Analytical Learning

Machine learning:

- Inductive methods
  - ▷ Data-intensive
  - ▷ Extract a general description of a *concept* from many examples
- Deductive methods
  - ▷ Knowledge-intensive
  - ▷ Explain and analyze single example of instance of concept
  - ▷ Explanation identifies the relevant features of the example = sufficient conditions for describing the concept.
  - ▷ Generalize instantiated explanation to apply to other instances of the concept.

Explanation-Based Generalization – EBG

Inputs:

- Target concept definition
- Training example
- Domain theory
- Operationality criterion

Output:

Generalization of the training example, that is

- sufficient to describe the target concept, and
- satisfies the operationality criterion.

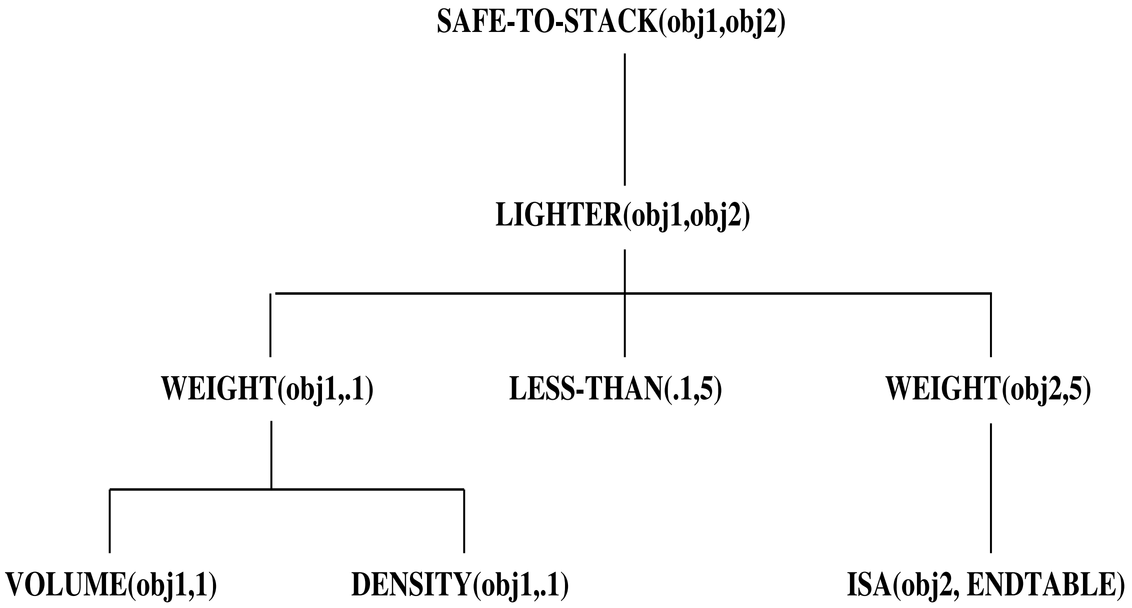
The SAFE-TO-STACK Example

Input:

- **target concept:** SAFE-TO-STACK(x,y)
- **training example:**
  ON(OBJ1,OBJ2)
  ISA(OBJ1, BOX) ISA(OBJ2, ENDTABLE)
  COLOR(OBJ1, RED) COLOR(OBJ2, BLUE)
  VOLUME(OBJ1,1) DENSITY(OBJ1,0.1) ...
- **domain theory:**
  1. NOT(FRAGILE(y)) or LIGHTER(x,y) $\rightarrow$ SAFE-TO-STACK(x,y)
  2. VOLUME(x,v) and DENSITY(x,d) $\rightarrow$ WEIGHT(x,v*d)
  3. WEIGHT(x1,w1) and WEIGHT(x2,w2) and LESS(w1,w2) $\rightarrow$ LIGHTER(x1,x2)
  4. ISA(x,ENDTABLE) $\rightarrow$ WEIGHT(x,5)
  5. LESS(0.1,5) ...
- **operationality criterion:**
  learned description should be built of *terms* used to describe examples directly, or other “easily” evaluated, such as `LESS`.

The SAFE-TO-STACK Example

- Explain why `obj1` is `SAFE-TO-STACK` on `obj2`.
  - ▷ Construct a proof.
  - ▷ Do **Goal regression**: regress target concept through proof structure.
  - ▷ Proof isolates *relevant* features.



Generating Operational Knowledge

- Generalize proof:
  - ▷ Sometimes simply replace constants by variables.
  - ▷ Prove that all identified relevant features are necessary in general (hard! – may need a lot of "extra" knowledge, *domain axioms*).

Output:

```
VOLUME(x,v1) and DENSITY(x,d1) and
ISA(y,ENDTABLE) and
and LESS(v1*d1,5) → SAFE-TO-STACK(x,y)
```

EBL: A Deductive Learning Method

Why are examples needed?

- Domain theory contains all the information: simply operationalize target concept.
- Examples help to <u>focus</u> on the relevant operationalizations:
  characterize only examples that actually occur.

Actual purpose of EBL:

▷ <u>not</u> to "learn" more about target concept,

▷ <u>but</u> to "re-express" target concept in a more operational manner (=efficiency).

▷ <u>control learning</u>.

EBL Applied to Problem Solving/Planning

Inputs:

- Target concept definition – decision to be made
- Training example:
  - ▷ The search episode with its successes and failures
- Domain theory:
  - ▷ Operators used in the search
  - ▷ Objects and possibly relationships in the world which may be used to build the explanation
- Operationality criterion:
  - ▷ Describe concept using terms that are interpretable (efficiently) by the problem solver
  - ▷ Several possible criteria

EBL **Applied to Problem Solving/Planning**

Output:

Generalization of the training example, that is

- Sufficient to describe the target concept,
- and satisfies the operationality criterion.

1. **Explain** (prove) why example is instance of target concept.
   - uses domain theory (set of operators)
   - prunes away unimportant aspects of example
   - final explanation is operational
2. **Generalize** explanation

EBL **in** PRODIGY

Goal: – improve the efficiency of the planner
– learn *control rules*.

- knowledge-intensive approach
- analyzes trace of solving a problem
- explains "why" the choices made during problem solving were, or were not, appropriate
- acquires control knowledge – *better* search heuristics

Control rule:

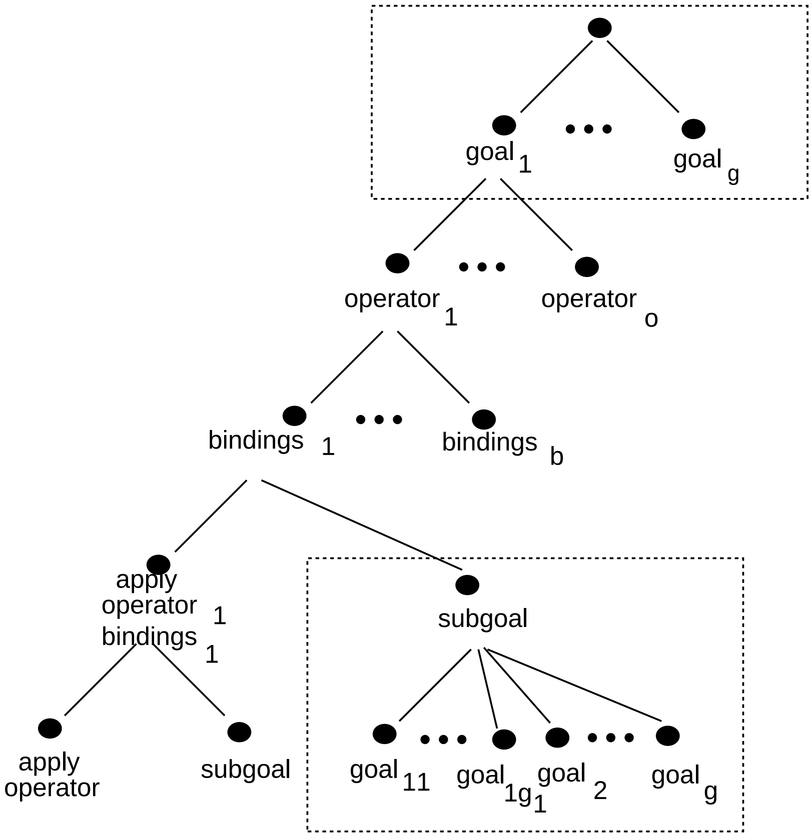
- Applies at individual decisions.
- Antecedent matches the state of the planner at decision making time.
- Antecedent is operational – planner can match its state using control rule language.
- Consequent *selects*, *rejects* or *prefers* particular alternatives.

Examples of Control Rules in PRODIGY

```
(CONTROL-RULE SELECT-OP-UNSTACK-FOR-HOLDING
      (if (and (current-goal (holding <x>))
                (true-in-state (on <x> <y>))))
      (then select operator UNSTACK))

(CONTROL-RULE SELECT-BINDINGS-UNSTACK-HOLDING
      (if (and (current-goal (holding <x>))
                (current-ops (UNSTACK))
                (true-in-state (on <x> <y>))))
      (then select bindings ((<ob> . <x>) (<underob> . <y>))))

(CONTROL-RULE SELECT-OP-PUTDOWN-FOR-ARMEMPTY
      (if (and (current-goal (arm-empty))
                (true-in-state (holding <ob>))))
      (then select operator PUT-DOWN))

(CONTROL-RULE SELECT-BINDINGS-PUTDOWN
      (if (and (current-ops (PUT-DOWN))
                (true-in-state (holding <x>))))
      (then select bindings ((<ob> . <x>))))
```

Target concepts

- Select goal *goal*
- Select operator *op* for achieving *goal*
- Select bindings for operator *op* and goal *goal*
- Decide subgoal if *op* is applicable
- Decide apply *op*



Discussion

- Very successful in a variety of domains.
- Learned rules are applied as other rules, i.e. if their antecedent *totally* matches planning situation.
- If EBL system is eager to learn provably correct knowledge, the explanation effort is really large and the EBL system requires a *complete* domain theory for generalization.

Utility problem: The more rules learned, the slower the deliberation

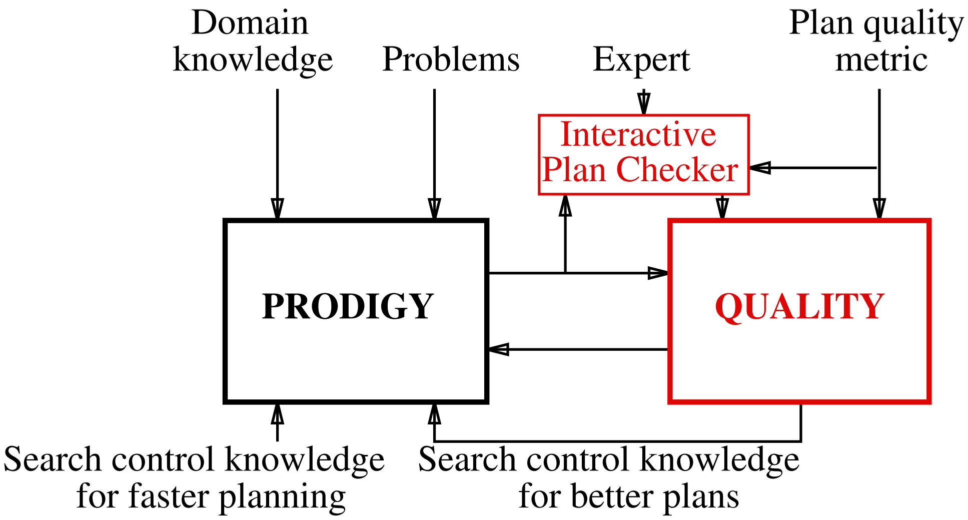
- Possible solutions:
  - ▷ Perform utility analysis and discard low-utility rules
  - ▷ Heuristics to learn only effective knowledge
  - ▷ Incremental refinement of learned rules
- Factors influencing utility of control knowledge
  - ▷ Matching cost (cost of utilization)
  - ▷ Frequency of application
  - ▷ Savings every time it is applied
  - ▷ Organization of learned rules!

Learning To Achieve Plan Quality

- Beyond learning to improve problem solving *efficiency*.
- Real-world applications begin to require *good quality* solutions.
- Interactions among goals and scenarios affect the quality of solutions
  - ▷ Explicit goal interactions – efficiency
  - ▷ Quality goal interactions (harder to learn)
- Plan length might not be the only cost measure
- Two approaches:
  - ▷ QUALITY learns from the difference between a *good* solution and a *worse* solution [Pérez 95]
  - ▷ HAMLET learns to select alternatives that lead to optimal solutions [Borrajo & Veloso 94, 96]

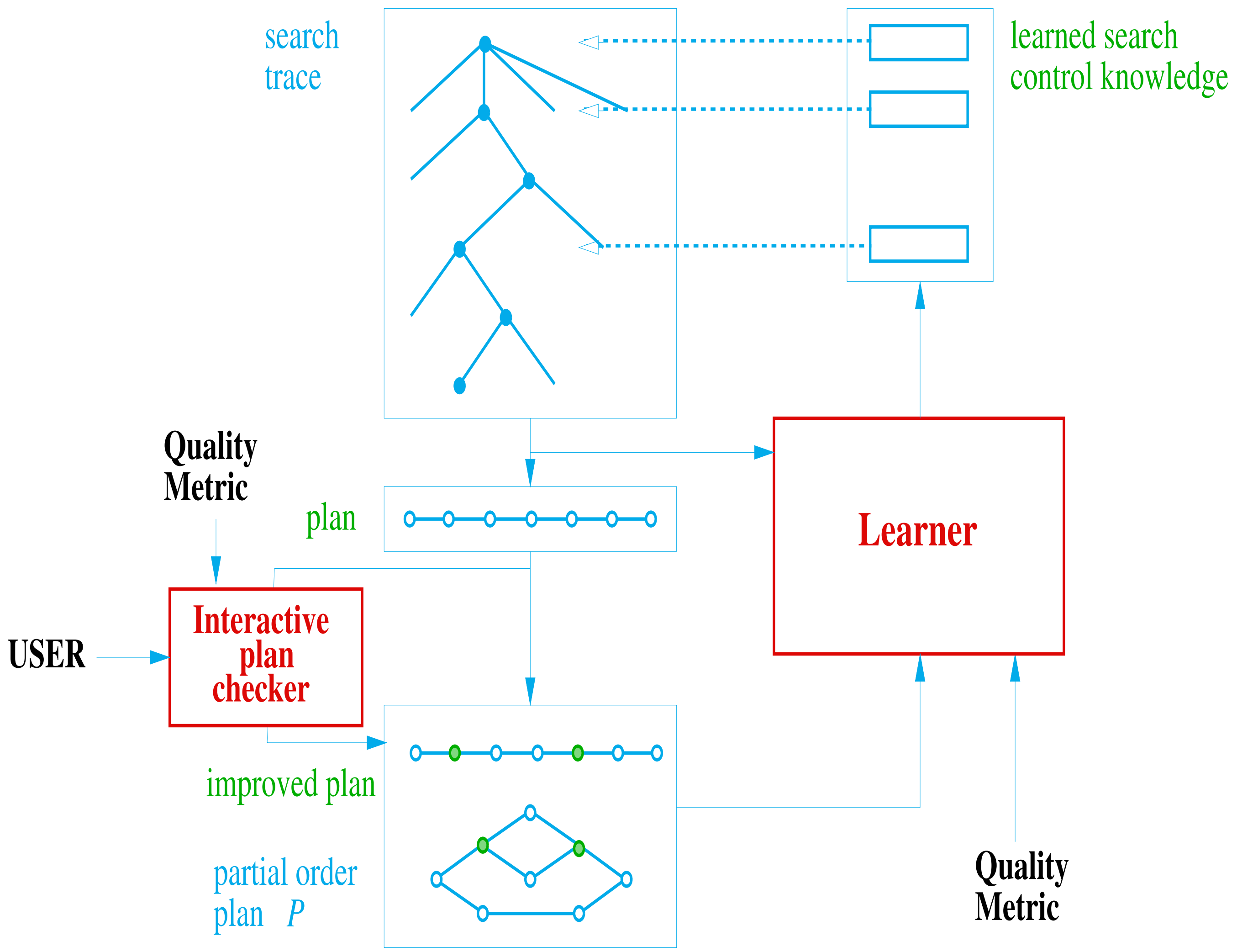
Learning to Generate Quality Plans

Learn **control knowledge** to guide future search towards better plans (instead of possible "post-facto" plan modification).



- **Learning = change of representation:**
  From quality metric **into** search control knowledge available at problem solving decision time, since plan and search tree are only partially available.

QUALITY: The Architecture [Pérez 95]



Discussion

- Learn control rules to prefer operators, bindings, and goals in **domain-independent** fashion.
- Learning is **driven by failure**, when current control strategy must be overridden.

- If the **quality metric changes**, the learned knowledge is invalidated and re-learned.
- **Limited class** of quality metrics.
  - Tradeoffs in the quality factors lead to conflicts between rules.
  - Non-local tradeoffs are hard to capture with local control rules.

Solution: algorithms to learn and use **control-knowledge trees**.

Learning in Planning Search Trees [Borrajo & Veloso 95]

Labeling procedure:

- Find failure and successes to learn from
- Traverse trace (in post-order) labeling each node (failure, success, unknown).

Generation of control rules:

- Identify relevant features by goal regression
- Generalize instances in rules
- Left hand side (antecedent): conjunction of relevant features
- Right hand side (consequent): the decision learned

Outcome:

- Learned rules may be overspecific, i.e. may have a superset of the real relevant features.
- Learned rules may be overgeneral, i.e. may have a subset of the real relevant features (when applied to nonlinear planning)

Example - The Logistics Domain

- Packages are moved between cities. Trucks carry packages between locations within a city and airplanes carry packages across cities.
- There is no knowledge about
  - ▷ not moving carriers if they need to be loaded
  - ▷ unload a truck if an object is in the same city
  - ▷ load two objects "at the same time" if they need to go to the same place, and they are in the same place
- Changing representation is an open research option that we are also exploring

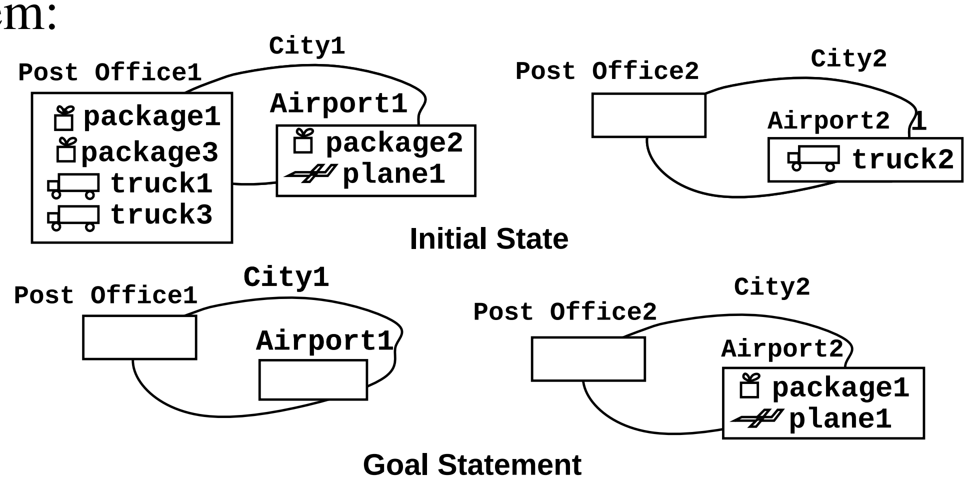
```
(OPERATOR FLY-AIRPLANE
   (params ((<plane> AIRPLANE)
      (<loc-from> AIRPORT)
      (<loc-to> AIRPORT))
   (preconds
     (at-airplane <plane> <loc-from>))
   (effects ((add (at-airplane <plane> <loc-to>))
             (del (at-airplane <plane> <loc-from>)))))
```

Other Logistics Domain Operators

```
(OPERATOR UNLOAD-AIRPLANE
   (params ((<obj> object) (<plane> airplane) (<loc> airport))
   (preconds (and (at-airplane <plane> <loc>)
                  (inside-airplane <obj> <plane>)))
   (effects ((del (inside-airplane <obj> <plane>))
             (add (at-object <obj> <loc>)))))

(OPERATOR LOAD-TRUCK
   (params ((<obj> object) (<truck> truck) (<loc> location))
   (preconds (and (at-truck <truck> <loc>)
                  (at-object <obj> <loc>)))
   (effects ((del (at-object <obj> <loc>))
             (add (inside-truck <obj> <truck>)))))

(OPERATOR DRIVE-TRUCK
   (params ((<truck> truck) (<loc-from> location)
             (<loc-to> location))
   (preconds (and (same-city <loc-from> <loc-to>)
                  (at-truck <truck> <loc-from>)))
   (effects ((del (at-truck <truck> <loc-from>))
             (add (at-truck <truck> <loc-to>)))))
```

Basic EBL is Over-General

Problem:



If only caring for efficiency, EBL learns the following rule:

(control-rule select-bind-fly-airplane-2
  (if (current-operator fly-airplane)
      (current-goal (at-airplane <plane1> <airport3>))
      (true-in-state (at-airplane <plane1> <airport2>)))
  (then select bindings ((<plane> . <plane1>)
                          (<loc-from> . <airport2>)
                          (<loc-to> . <airport3>))))

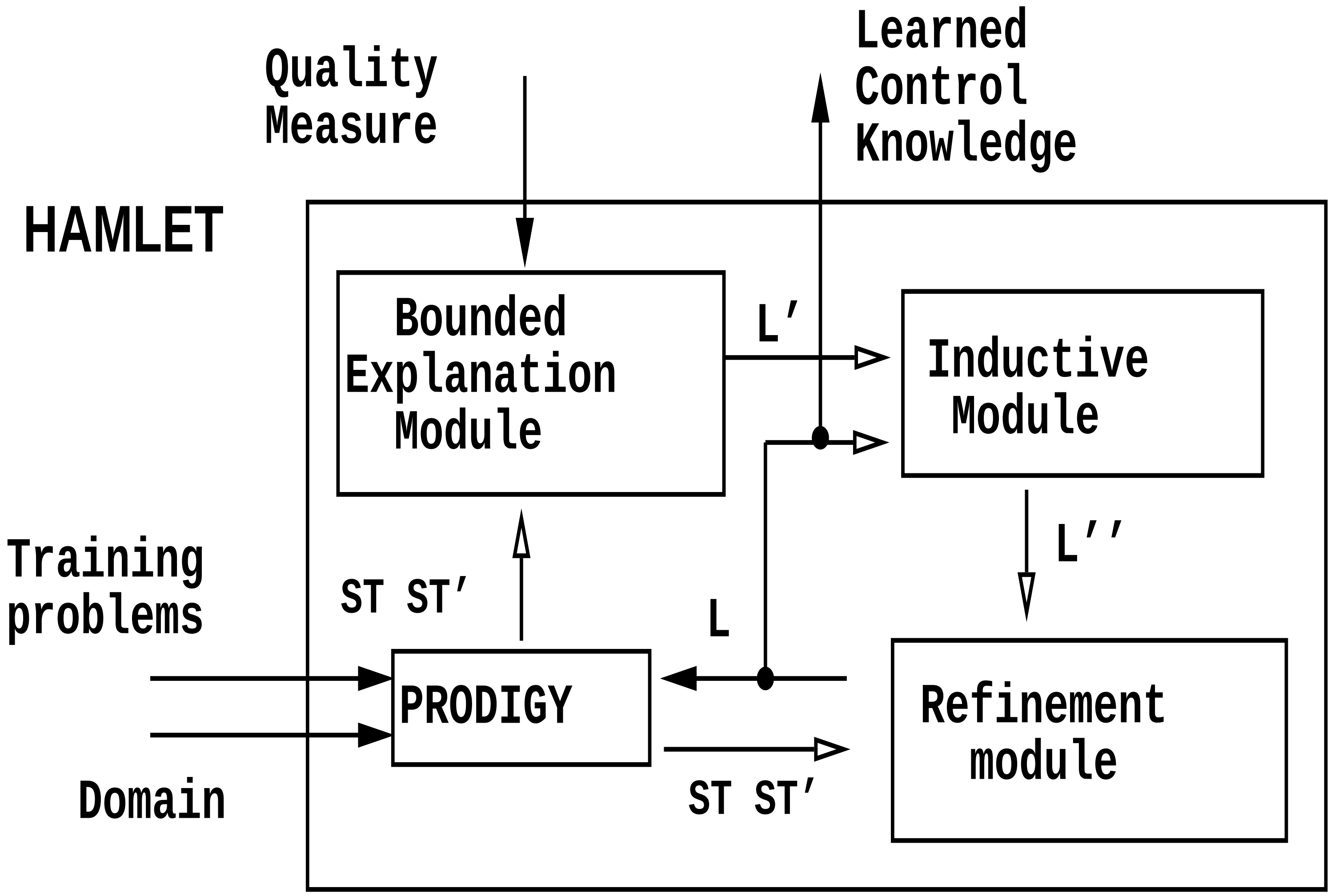
HAMLET: Deduction and Induction

- Extend the basic EBL approach developed for linear problem solving
  - ▷ Define new learning opportunities
  - ▷ Consider solution quality
- Reduce the explanation effort
  - ▷ No need to acquire extra domain knowledge
- Incrementally refine control knowledge
  - ▷ Converges towards an experience-supported correct set of rules

Rule learned by HAMLET (previous example):

```
(control-rule select-bind-fly-airplane-1
  (if (current-operator fly-airplane)
      (current-goal (at-airplane <plane1> <airport3>))
      (true-in-state (at-airplane <plane1> <airport2>))
      (true-in-state (at-object <package4> <airport1>))
      (other-goals ((at-object <package4> <airport3>))))
  (then select bindings ((<plane> . <plane1>)
                         (<loc-from> . <airport1>)
                         (<loc-to> . <airport3>))))
```

HAMLET's Architecture



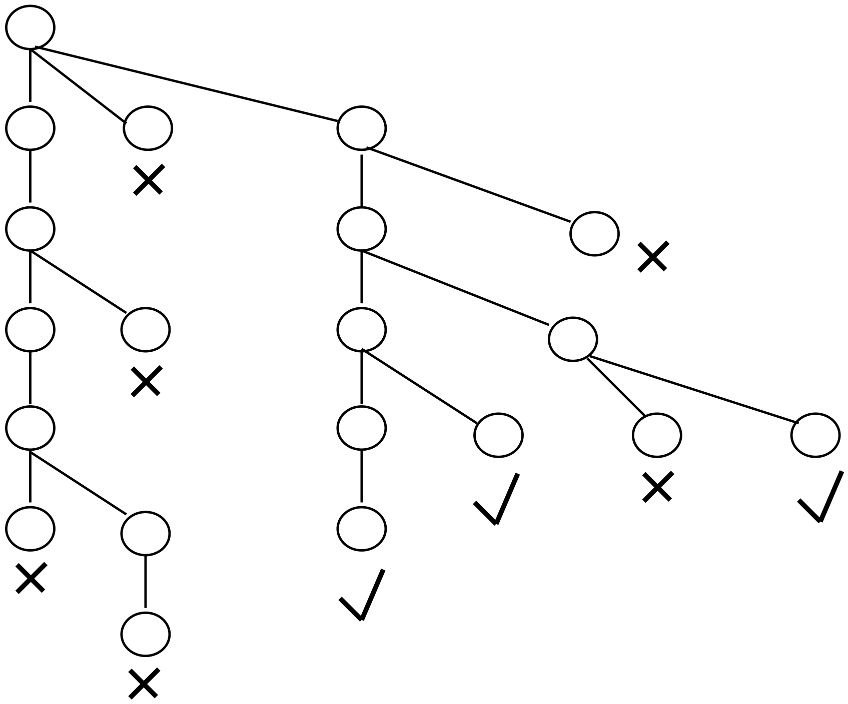
HAMLET's Algorithm

Let L refer to the set of learned control rules.
Let ST, ST' refer to search trees.
Let P be a problem to be solved.
Let Q be a quality measure.
Initially L is empty.
For all P in training problems
    ST = Result of solving P without any rules.
    ST' = Result of solving P with current set of rules L.
    If positive-examples-p(ST, ST',Q)
    Then L' = Bounded-Explanation(ST, ST',Q)
        L''= Induce(L,L')
    If negative-examples-p(ST, ST',Q)
    Then L=Refine(ST, ST',L'')

Bounded Explanation Module

- HAMLET's characteristics
  - ▷ no need for extra domain knowledge
  - ▷ reduced explanation effort
  - ▷ convergence towards correctness
- Bounded explanation steps
  - ▷ Labeling the decision tree. *Eagerness*
  - ▷ Credit Assignment. *Optimal learning*
  - ▷ Generation of control rule. *Goal Regression*
  - ▷ Parametrization. *Variable differentiation*

A Typical Search Tree



What are the learning opportunities?

Induction Module

- Why induction?
  - ▷ Bounded explanation generates possibly over-specific rules
- HAMLET does induction over
  - ▷ State
  - ▷ Subgoaling structure
  - ▷ Interacting goals
  - ▷ Type hierarchy
- Inductive operators
  - ▷ Deletion of rules that subsume others
  - ▷ Intersection of preconditions. *state*
  - ▷ Refinement of subgoaling dependencies. *prior goal*
  - ▷ Relaxing the subgoaling dependencies. *prior goal*
  - ▷ Refinement of the set of interacting goals. *other goals*
  - ▷ Find common superclass. *type of object*

Inducing Over Two Rules

- Old rule:

```
(control-rule select-unload-airplane-1
    (if (current-goal (at-object <object1> <airport2>))
        (true-in-state (at-airplane <plane4> <airport3>))
        (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
```

- New rule:

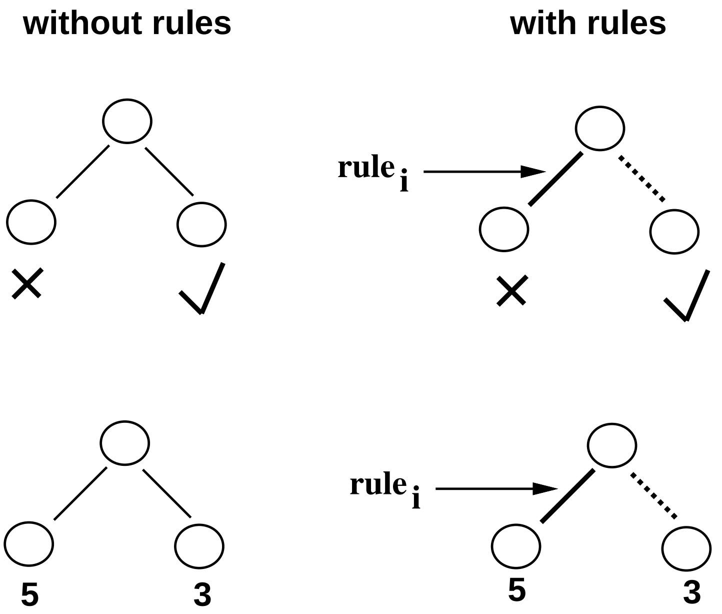
```
(control-rule select-unload-airplane-2
    (if (current-goal (at-object <object1> <airport2>))
        (true-in-state (at-airplane <plane4> <airport5>))
        (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
```

- Induced rule:

```
(control-rule induced-select-unload-airplane-3
    (if (current-goal (at-object <object1> <airport2>))
        (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
```

Refining

- Why refinement?
  - ▷ HAMLET may produce over-general rules
- Negative examples: occasions in which control rules have been applied and should have not
- A negative example for HAMLET is
  - ▷ Situation in which a control rule was applied, and
  - ▷ the resulting decision led to a *failure*, or
  - ▷ the resulting decision led to a *worse solution* than the best one for that decision



Overgeneralization

- Induced rule

```
(control-rule induced-select-unload-airplane-3
    (if (current-goal (at-object <object1> <airport2>))
        (true-in-state (at-object <object1> <airport3>)))
    (then select operators unload-airplane))
```

- New rule

```
(control-rule induced-select-unload-airplane-4
    (if (current-goal (at-object <object1> <airport2>))
        (true-in-state (inside-airplane <object1> <plane4>)))
    (then select operators unload-airplane))
```

- Overgeneral rule

```
(control-rule induced-select-unload-airplane-5
    (if (current-goal (at-object <object1> <airport2>)))
    (then select operators unload-airplane))
```

Empirical Results

Test sets		Unsolved problems		Solved by both (279 problems, 53.14%)					
				Better solutions		Solution length		Nodes explored	
Goals	Problems	without rules	with rules	without rules	with rules	without rules	with rules	without rules	with rules
1	100	5	0	0	11	327	307	2097	1569
2	100	15	6	0	25	528	479	3401	2308
5	100	44	18	1	33	865	777	5170	3463
10	100	68	32	1	24	770	668	3482	2941
20	75	62	36	0	10	505	455	2216	1924
50	50	49	40	0	0	34	34	143	141
Totals	525	243	132	2	103	3029	2720	16509	12346
%		46.3%	25.1%	0.7%	36.9%			Ratio	1.3

Training problems	Unsolved problems		Solved by both				
			Better solutions		Ratio Solution Length	Ratio Time	Ratio Nodes
	without rules	with rules	without rules	with rules	without/ with rules	without/ with rules	without/ with rules
75	46.29 %	36.38 %	0.35 %	25.89 %	1.11	0.49	1
150	46.29 %	34.29 %	0.72 %	31.9 %	1.06	0.33	1.25
400	46.29 %	25.14 %	0.72 %	36.92 %	1.08	0.32	1.34

Summary – Analytical Learning

- Long-term goal of automating planning efficiency.
- Knowledge in domain theory is not usually effective.
- Explain examples to produce operational control knowledge for decisions.
- Provably correct explanations that generalize to new situations are hard to learn.
- Difficult goal and operator choice interactions can be learned through a combined deductive and inductive approach.
- User's quality metrics can be cast in the learned knowledge.