Move inward the negation symbol, e.g., $\neg\forall \boldsymbol{x}\ \boldsymbol{P}$ becomes $\exists \boldsymbol{x}\ \neg \boldsymbol{P}$
  - 3. Standardize variables apart, e.g., $\forall \boldsymbol{x}\ \boldsymbol{P} \vee \exists \boldsymbol{x}\ \boldsymbol{Q}$ becomes $\forall \boldsymbol{x}\ \boldsymbol{P} \vee \exists \boldsymbol{y}\ \boldsymbol{Q}$
  - 4. Move quantifiers left in order, e.g., $\forall \boldsymbol{x}\ \boldsymbol{P} \vee \exists \boldsymbol{y}\ \boldsymbol{Q}$ becomes $\forall \boldsymbol{x} \exists \boldsymbol{y}\ (\boldsymbol{P} \vee \boldsymbol{Q})$
  - 5. Eliminate $\exists$ by Skolemization (see later slide)
  - 6. Drop universal quantifiers (we'll assume they are implicit)
  - 7. Distribute $\wedge$ over $\vee$, e.g., $(\boldsymbol{P} \wedge \boldsymbol{Q}) \vee \boldsymbol{R}$ becomes $(\boldsymbol{P} \vee \boldsymbol{Q}) \wedge (\mathbf{P} \vee \boldsymbol{R})$
  - 8. Split conjunctions (into a set of clauses) and rename variables

Conversion to CNF - Example 1

- **Original sentence** $(A \wedge B \Rightarrow C) \vee (D \wedge \neg G)$
- **Eliminate $\Rightarrow$:** $(\neg(A \wedge B) \vee C) \vee (D \wedge \neg G)$
- **Move in negation:** $\neg A \vee \neg B \vee C \vee (D \wedge \neg G)$
- **Distribute $\wedge$ over $\vee$:** $(\neg A \vee \neg B \vee C \vee D) \wedge (\neg A \vee \neg B \vee C \vee \neg G)$
- **Split conjunction**

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

or equivalently

$$(A \wedge B \Rightarrow C \vee D)$$
$$(A \wedge B \wedge G \Rightarrow C)$$

This is a set of two clauses

Skolemization

- **The rules for Skolemization is essentially the same as those we described for quantifier inference rules**
  - if ∃ does not occur within the scope of a ∀, then drop ∃, and replace all occurrence of the existentially quantified variable with a new constant symbol (called the Skolem constant)
  - e.g., $\exists x\, P(x)$ becomes $P(\hat{a})$, where $\hat{a}$ is a new constant symbol
  - if ∃ is within the scope of any ∀, then drop ∃, and replace the associated variable with a Skolem function (a new function symbol), whose arguments are the universally quantified variables
  - e.g., $\forall x \forall y \exists z\, P(x, y, z)$ becomes $\forall x \forall y\, P(x, y, sk(x, y))$
  - e.g., $\forall x\, person(x) \Rightarrow \exists y\, heart(y) \wedge has(x,y)$

    becomes $\forall x\, person(x) \Rightarrow heart(sk(x)) \wedge has(x, sk(x))$

Conversion to CNF - Example 2

Convert: $\forall x\,[(\forall y\; p(x,y)) \Rightarrow \neg(\forall y\,(q(x,y) \Rightarrow r(x,y)))]$

(1) $\forall x\,[\neg(\forall y\; p(x,y)) \vee \neg(\forall y\,(\neg q(x,y) \vee r(x,y)))]$

(2) $\forall x\,[(\exists y\; \neg p(x,y)) \vee (\exists y\,(q(x,y) \wedge \neg r(x,y)))]$

(3) $\forall x\,[(\exists y\; \neg p(x,y)) \vee (\exists z\,(q(x,z) \wedge \neg r(x,z)))]$

(4) $\forall x \exists y \exists z\,[\neg p(x,y) \vee (\, q(x,z) \wedge \neg r(x,z))]$

(5) $\forall x\,[\neg p(x,sk_1(x)) \vee (\, q(x,sk_2(x)) \wedge \neg r(x,sk_2(x)))]$

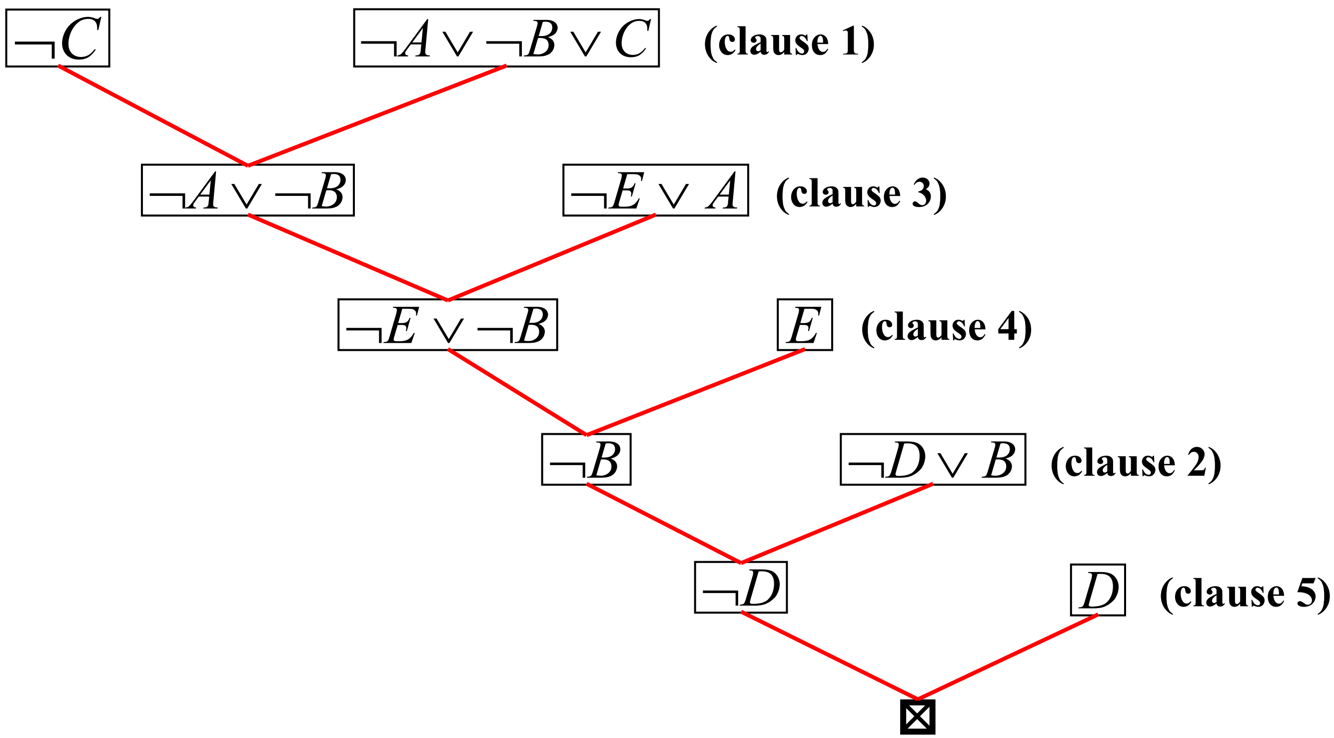
(6) $\neg p(x,sk_1(x)) \vee (\, q(x,sk_2(x)) \wedge \neg r(x,sk_2(x)))$

(7) $[\neg p(x,sk_1(x)) \vee q(x,sk_2(x))] \wedge [\neg p(x,sk_1(x)) \vee \neg r(x,sk_2(x))]$

(8) $\{\, \neg p(x,sk_1(x)) \vee q(x,sk_2(x)), \quad \neg p(w,sk_1(w)) \vee \neg r(w,sk_2(w)) \,\}$

Refutation Procedure - Example 1

Given $\boldsymbol{KB} = \left\{ \begin{array}{ll} 1. & \neg A \vee \neg B \vee C \\ 2. & \neg D \vee B \\ 3. & \neg E \vee A \\ 4. & E \\ 5. & D \end{array} \right\}$ prove $\boldsymbol{KB} \models \boldsymbol{C}$



Refutation Procedure - Example 2

$$
KB = \left\{ \begin{array}{ll}
1. & father(john, mary) \\
2. & mother(sue, john) \\
3. & father(bob, john) \\
4. & \forall x \forall y [(father(x,y) \vee mother(x,y)) \Rightarrow parent(x,y)] \\
5. & \forall x \forall y [\exists z (parent(x,z) \wedge parent(z,y)) \Rightarrow grand(x,y)]
\end{array} \right\}
$$

Converting 4 to CNF:

$$4. \quad (\neg father(x,y) \vee parent(x,y)) \wedge (\neg mother(x,y) \vee parent(x,y))$$

Converting 5 to CNF:

$$5. \quad \forall x \forall y [\neg \exists z (parent(x,z) \wedge parent(z,y)) \vee grand(x,y)]$$

$$\equiv \forall x \forall y \forall z [\neg (parent(x,z) \wedge parent(z,y)) \vee grand(x,y)]$$

$$\equiv \neg parent(x,z) \vee \neg parent(z,y) \vee grand(x,y)$$

Refutation Procedure - Example 2 (cont.)

$$KB = \left\{ \begin{array}{ll} 1. & father(john, mary) \\ 2. & mother(sue, john) \\ 3. & father(bob, john) \\ 4. & \neg father(x,y) \vee parent(x,y) \\ 5. & \neg mother(x,y) \vee parent(x,y) \\ 6. & \neg parent(x,z) \vee \neg parent(z,y) \vee grand(x,y) \end{array} \right\}$$

Here is the final KB in clausal form:

A digression: what if we wanted to add a clause saying that there is someone who is neither the father nor the mother of *john*:

$$\exists x\, [\neg father(x, john) \wedge \neg mother(x, john)]$$

In clausal form:

$$\{ \neg father(\hat{a}, john), \neg mother(\hat{a}, john) \}$$

Next we want to prove each of the following using resolution refutation:

$grand(sue, mary)$ (sue is a grandparent of mary)

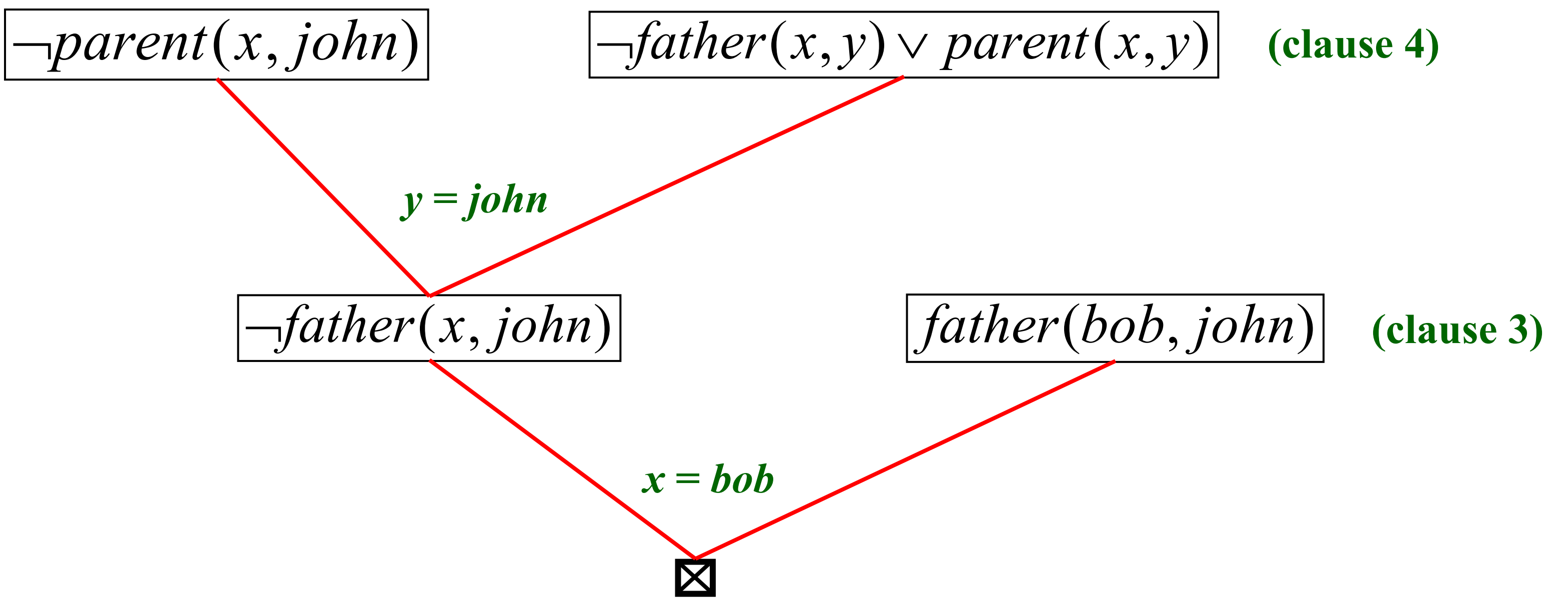
$\exists x\; parent(x, john)$ (there is someone who is john's parent)

Refutation Procedure - Example 2 (cont.)

To prove, we must first negate the goal and transform into clausal form:

$$\neg\exists x\ parent(x, john) \rightarrow \forall x\ \neg parent(x, john) \rightarrow \neg parent(x, john)$$

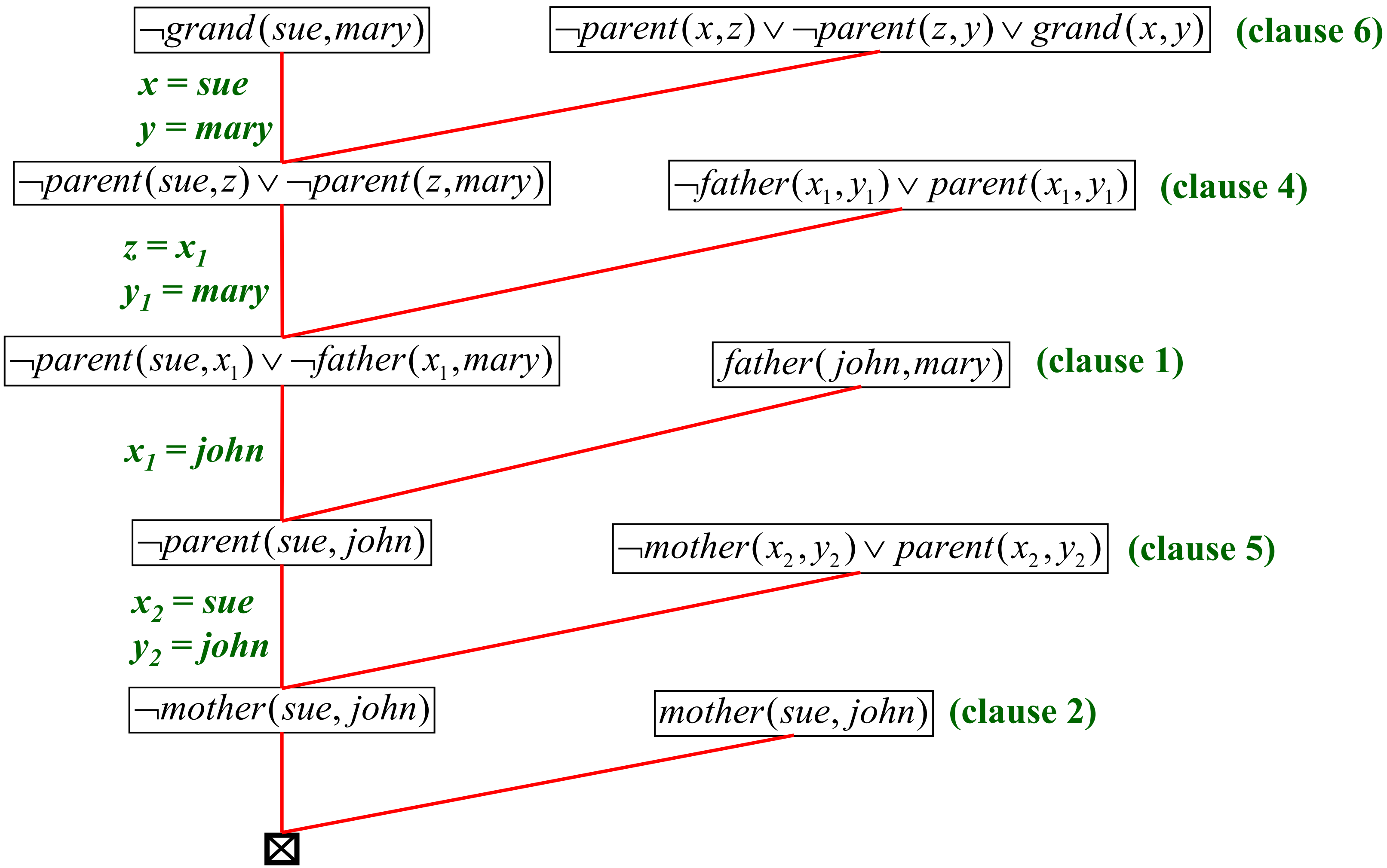
The refutation (proof by contradiction):



Note that the proof is *constructive*: we end up with an *answer* $x = bob$

Refutation Procedure - Example 2 (cont.)

Now, let’s prove that *sue* is the grandparent of *mary*:



Substitutions and Unification

- **A *substitution* is a set of *bindings* of the form $v = t$, where v is a variable and t is a term**
  - If P is an expression and σ is a substitution, then application of σ to P, denoted by $(P)\sigma$, is the result of *simultaneously* replacing each variable x in P with a term t, where $x = t$ is in σ
  - E.g., $P = likes(\texttt{sue}, z)$, and $\sigma = \{w = \texttt{john}, z = mother\_of(\texttt{john})\}$ then $(P)\sigma = likes(\texttt{sue}, mother\_of(\texttt{john}))$
  - E.g., $P = likes(father\_of(w), z)$, and $\sigma = \{w = \texttt{john}, z = mother\_of(x)\}$ then $(P)\sigma = likes(father\_of(john), mother\_of(x))$
  - E.g., $P = likes(father\_of(z), z)$, and $\sigma = \{z = mother\_of(\texttt{john})\}$ then $(P)\sigma = likes(father\_of(mother\_of(\texttt{john})), mother\_of(\texttt{john}))$
  - E.g., $P = likes(w, z)$, and $\sigma = \{w = \texttt{john}, z = mother\_of(w)\}$ then $(P)\sigma = likes(\texttt{john}, mother\_of(\texttt{john}))$

Substitutions and Unification

- **Let P and Q be two expressions, and σ a substitution. Then σ is a *unifier* of P and Q, if $(P)\sigma = (Q)\sigma$**
  - In the above definition, "=" means syntactic equality only
  - E.g., $P = likes(\texttt{john}, z)$, and $Q = likes(w, mother\_of(\texttt{john}))$
    then $\sigma = \{w = \texttt{john}, z = mother\_of(\texttt{john})\}$ is a unifier of P and Q
  - E.g., $E_1 = p(x, f(y))$, and $E_2 = p(g(z), w)$
    then $\sigma_1 = \{ x = g(a),\ y = b,\ z = a,\ w = f(b) \}$
    $\sigma_2 = \{ x = g(a),\ z = a,\ w = f(y) \}$
    $\sigma_3 = \{ x = g(z), w = f(y) \}$
    are all unifiers for the two expressions. What's the difference?
  - In the above example, σ_2 is more general than σ_1, since by applying some other substitution (in this case $\{y = b\}$) to elements of σ_2, we can obtain σ_1. We say that σ_1 is an *instance* of σ_2. Note that σ_3 is in fact the *most general unifier* (*mgu*) of E_1 and E_2: all instances of σ_3 are unifiers, and any substitution that is more general than σ_3 is not a unifier of E_1 and E_2 (e.g., $\sigma_4 = \{ x = v, w = f(y) \}$ is more general than σ_3, but is not a unifier.

Substitutions and Unification

- **Expressions may not be unifiable**
  - E.g., $E_1 = p(x, y)$, and $E_2 = q(x, y)$
    $E_1 = p(a, y)$, and $E_2 = p(f(x), y)$
    $E_1 = p(x, f(y))$, and $E_2 = p(g(z), g(w))$
    $E_1 = p(x, f(x))$, and $E_2 = p(y, y)$ (why are these not unifiable?)
  - How about $p(x)$ and $p(f(x))$?
    - the "occur check" problem: when unifying two expressions, need to check to make sure that a variable of one expression, does not occur in the other expression.
- **Another Example (find the mgu of E_1 and E_2)**

$$E_1 = p(\, f(x, g(x, y), h(z, y)\,) \qquad E_2 = p(\, z, h(f(u, v), f(a, b)\,)$$

  - how about $\sigma_1 = \{\, z = f(x, g(x,y)),\ z = f(u, v),\ y = f(a, b) \,\}$
    not good: don't know which binding for z to apply
  - how about $\sigma_2 = \{\, z = f(x, g(x,y)),\ u = x,\ v = g(x, y),\ y = f(a, b) \,\}$
    not good: is not a unifier
  - $\text{mgu}(E_1, E_2) = \{\, z = f(x, g(x, f(a,b))),\ u = x,\ v = g(x, f(a,b)),\ y = f(a, b) \,\}$

Forward and Backward Chaining

- **Generalized Modus Ponens**

$$\frac{p_1, p_2, \ldots, p_n, \quad q_1 \wedge q_2 \wedge \ldots \wedge q_n \Rightarrow q}{q\theta}$$

where θ is a substitution that unifies p_i and q_i for all i, i.e., $(p_i)\theta = (q_i)\theta$.

- GMP is complete for Horn knowledge bases
- Recall: a Horn knowledge base is one in which all sentences are of the form
  - $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ **OR**
  - $p_1 \wedge p_2 \wedge \ldots \wedge p_n$
- In other words, all sentence are in the form of an implication rule with zero or one predicate on the right-hand-side (sentences with zero predicates on the rhs are sometimes referred to as "facts").
- For such knowledge bases, we can apply GMP in a forward or a backward direction.

Forward and Backward Chaining

- **Forward Chaining**
  - Start with KB, infer new consequences using inference rule(s), add new consequences to KB, continue this process (possibly until a goal is reached)
  - In a knowledge-based agent this amounts to repeated application of the TELL operation
  - May generate many irrelevant conclusions, so not usually suitable for solving for a specific goal
  - Useful for building a knowledge base incrementally as new facts come in
  - Usually, the forward chaining procedure is triggered when a new fact is added to the knowledge base
    - In this case, FC will try to generate all consequences of the new fact (based on existing facts) and adds those which are note already in the KB.

Forward and Backward Chaining

- **Backward Chaining**
  - Start with goal to be proved, apply modus ponens in a backward manner to obtain premises, then try to solve for premises until known facts (already in KB) are reached
  - This is useful for solving for a particular goal
  - In a knowledge-based agent this amounts to applications of the ASK operation
  - The proofs can be viewed as an "AND/OR" tree
    - Root is the goal to be proved
    - For each node, its children are the subgoals that must be proved in order to prove the goal at the current node
    - If the goal is conjunctive (i.e., the premise of rule is a conjunction), then each conjunct is represented as a child and the node is marked as an "AND node" – in this case, both subgoals have to be proved
    - If the goal can be proved using alternative facts in KB, each alternate subgoal is represented as a child and the node is marked as an "OR node" – in this case, only one of the subgoals need to be proved

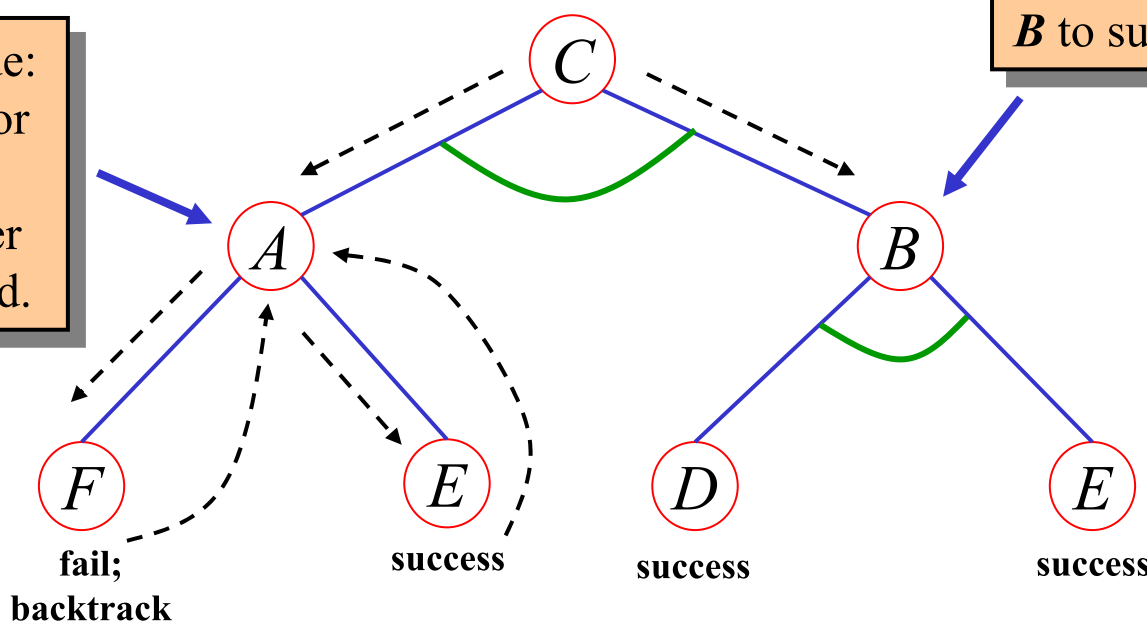
Proof Tree for Backward Chaining

$$KB = \left\{ \begin{array}{ll} 1. & A \wedge B \Rightarrow C \\ 2. & D \wedge E \Rightarrow B \\ 3. & F \Rightarrow A \\ 4. & E \Rightarrow A \\ 5. & E \\ 6. & D \end{array} \right\}$$

prove $KB \models C$

A is an OR node: it's sufficient for one branch to succeed in order for ***A*** to succeed.

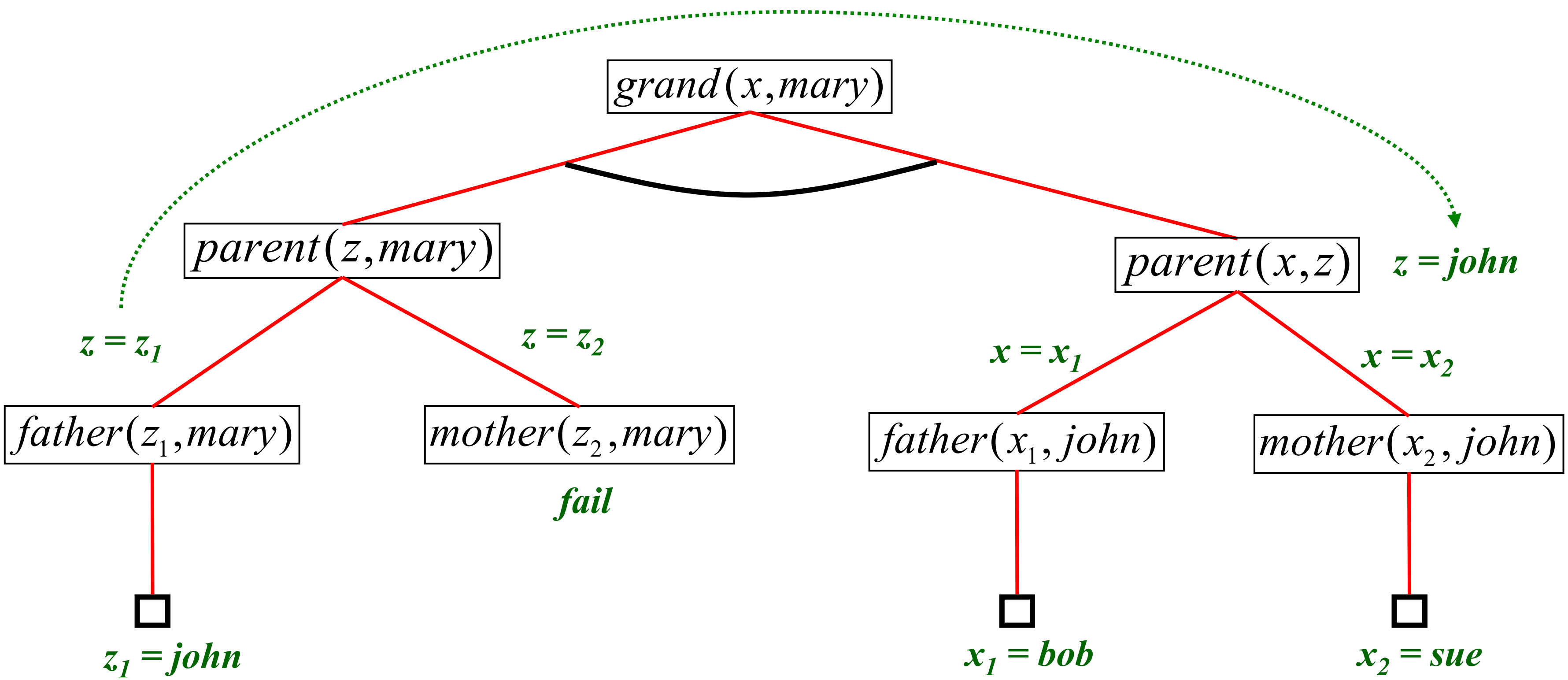
B is an AND node: both branches must succeed in order for ***B*** to succeed.



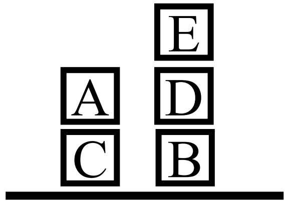
What if clause 4 was *G* => *A* instead?

Proof Tree for Backward Chaining

$$
KB = \left\{
\begin{array}{llll}
1. & father(john, mary) & 2. & mother(sue, john) \\
3. & father(bob, john) & 4. & father(x,y) \Rightarrow parent(x,y) \\
5. & mother(x,y) \Rightarrow parent(x,y) & & \\
6. & parent(x,z) \wedge parent(z,y) \Rightarrow grand(x,y) & &
\end{array}
\right\}
$$

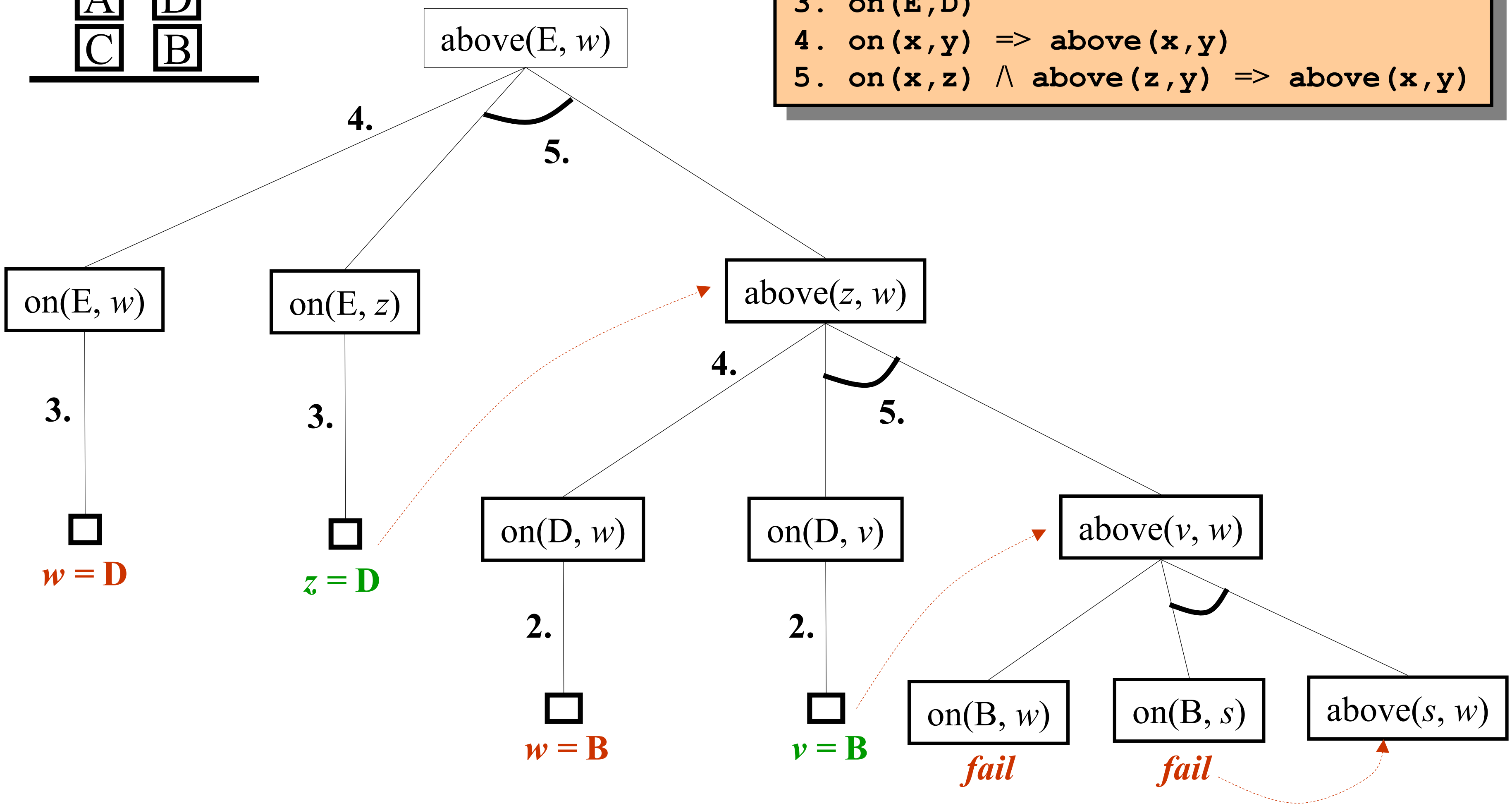


Backward Chaining: Blocks World

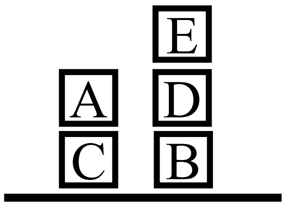


Query: $\exists w$ above(E,w)?

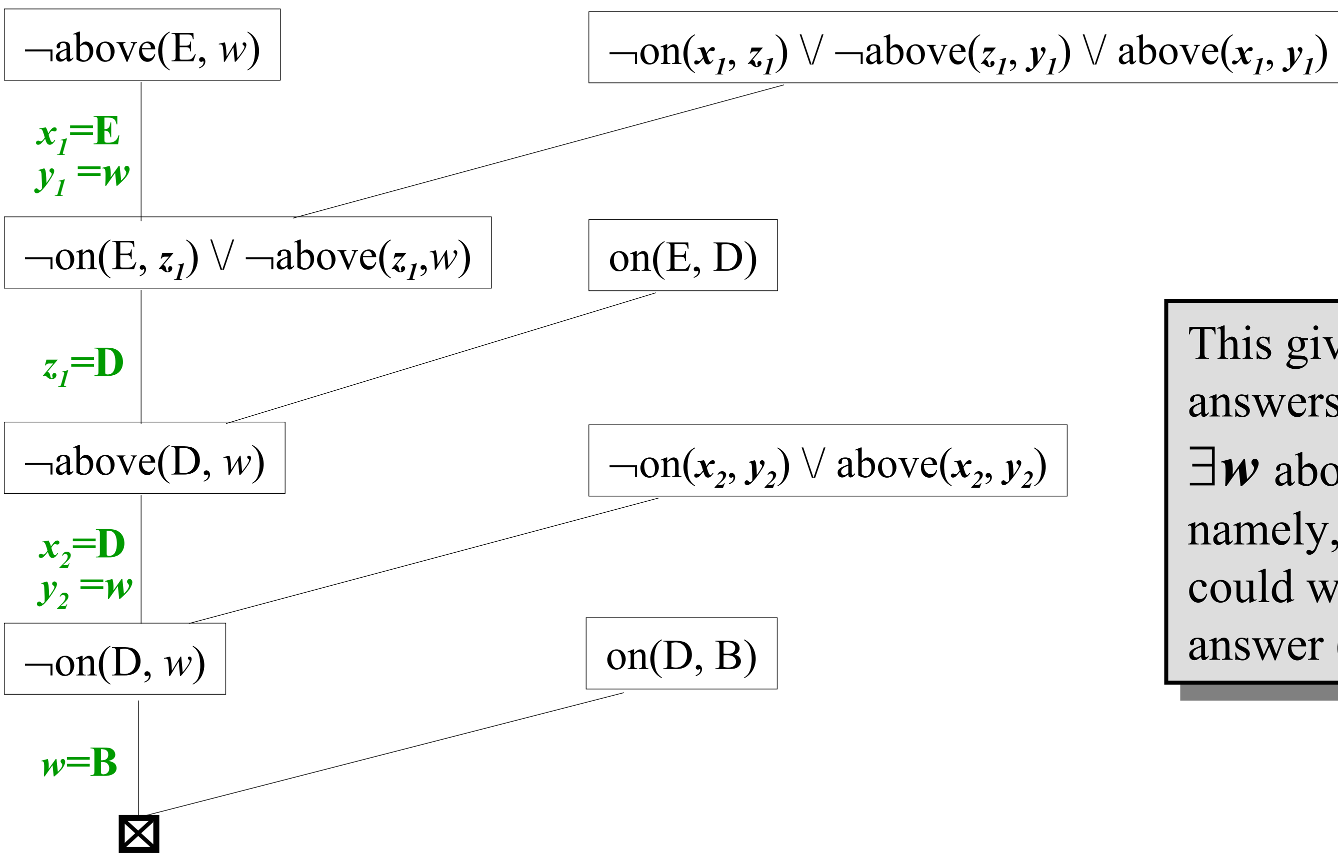
```
1. on(A,C)
2. on(D,B)
3. on(E,D)
4. on(x,y) => above(x,y)
5. on(x,z) /\ above(z,y) => above(x,y)
```



Example: Using Resolution in Blocks World




```
1. on(A,C)    4. ¬on(x,y) ∨ above(x,y)
2. on(D,B)    5. ¬on(x,z) ∨ ¬above(z,y) ∨ above(x,y)
3. on(E,D)
```

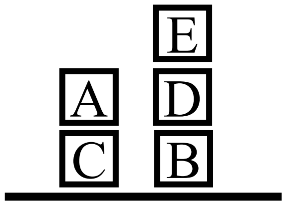


This gives one of the answers to the query $\exists \boldsymbol{w}$ above(E,w), namely, w = B. How could we get the other answer (i.e., w = D)?

A Knowledge-Based Agent for Blocks World

- **Scenario: our agent is a robot that needs to be able to move blocks on top of other blocks (if they are "clear") or onto the floor.**
- **Full axiomatization of the problem requires two types of axioms:**
  - A set of axioms (facts) describing the current state of the world (this includes "definitions" of predicates such as `on`, `above`, `clear`, etc)
  - A set of axioms that describe the effect of our actions
    - in this case, there is one action: "move(x, y)"
    - need axioms that tell us what happens to blocks when they are moved
    - **Important:** in the real implementation of the agent a predicate such as "move(x, y)" is associated with a specific action of the robot which is triggered when the subgoal involving the "move" predicate succeeds.

Agent for Blocks World



onFloor(C)
onFloor(B)
on(A,C)
on(D,B)
on(E,D)

clear(A)
clear(E)

on(*x*,*y*) => above(*x*,*y*)
on(*x*,*z*) /\ above(*z*,*y*) => above(*x*,*y*)

Current state of the world and other things we know.

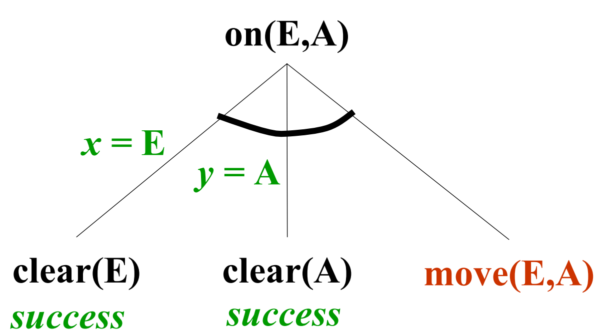
~on(*y*,*x*) => clear(*x*)

Need this to tell us what it means for a block to be “clear.” It also tells us how to clear a block.

clear(*x*) /\ clear(*y*) /\ move(*x*,*y*) => on(*x*,*y*)
clear(*x*) /\ move(*x*,Floor) => onFloor(*x*)
on(*x*,*y*) /\ clear(*x*) /\ move(*x*,Floor) => clear(*y*)
. . .

How actions affect our world

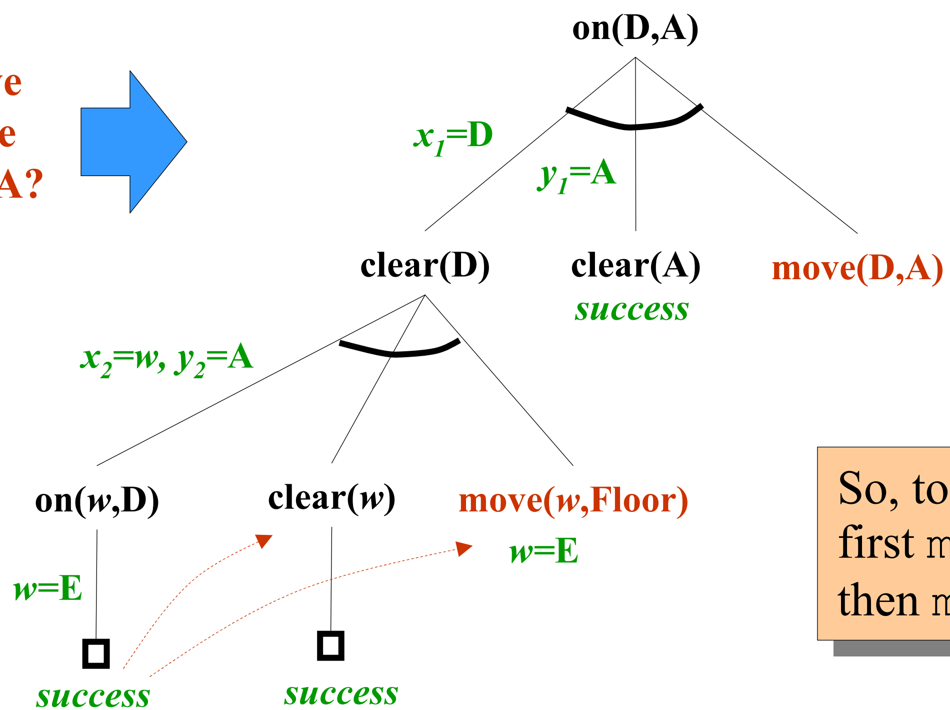
How do we get E to be on top of A?

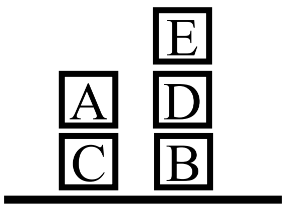


Note that “move” predicate is assumed to always succeed, and is associated with some real operation.

Agent for Blocks World

How do we get D to be on top of A?





So, to get D to be on A, we first `move(E,Floor)` and then `move(D,A)`.

Efficient Control of Reasoning

- **We have seen that during proofs (using resolution or Modus Ponens, etc.), there are different choices we can make at each step**
- **Consider:** $\boldsymbol{house(h, p) \wedge rich(p) \Rightarrow big(h)}$
  - if we want to find h for which $big(h)$ is true, we can do it in two ways
    - 1. find a rich person p, and hope that h will turn-out to be p's house
    - 2. first show h is a house owned by p, then try to show that p is rich
  - usually 2nd approach is more likely to yield a solution; first approach is often too random, but this is not always the case
  - Prolog always takes the left-most subgoal to resolve with a clause in KB
  - we can always order conjuncts on the left: "ordered resolution"
- **Limitations (of controlling the search)**
  - control info. is static (2nd subgoal is deferred and we can't change this during the search)
  - control information is provided by user (in form of axioms, ordering, etc.); we want the computer to do this

Types of Control Strategies

- **Fundamental question is when to make the control decision: 3 possibilities**
  - 1. when the knowledge base is constructed (compile-time or static control)
  - 2. during the search (run-time or dynamic control)
  - 3. when the query appears (hybrid approach)

- **Trade-offs**
  - static is more efficient, but less flexible (less intelligent), since we don't need to figure it out as the interpreter is running
  - dynamic is more flexible, but less efficient and harder to implement
  - hybrid approach may work well if we make the right choice on which part should be static and which part dynamic

Using Statistical Properties of the KB

- **In hybrid approach, ordering of subgoals may depend on statistical properties of the KB**
- **Example:**

$$related(x,y) \land loves(x,y) \Rightarrow family\text{–}oriented(x)$$

  - now suppose:
    - john has a small family and loves some of them
    - mary has a large family, but only loves her cat
  - which ordering to use for queries: *family-oriented*(`john`) and *family-oriented*(`mary`)?
- **For `john`**
  - begin by enumerating relatives and then check to see if he loves any of them
- **For `mary`**
  - better to notice that she only loves her cat, and then check to see that they are not related

Controlling Search at Run-Time

- **Method 1: Forward Checking**
  - basic idea: if during the search we commit to a choice that "we know" will lead to dead end, then we backtrack and make another choice
  - but, how can we "know" this without solving the problem completely?
  - answer: look ahead for a while to make sure that there are potential solutions for other subgoals based on choices made so far
- **Example: crossword puzzle**
  - when filling-in a word, check ahead to make sure that there are still solutions for any crossing word
- **Example:**

$$mother(m,c) \land lives\text{–}at(m,h) \land married(c,s) \land lives\text{–}at(s,h) \Rightarrow sad(s)$$

  - i.e., "people are unhappy if they live with their mothers-in-law;" now suppose we want to find someone who is sad
  - look-ahead here could be checking info. about all marriages, if this information is explicitly state in the KB
  - so, first find a mother and a child; then find out where the mother lives; but what if the child isn't married: no reason to continue; should go back and find another binding for *c*

Controlling Search at Run-Time

- **Method 2: Cheapest-First Heuristic**
  - good idea to first solve terms for which there are only a few solutions; this choice would simultaneously reduce the size of subsequent search space (harder predicates in the conjuncts are solved before they become impossible, so there is less need for backtracking)
- **Example: want to find a carpenter whose father is a senator!!!**

$$carpenter(x) \wedge father(y,x) \wedge senator(y)$$

  - suppose we have the following statistics about the knowledge base

Conjunct	No. of Solutions	
carpenter(x)	10^5	
senator(y)	100	
father(y, x)	10^8	
father(y, `constant`)	1	(a specific person has only one father)
father(`constant`, x)	2.3	(people on average have 2.3 children)

  - in the above ordering, we have 10^5 choices for carpenters, but once we choose one, then there is one choice for a father, and he is either a senator or not (search space: 10^5)
  - but, it is easier to enumerate senators first, then consider the term *father*(`constant`, x); once x has been bound to another constant, it is either a carpenter or it is not (search space: 100 * 2.3 = 230)

Declarative Control of Search

- **How about giving the system declarative information about the problem itself (i.e., include meta-information in the KB)?**
  - We can add control rules which would be treated as other (base-level) rules
  - Problem: we now have to solve the control problem itself
  - When would this be a good idea
- **An Example**
  - Planning a trip: when to head to the airport?
  - We know that flights are scheduled, and we can't control them (this is base-level info.)
  - So, control rule: "when planning a trip, plan the flight first"
  - Note that we used base-level info. to develop a meta-level control rule
- **Problem:**
  - After storing the control rule we have lost the information about its justification
  - Suppose we find out that flights are every 30 minutes, but we can only get a ride to airport between 10 and 11 AM; this suggests that we should first plan out trip to airport
  - But, since the control rule was stored directly in KB, we can't change the control behavior during the execution
- **Principle: if control rules are to be stored, they should be independent of base-level information**

Meta- vs. Base-Level Reasoning Tradeoff

- **The Basic Rule (Computational Principle)**
  - the time spent at meta-level must be recovered at the base-level by finding a quicker (more optimal) path to the solution
  - but, how do we know this without first solving the problem?
    - must somehow determine the "expected" time that will be recovered
  - open problem:
    - we know very little about how this "expected" time should be quantified
- **Two Extremes:**
  - 1. ignore meta-level entirely: take action without worrying about their suitability (shoot from the hip approach), e.g., BFS, DFS
  - 2. work compulsively at meta-level: refuse to take *any* action before *proving* it is the right thing to do
  - Problem with these is that you can always find cases where either is a bad protocol
    - e.g., we could miss easy exam heuristic in case 1: do problems with most points first

Meta-Reasoning (Cont.)

- **The Interleaving Approach**
  - only specific proposal has been to interleave the two approaches, i.e., merge two computational principles
    - 1. never introspect; 2. introspect compulsively
  - shown to give results generally within a factor of two of optimal solution
  - this is the "schizophrenic" AI system approach
    - there are adherents to this idea in psychology: "everyone has two opposite personalities that keep each other in check

- **The Human Model**
  - human problem solvers don't do this kind of interleaving
  - usually start by expecting problem to be easy enough to solve directly; as time passes, spend more time on strategies to solve the problem