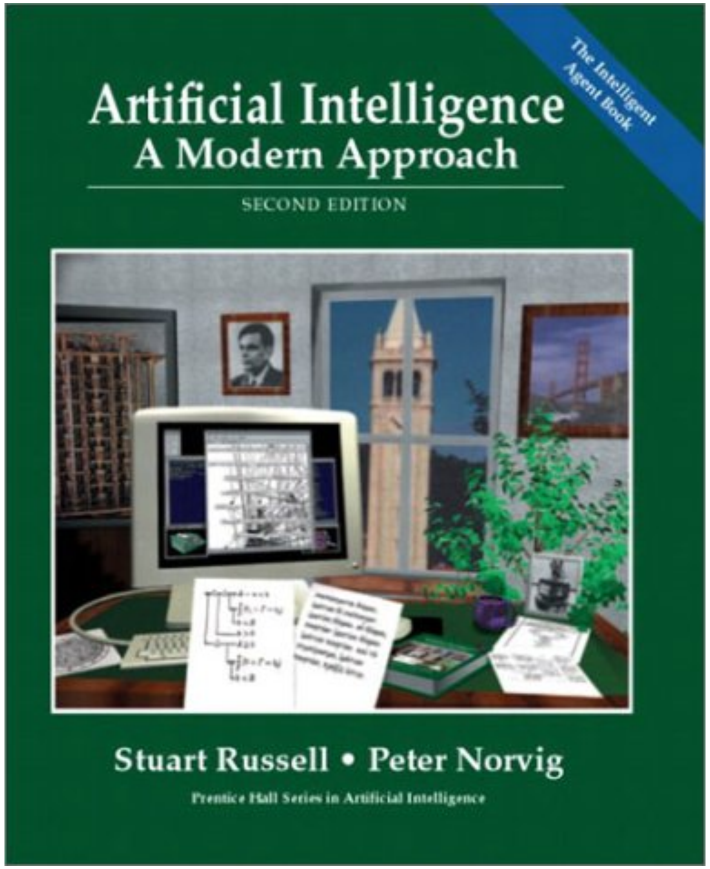
Computational Logic

Learning

R. Moeller

Hamburg University of Technology

Acknowledgments



Slides adapted from an AIMA presentation by

Reijer Grimbergen

http://boole.cs.iastate.edu/book/1-Science/1-ComputerScience/2-Book/Machine%20learning/

Logical description of learning

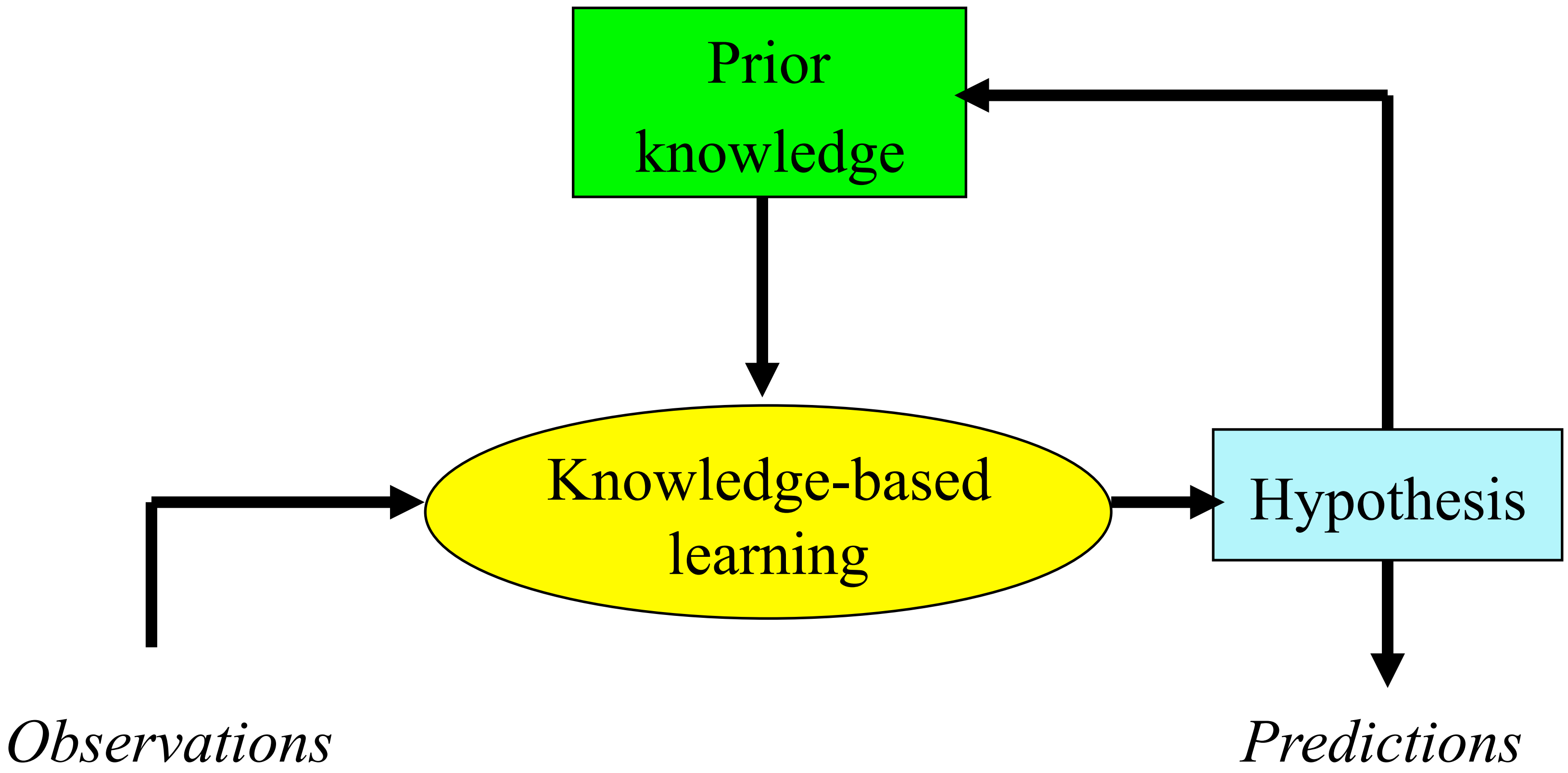
- Examples are composed of descriptions and classifications
  - Objective is to find a hypothesis that explains the classification of the examples, given their descriptions

- Entailment constraint

$$Hypothesis \wedge Descriptions \models Classifications$$

  - With *Descriptions* the conjunction of all the example descriptions and *Classifications* the conjunction of all the example classifications is indicated
  - *Example*: a decision tree that is consistent with all the examples will satisfy the entailment constraint
  - *Note*: Use Ockham's razor to avoid *Hypothesis* = *Classifications*

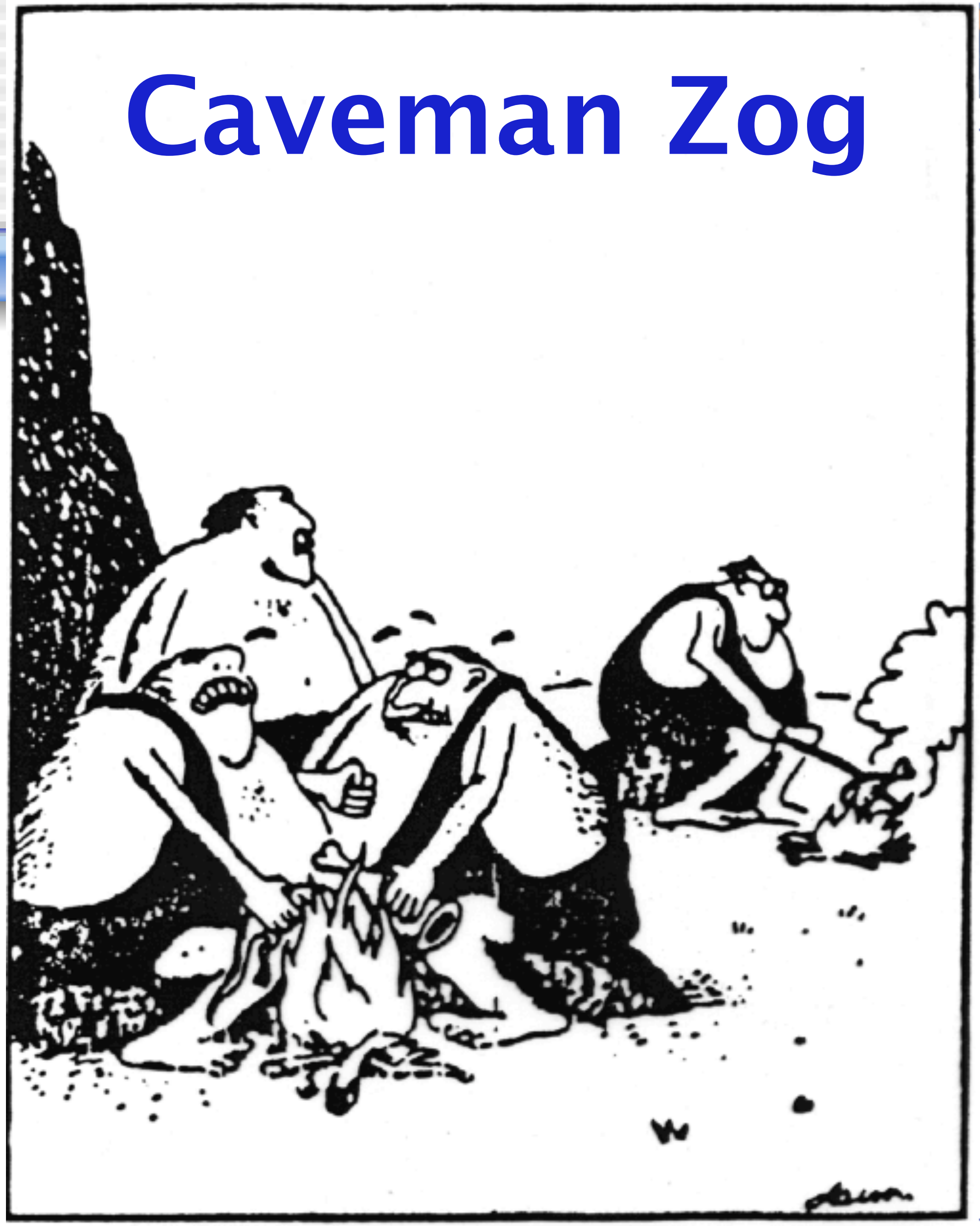
Knowledge-based Learning



Cumulative or Incremental Development

- To use background knowledge, a method to obtain background knowledge is needed
- This must be a learning process
- Use knowledge to learn more effectively
- *Question*: How to do this?
- *Examples where use of background knowledge is vital*
  - Caveman Zog and the lizard on a stick
  - Generalizing from one Brazilian
  - Density and conductance of copper can be generalized, but not mass
  - Inferring a general rule about antibiotic being effective for a particular type of infections

Caveman Zog



"Hey! Look what Zog do!"

Adding Background Knowledge

- Explanation-based learning (EBL)
- Relevance-based learning (RBL)
- Knowledge-based inductive learning (KBIL)

Explanation-based Learning

- Use explanation of success to infer a general rule
- General rule *follows logically* from the background knowledge

$$\textit{Hypothesis} \wedge \textit{Descriptions} \models \textit{Classifications}$$
$$\textit{Background} \models \textit{Hypothesis}$$

- *Does not learn anything factually new*
  - Converting first-principles theories into useful, special purpose knowledge

Relevance-based Learning

- The prior knowledge concerns the *relevance* of a set of features to the goal predicate
  - *Example*: In a given country most people speak the same language, but do not have the same name

Hypothesis ∧ *Descriptions* ⊨ *Classifications*

Background ∧ *Descriptions* ∧ *Classifications* ⊨ *Hypothesis*

- *Deductive learning*: Makes use of the observations, but does not produce hypothesis beyond the background knowledge and the observations

Knowledge-based Inductive Learning

- The background knowledge and the new hypothesis combine to explain the examples
- *Example*
  - Inferring disease *D* from the symptoms is not enough to explain the prescription of medicine *M*
  - A rule that *M* is effective against *D* is needed

$$\textit{Background} \land \textit{Hypothesis} \land \textit{Descriptions} \models \textit{Classifications}$$

Inductive Logic Programming

- Main field of study for KBIL algorithms
- Prior knowledge plays two key roles
  - The effective hypothesis space is reduced to include only those theories that are consistent with what is already known
  - Prior knowledge can be used to reduce the size of the hypothesis explaining the observations
    - Smaller hypotheses are easier to find
- *ILP systems can formulate hypotheses in first-order logic*
  - Can learn in environments not understood by simpler systems

Explanation-based Learning

- Extracting general rules from individual observations
- *Example*: differentiating and simplifying algebraic expressions
  - Differentiate X^2 with respect to X to get $2X$
  - Logical reasoning system

    Ask($Derivative(X^2, X) = d, KB$) with solution $d = 2X$
  - Solving this for the first time using standard rules of differentiation gives $1 \times (2 \times (X^{(2-1)}))$
  - Takes a first-time program 136 proof steps with 99 dead end branches
- Memoization
  - Speed up by saving the results of computation
  - Create a database of input/output pairs

Creating general rules

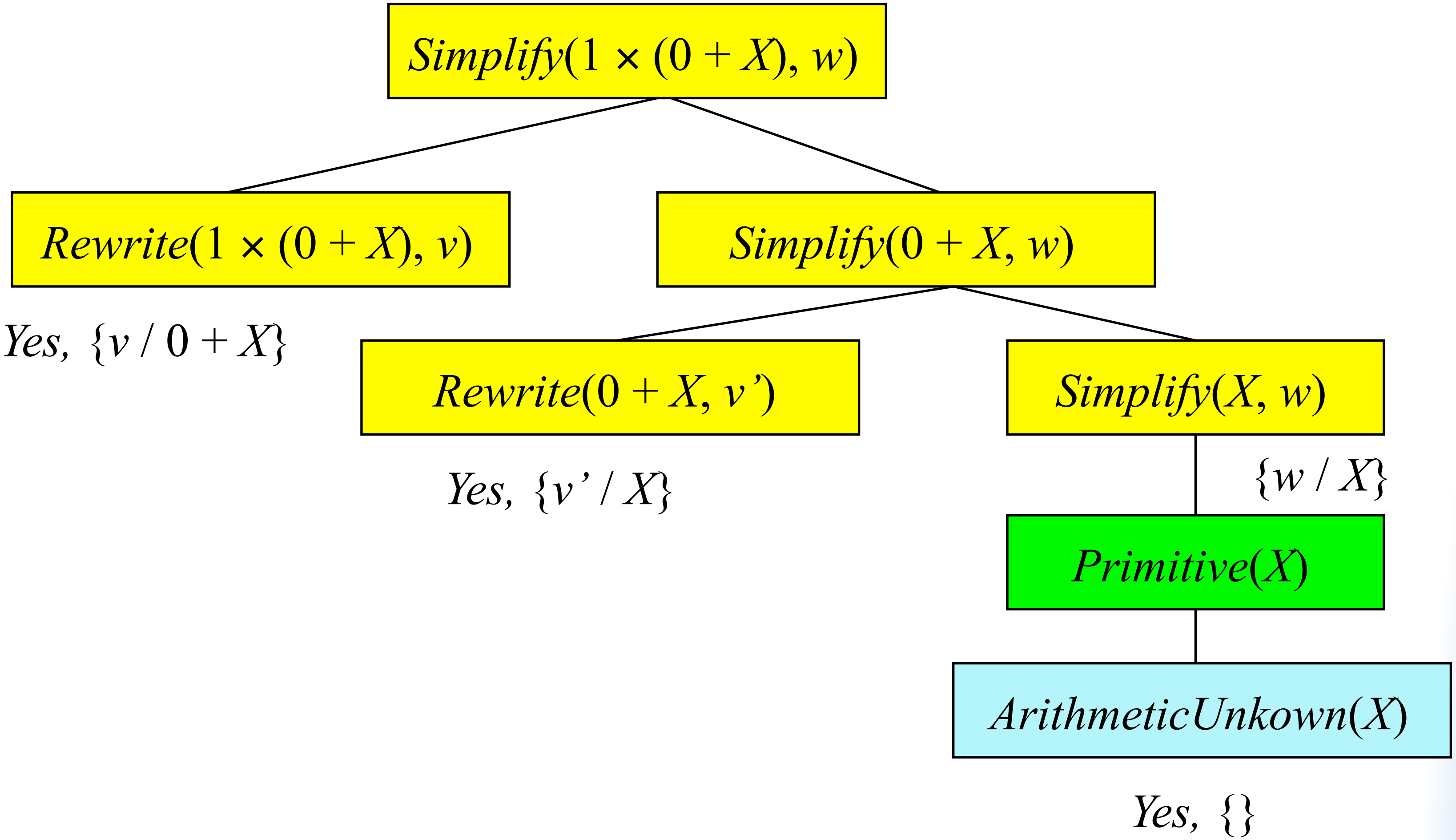
- Memoization in explanation-based learning
  - Create *general rules* that cover an entire class of cases
  - *Example*: extract the general rule

    $ArithmeticUnknown(u) \Rightarrow Derivative(u^2, u) = 2u$
- *Once something is understood, it can be generalized and reused in other circumstances*
  - "Civilization advances by extending the number of important operations that we can do without thinking about them"
- *Explaining why something is a good idea is much easier than coming up with the idea in the first place*
  - Watch caveman Zog roast his lizard vs. thinking about putting the lizard on a stick

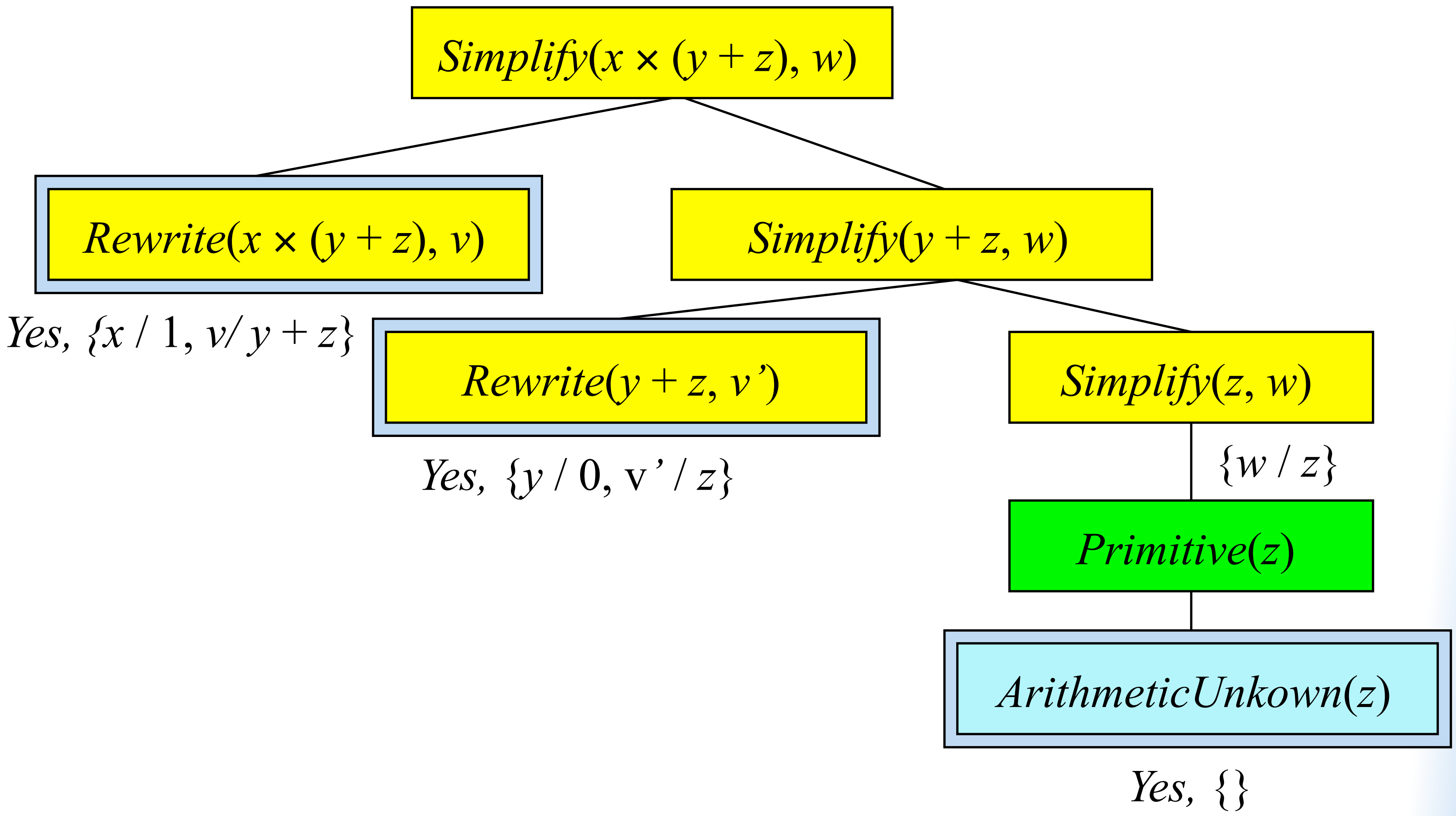
Extracting rules from examples

- Basic idea behind EBL
  - Construct an explanation of the observation using prior knowledge
  - Establish a definition of the class of cases for which the same explanation can be used
- *Example*: simplifying $1 \times (0 + X)$ using a knowledge base with the following rules
  - $Rewrite(u, v) \land Simplify(v, w) \Rightarrow Simplify(u, w)$
  - $Primitive(u) \Rightarrow Simplify(u, u)$
  - $ArithmeticUnknown(u) \Rightarrow Primitive(u)$
  - $Number(u) \Rightarrow Primitive(u)$
  - $Rewrite(1 \times u, u)$
  - $Rewrite(0 + u, u)$
  - …

Proof tree for original problem



Generalized proof tree



Generalizing proofs

- *The variabilized proof proceeds using exactly the same rule applications*
  - May lead to variable instantiation
- *Take the leaves of the generalized proof tree to get the general rule*

  $Rewrite(1 \times (0 + z), 0 + z) \wedge Rewrite(0 + z, z) \wedge ArithmeticUnknown(z) \Rightarrow Simplify(1 \times (0 + z), z)$

  - The first two conditions are independent of *z*, so this becomes

    $ArithmeticUnknown(z) \Rightarrow Simplify(1 \times (0 + z), z)$

- Recap
  - Use background knowledge to construct a proof for the example
  - In parallel, construct a generalized proof tree
  - New rule is the conjunction of the leaves of the proof tree and the variabilized goal
  - Drop conditions that are true regardless of the variables in the goal

Improving efficiency

- Pruning the proof tree to get more general rules

  $Primitive(z) \Rightarrow Simplify(1 \times (0 + z), z)$

  $Simplify(y + z, w) \Rightarrow Simplify(1 \times (y + z), w)$

- *Problem*: Which rules to choose?
  - Adding large numbers of rules to the knowledge base slows down the reasoning process (increases the *branching factor* of the search space)
  - To compensate, the derived rules must offer significant speed increases
  - Derived rules should be as general as possible to apply to the largest possible set of cases

Improving efficiency

- Operationality of subgoals in the rule
  - A subgoal must be "easy" to solve
  - *Primitive*(*z*) is easy to solve, but *Simplify*(*y* + *z*, *w*) leads to an arbitrary amount of inference
  - Keep operational subgoals and prune the rest of the tree
- Trade-off between operationality and generality
  - More specific subgoals are easier to solve but cover fewer cases
  - How many steps are still called operational?
  - Cost of a subgoal depends on the rules in the knowledge base

⇒ Maximizing the efficiency of an initial knowledge base is a complex optimization problem

Improving efficiency

- Empirical analysis of efficiency
  - Average-case complexity on a population of problems that needs to be solved
- *By generalizing from past example problems, EBL makes the knowledge base more efficient for the kind of problems that it is reasonable to expect*
  - Works if the distribution of past problems is roughly the same as for future problems
  - Can lead to great improvement
    - Swedish to English translator was made 1200 times faster by using EBL

Recap: Relevance-based Learning

- The prior knowledge concerns the *relevance* of a set of features to the goal predicate
- *Example*: In a given country most people speak the same language, but do not have the same name

Hypothesis ∧ *Descriptions* ⊨ *Classifications*

Background ∧ *Descriptions* ∧ *Classifications* ⊨ *Hypothesis*

- *Deductive learning*: Makes use of the observations, but does not produce hypothesis beyond the background knowledge and the observations

Relevance-based Learning

- Functional dependencies or determinations
  - Background knowledge in Brazil example

$$\forall_{x,y,n,l} Nationality(x,n) \wedge Nationality(y,n) \wedge Language(x,l) \Rightarrow Language(y,l)$$

  - Therefore, from

$$Nationality(Fernando, Brazil) \wedge Language(Fernando, Portuguese)$$

  it follows

$$\forall_x Nationality(x, Brazil) \Rightarrow Language(x, Portuguese)$$

- Special syntax

$$Nationality(x,n) \succ Language(x,l)$$

Determining the hypothesis space

- Determinations limit the hypothesis space
  - Only consider the important features (i.e. not day of the week, hair style of David Beckham)
- *Determinations specify a sufficient basis vocabulary from which to construct hypotheses*
- Reduction of the hypothesis space makes it easier to learn the target predicate
  - Learning boolean functions of n variables in CNF: Size of the hypothesis space $|\mathbf{H}| = O(2^{2^n})$
  - For boolean functions $\log(|\mathbf{H}|)$ examples are needed in a $|\mathbf{H}|$ size hypothesis space: Without restrictions, this is $O(2^n)$ examples
  - If the determination contains d predicates on the left, only $O(2^d)$ examples are needed
  - Reduction of size $O(2^{n-d})$

Learning relevance information

- Prior knowledge also needs to be learned
- Learning algorithm for determinations
  - Find the simplest determination consistent with the observations
  - A determination $P \succ Q$ says that if examples match *P* they must also match *Q*
  - *A determination is consistent with a set of examples if every pair that matches on the predicates on the left-hand side also matches on the target predicate*

Learning relevance information

Sample	Mass	Temp	Material	Size	Conductance
S1	12	26	Copper	3	0.59
S1	12	100	Copper	3	0.57
S2	24	26	Copper	6	0.59
S3	12	26	Lead	2	0.05
S3	12	100	Lead	2	0.04
S4	24	26	Lead	4	0.05

- *Minimal consistent determination*

  $$Material \wedge Temperature \succ Conductance$$

- *Non-minimal consistent determination*

  $$Mass \wedge Size \wedge Temperature \succ Conductance$$

Learning relevance information

```
function Minimimal-Consistent-Det(E, A) returns a determination
    inputs: E, a set of examples
            A, a set of attributes, of size n
    for i ← 1, …, n do
        for each subset A_i of A of size i do
            if Consistent-Det?(A_i ,E) then return A_i
        end
end

function Consistent-Det?(A, E) returns a truth-value
    inputs: A, a set of attributes
            E, a set of examples
    local variables: H, a hash table
    for each example e in E do
        if some example in H has the same value as e for the attributes A
           but a different classification then return False
        store the class of e in H, indexed by the values for attributes A of the example e
    end
    return True
```

Complexity

- Time complexity depends on the size of the minimal consistent determination
  - In case of *p* attributes and a total of *n* attributes, the algorithm has to search all subsets of *A* of size *p*
  - *There are $O(n^p)$ of these, so the algorithm is exponential*
  - *The general problem is NP-complete*
  - In most domains there is sufficient local structure to make *p* small

Deriving Hypotheses

- Use decision tree learning for computing hypotheses
- Goal: Minimize size of hypotheses
- Idea: Use relevance-based decision tree learning

Relevance-based Decision Tree Learning

function RBDTL(*E, A, v*) **returns** a decision tree
  return DTL(*E*, Minimal-Consistent-Det(*E,A), v*)

function DTL(*examples, attributes, default*) **returns** a decision tree
  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MODE(*examples*)
  else
    best ← CHOOSE-ATTRIBUTE(*attributes, examples*)
    tree ← a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DTL($examples_i$, *attributes* − *best*, MODE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    return *tree*

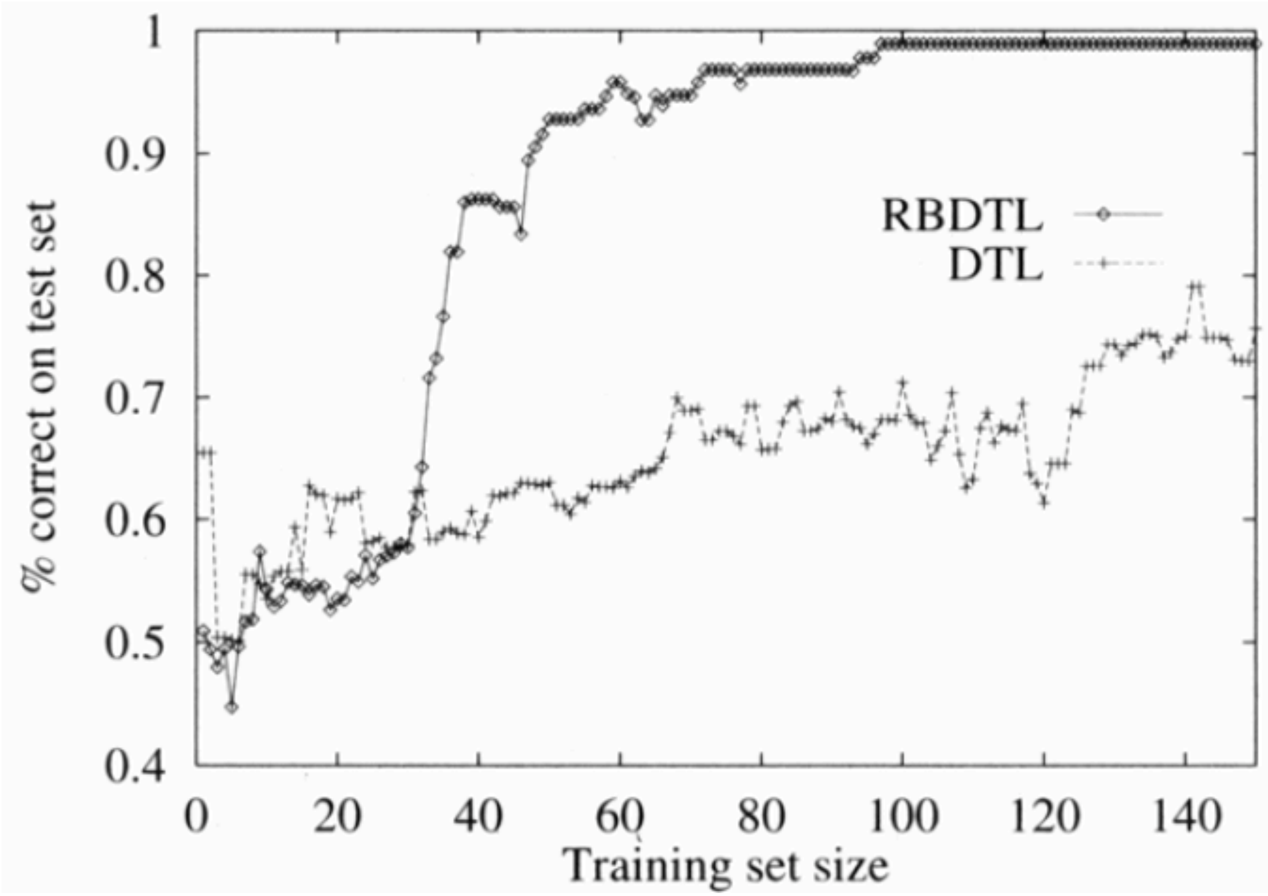
MODE(·) = Majority(·)

Exploiting Knowledge

- RBDTL simultaneously learns and uses relevance information to minimize its hypothesis space

- Declarative bias
  - How can prior knowledge be used to identify the appropriate hypothesis space to search for the correct target definition?
  - *Unanswered questions*
    - How to handle noise?
    - How to use other kinds of prior knowledge besides determinations?
    - How can the algorithms be generalized to cover any first-order theory?

RBDTL vs. DTL



RBDTL
DTL
% correct on test set
Training set size
1
0.9
0.8
0.7
0.6
0.5
0.4
0
20
40
60
80
100
120
140

Inductive Logic Programming

- Combines inductive methods with the power of first-order representations
- Offers a rigorous approach to the general KBIL problem
- Offers complete algorithms for inducing general, first-order theories from examples

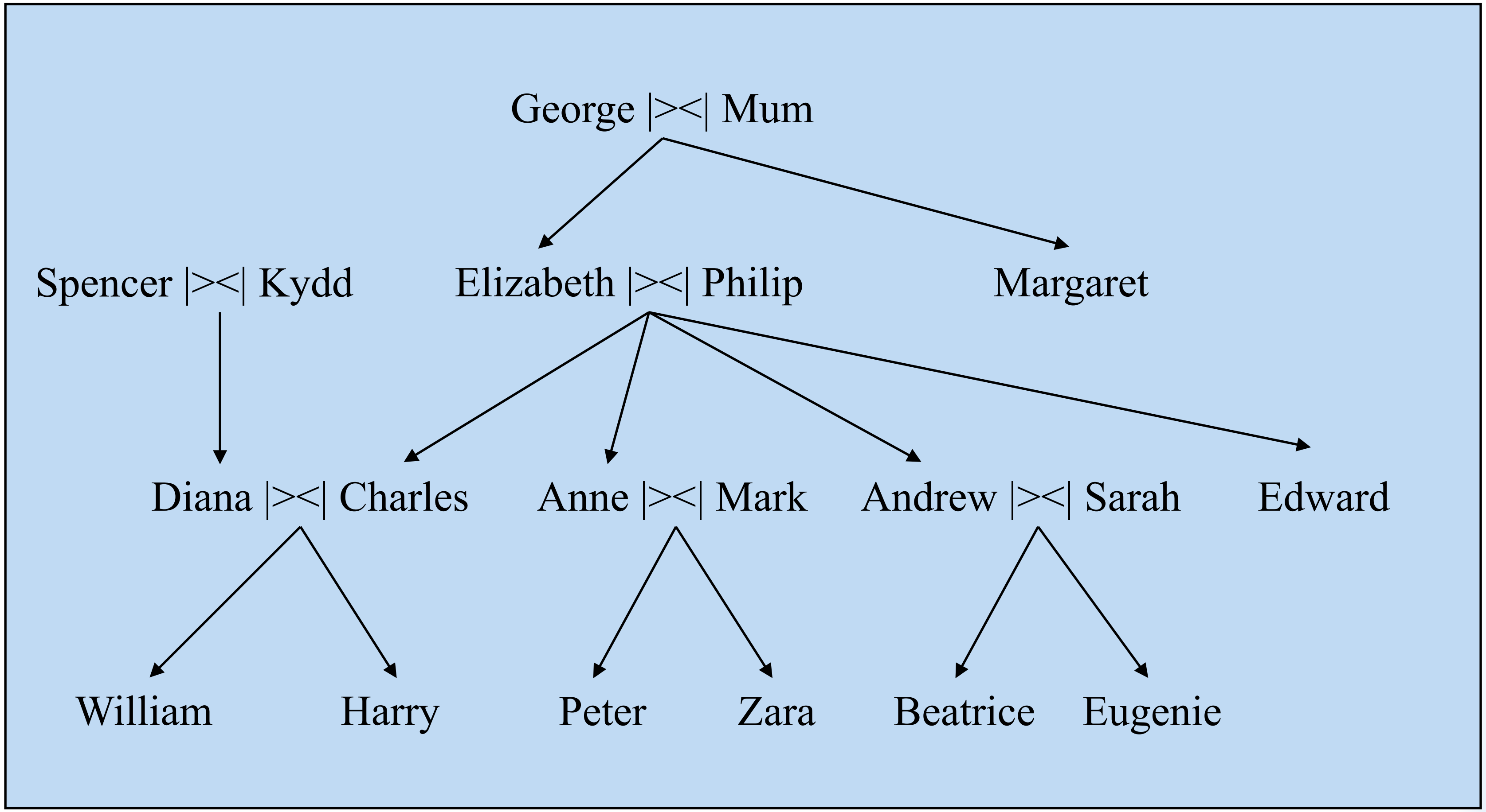
ILP: An example

- General knowledge-based induction problem

$$Background \wedge Hypothesis \wedge Descriptions \models Classifications$$

- *Example*: Learning family relations from examples
  - Observations are an extended family tree
    - *Mother, Father* and *Married* relations
    - *Male* and *Female* properties
  - Target predicates: *Grandparent, BrotherInLaw, Ancestor*

Example (prob. not up to date)



Example

- *Descriptions* include facts like
  - *Father(Philip, Charles)*
  - *Mother(Mum, Margaret)*
  - *Married(Diana, Charles)*
  - *Male(Philip)*
  - *Female(Beatrice)*
- Sentences in *Classifcations* depend on the target concept being learned (in the example: 12 positive, 388 negative)
  - *Grandparent(Mum, Charles)*
  - ¬*Grandparent(Mum, Harry)*
- *Goal*: find a set of sentences for *Hypothesis* such that the entailment constraint is satisfied
  - Without background knowledge this is for example

$$\begin{aligned} Grandparent(x,y) \Leftrightarrow & [\exists_z Mother(x,z) \wedge Mother(z,y)] \\ \vee & [\exists_z Mother(x,z) \wedge Father(z,y)] \\ \vee & [\exists_z Father(x,z) \wedge Mother(z,y)] \\ \vee & [\exists_z Father(x,z) \wedge Father(z,y)] \end{aligned}$$

Why Attribute-based Learning Fails

- Decision-Tree-Learning will get nowhere
  - To express *Grandparent* as a (boolean) attribute, pairs of people need to be objects
    Grandparent(<*Mum,Charles*>)
  - But then the example descriptions can not be represented
    FirstElementIsMotherOfElizabeth(<*Mum,Charles*>)
  - A large disjunction of specific cases without any hope of generalization to new examples

Attribute-based learning algorithms are incapable of learning relational predicates

Background knowledge

- A little bit of background knowledge helps a lot
  - Background knowledge contains

$$Parent(x, y) \Leftrightarrow [Mother(x, y) \vee Father(x, y)]$$

  - *Grandparent* is now reduced to

$$Grandparent(x,y) \Leftrightarrow [\exists_z Parent(x,z) \wedge Parent(z,y)]$$

- Constructive induction algorithm
  - Create new predicates to facilitate the expression of explanatory hypotheses
  - *Example*: introduce a predicate *Parent* to simplify the definitions of the target predicates

Top-down inductive learning

- Top-down learning method
  - *Decision-tree learning*: start from the observations and work backwards
    - Decision tree is gradually grown until it is consistent with the observations
  - *Top-down learning*: start from a general rule and specialize it

Top-Down Inductive Learning: FOIL

- Split positive and negative examples
  - Positive: *<George, Anne>, <Philip, Peter>, <Spencer, Harry>*
  - Negative: *<George, Elizabeth>, <Harry, Zara>, <Charles, Philip>*
- Construct a set of Horn clauses with *Grandfather(x,y)* as the head with the positive examples instances of the *Grandfather* relationship
  - Start with a clause with an empty body

    $\Rightarrow$ *Grandfather(x,y)*
  - All examples are now classified as positive, so specialize to rule out the negative examples: Here are 3 potential additions:

    1) *Father(x,y)* $\Rightarrow$ *Grandfather(x,y)*
    2) *Parent(x,z)* $\Rightarrow$ *Grandfather(x,y)*
    3) *Father(x,z)* $\Rightarrow$ *Grandfather(x,y)*
  - The first one incorrectly classifies the 12 positive examples
  - The second one is incorrect on a larger part of the negative examples
  - Prefer the third clause and specialize

    Father(x,z) $\wedge$ *Parent(z,y)* $\Rightarrow$ *Grandfather(x,y)*

FOIL

function Foil(*examples, target*) **returns** a set of Horn clauses
  inputs: *examples*, set of examples
    target, a literal for the goal predicate
  local variables: *clauses,* set of clauses, initially empty
  while *examples* contains positive examples **do**
    clause ← New-Clause(*examples, target*)
    remove examples covered by *clause* from *examples*
    add *clause* to *clauses*
  return *clauses*

FOIL

function New-Clause(*examples, target*) **returns** a Horn clause
  local variables:
    clause, a clause with *target* as head and an empty body
    l, a literal to be added to the clause
    extended-examples, a set of examples with values for new variables
  extended-examples ← *examples*
  while *extended-examples* contains negative examples **do**
    l ← Choose-Literal(New-Literals(*clause*), *extended-examples*)
    append *l* to the body of *clause*
    extended-examples ← set of examples created by applying Extend-Example to each example in *extended-examples*
  return *clause*

FOIL

function Extend-Example(*example, literal)* **returns**

if *example* satisfies *literal*

then return the set of examples created by extending *example* with each possible constant value for each new variable in *literal*

else return the empty set

FOIL

- New-Literals
  - Takes a clause and constructs all possible “useful” literals
- *Example*: *Father(x,z)* ⇒ *Grandfather(x,y)*
  - *Add literals using predicates*
    - Negated or unnegated
    - Use any existing predicate (including the goal)
    - Arguments must be variables
    - Each literal must include at least one variable from an earlier literal or from the head of the clause
    - Valid: *Mother(z,u), Married(z,z), Grandfather(v,x)*
    - Invalid: *Married(u,v)*
  - *Equality and inequality literals*
    - E.g. $z \neq x$, empty list
  - *Arithmetic comparisons*
    - E.g. $x > y$, threshold values

FOIL

- The way New-Literal changes the clauses leads to a very large branching factor
- Improve performance by using type information
  - E.g., *Parent(x,n)* where *x* is a person and *n* is a number
- Choose-Literal uses a heuristic similar to information gain
- Ockham's razor to eliminate hypotheses
  - If the clause becomes longer than the total length of the positive examples that the clause explains, this clause is not a valid hypothesis
- *Most impressive demonstration*
  - Learn the correct definition of list-processing functions in Prolog from a small set of examples, using previously learned functions as background knowledge

Inverse Resolution

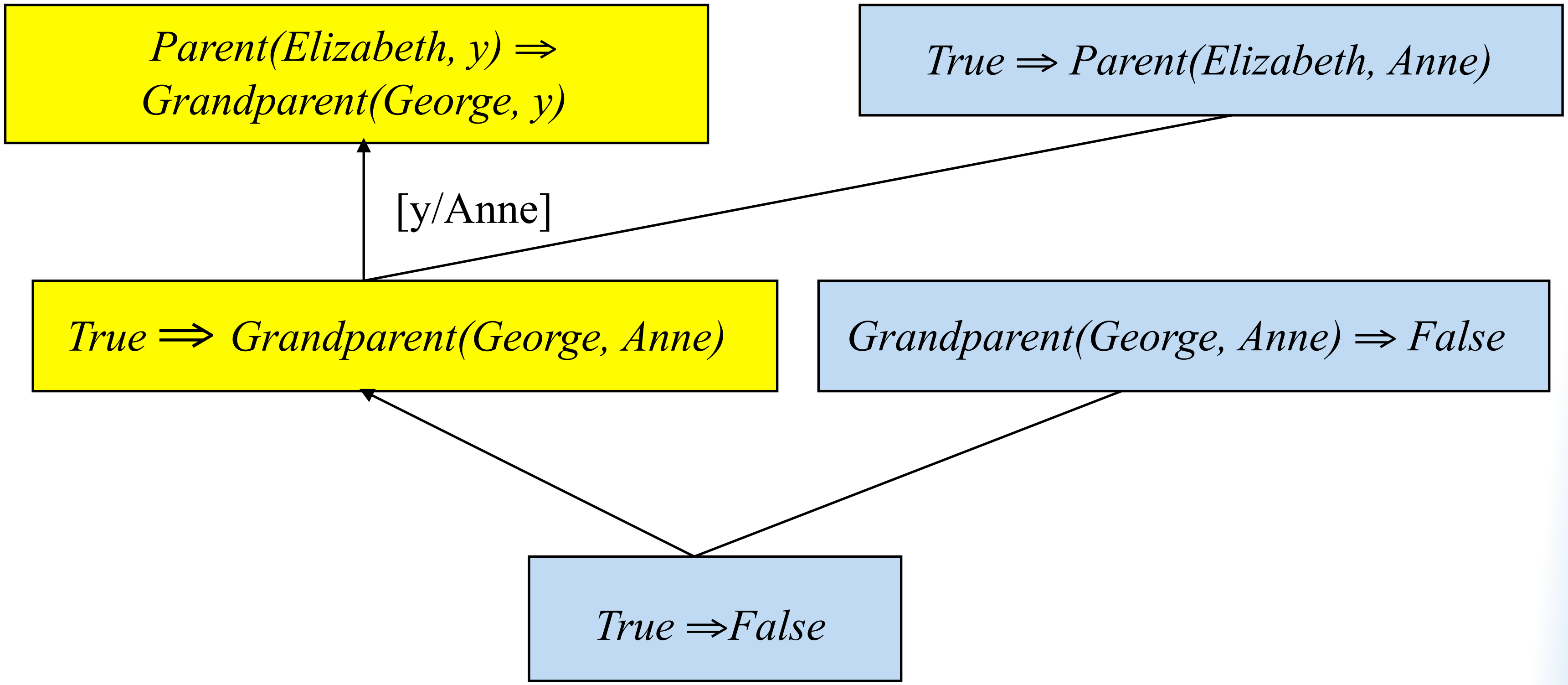
- Inverse resolution
  - *Classifications* follows from

    Background ∧ *Hypothesis* ∧ *Descriptions*
  - This can be proven by resolution
  - Run the proof backwards to find *Hypothesis*
  - *Problem*: How to run the proof backwards?

Generating Inverse Proofs

- Ordinary resolution
  - Take two clauses C_1 and C_2 and resolve them to produce the *resolvent* C
- Inverse resolution
  - Take resolvent C and produce two clauses C_1 and C_2
  - Take C and C_1 and produce C_2

Generating Inverse Proofs



Generating Inverse Proofs

- Inverse resolution is a search
  - For any *C* and C_1 there can be several or even an infinite number of clauses C_2
    - Instead of *Parent(Elizabeth,y)* ⇒ *Grandparent(George,y)* there were numerous alternatives

      Parent(Elizabeth,Anne) ⇒ *Grandparent(George,Anne)*

      Parent(z,Anne) ⇒ *Grandparent(George,Anne)*

      Parent(z,y) ⇒ *Grandparent(George,y)*
  - The clauses C_1 that participate in each step can be chosen from *Background, Descriptions, Classifications* or from hypothesized clauses already generated
- ILP needs restrictions to make the search manageable
  - Eliminate function symbols
  - Generate only the most specific hypotheses
  - Use Horn clauses
  - All hypothesized clauses must be consistent with each other
  - Each hypothesized clause must agree with the observations

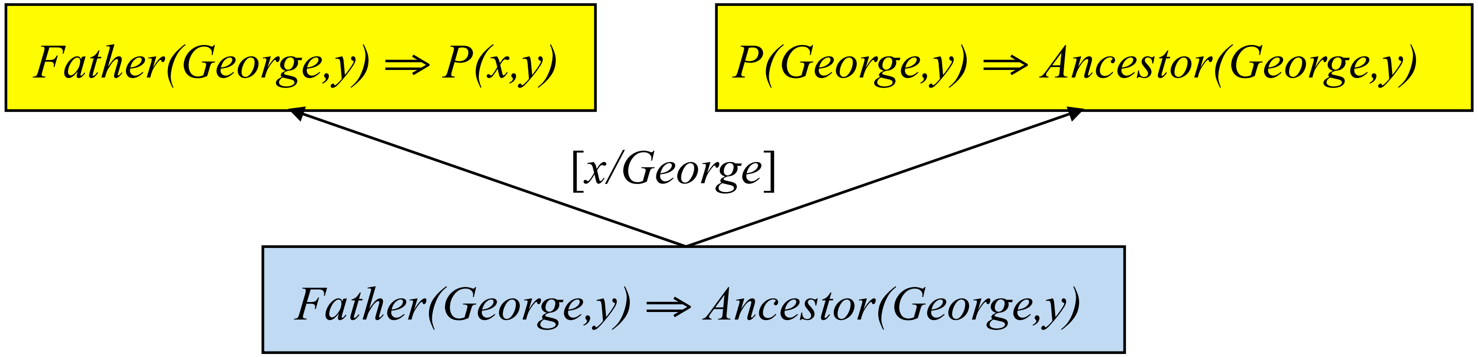
New Predicates and New Knowledge

- An inverse resolution procedure is a complete algorithm for learning first-order theories
  - If some unknown *Hypothesis* generates a set of examples, then an inverse resolution procedure can generate *Hypothesis* from the examples
- Can inverse resolution infer the law of gravity from examples of falling bodies?
  - Yes, given suitable background mathematics
- *Monkey and typewriter problem*: How to overcome the large branching factor and the lack of structure in the search space?

New Predicates and New Knowledge

- Inverse resolution is capable of generating new predicates
  - Resolution of C_1 and C_2 into C eliminates a literal that C_1 and C_2 share
  - This literal might contain a predicate that does not appear in C
  - When working backwards, one possibility is to generate a new predicate from which to construct the missing literal

New Predicates and New Knowledge



- *P* can be used in later inverse resolution steps
  - *Example*: *Mother(x,y)* ⇒ *P(x,y)* or *Father(x,y)* ⇒ *P(x,y)* leading to the "Parent" relationship
- <u>Inventing new predicates is important to reduce the size of the definition of the goal predicate</u>
  - Some of the deepest revolutions in science come from the invention of new predicates (e.g. Galileo's invention of acceleration)

Applications

- ILP systems have outperformed knowledge-free methods in a number of domains
  - *Molecular biology*: the GOLEM system has been able to generate high-quality predictions of protein structures and the therapeutic efficacy of various drugs
  - GOLEM is a completely general-purpose program that is able to make use of background knowledge about any domain

Knowledge in Learning: Summary

- Cumulative learning
  - Improve learning ability as new knowledge is acquired
- *Prior knowledge helps to eliminate hypothesis and fills in explanations, leading to shorter hypotheses*
- Entailment constraints
  - Logical definition of different learning types
- Explanation-based learning (EBL)
  - Explain the examples and generalize the explanation
- Relevance-base learning (RBL)
  - Use prior knowledge in the form of determinations to identify the relevant attributes
- Knowledge-based inductive learning (KBIL)
  - Finds inductive hypotheses that explain sets of observations
- Inductive logic programming (ILP)
  - Perform KBIL using knowledge expressed in first-order logic
  - Generates new predicates with which concise new theories can be expressed