An Architectural Account of Variation in Problem Solving and Execution

Pat Langley
Institute for the Study of Learning and Expertise
Palo Alto, California, USA

This research was supported by ONR Grant N00014-15-1-2517. Thanks to Chris Pearce, Yu Bai, Charlotte Worsfold, Ben Meadows, Mike Barley, and Ed Katz for their contributions to the project.

Background and Motivation

Cognitive architectures specify *invariant* features of the mind, but people can solve the same task in many *different* ways.

This creates a tension between the desire to identify universals and to explain observed variations.

We would like a theory of the *human problem solving* that:

- Remains generally consistent with the standard accounts of problem solving, which explain many phenomena;
- Explains the great variety of strategies observed not only in humans and but also in machines.

In this talk, we describe a cognitive architecture that embodies such a theory, including an account of solution *execution*.

The Standard Theory

The standard theory of problem solving (Newell & Simon, 1961) makes a number of claims:

- Problem solving involves the mental representation, interpretation, and manipulation of *symbol structures*.
- This process involves *search* through a space of candidates that it encodes as such symbol structures.
- Search is not exhaustive but rather guided by *heuristics* that make it selective and tractable.
- Problem solvers use *means-ends analysis* to decompose complex problems into simpler ones.

Repeated studies have been consistent with most aspects of this theory, but *not the final one*; people are far more variable.

Means-Ends Analysis

As embodied in Newell and Simon’s *General Problem Solver*, means-ends analysis relies on four basic ideas:

- Problem solving interleaves *transforming* the current state into a desired one with *applying* an operator to do the transformation.
- Operators are considered only if they *reduce differences* between the current and desired state.
- Problem solving involves search through a space of alternative problem decompositions.
- The selected operator determines the structure of the resulting problem decomposition.

Only the *second* assumption is limiting and problematic. The other three tenets may well be worth retaining.

A Revised Theory of Problem Solving

We propose a revised theory of problem solving that replaces means-ends analysis with three postulates:

- Problem solving involves recursively dividing problems into subproblems, with solutions stated as *decomposition trees*.
- Search details – operator generation / evaluation, node selection, success / failure criteria – are controlled by *strategic parameters*.
- Domain expertise is often encoded as *generalized decompositions* for breaking a problem into subproblems (as in HTNs).

This revision retains the key ideas of means-ends analysis without committing to chaining off goals.

We have embedded these assumptions in HPS, an architecture for *h*ierarchical *p*roblem *s*olving.

Encoding Problem Decompositions

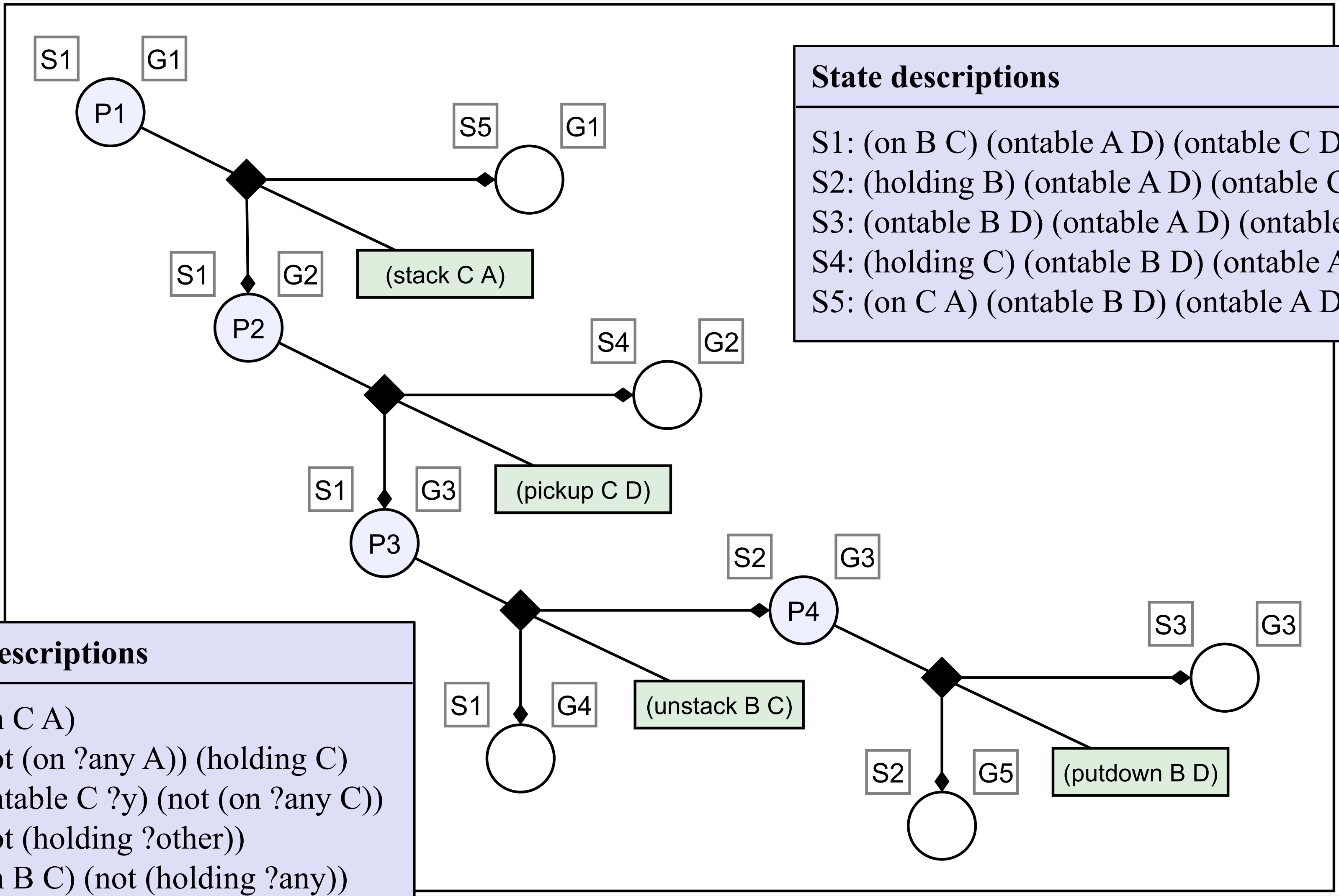
The HPS architecture encodes problem solutions – both partial and complete – as decomposition trees.

Each element in such an AND tree has two components:

- A *problem*, which includes a *state* and a *goal* description;
- A *decomposition*, which specifies an operator instance that breaks the problem into:
  - A *down* subproblem, which must be solved before applying the operator instance;
  - A *right* subproblem, which must be solved afterward to achieve the parent's goals.

Each operator instance has application conditions, expected results, and constraints on shared variables.

Encoding Problem Decompositions



State descriptions

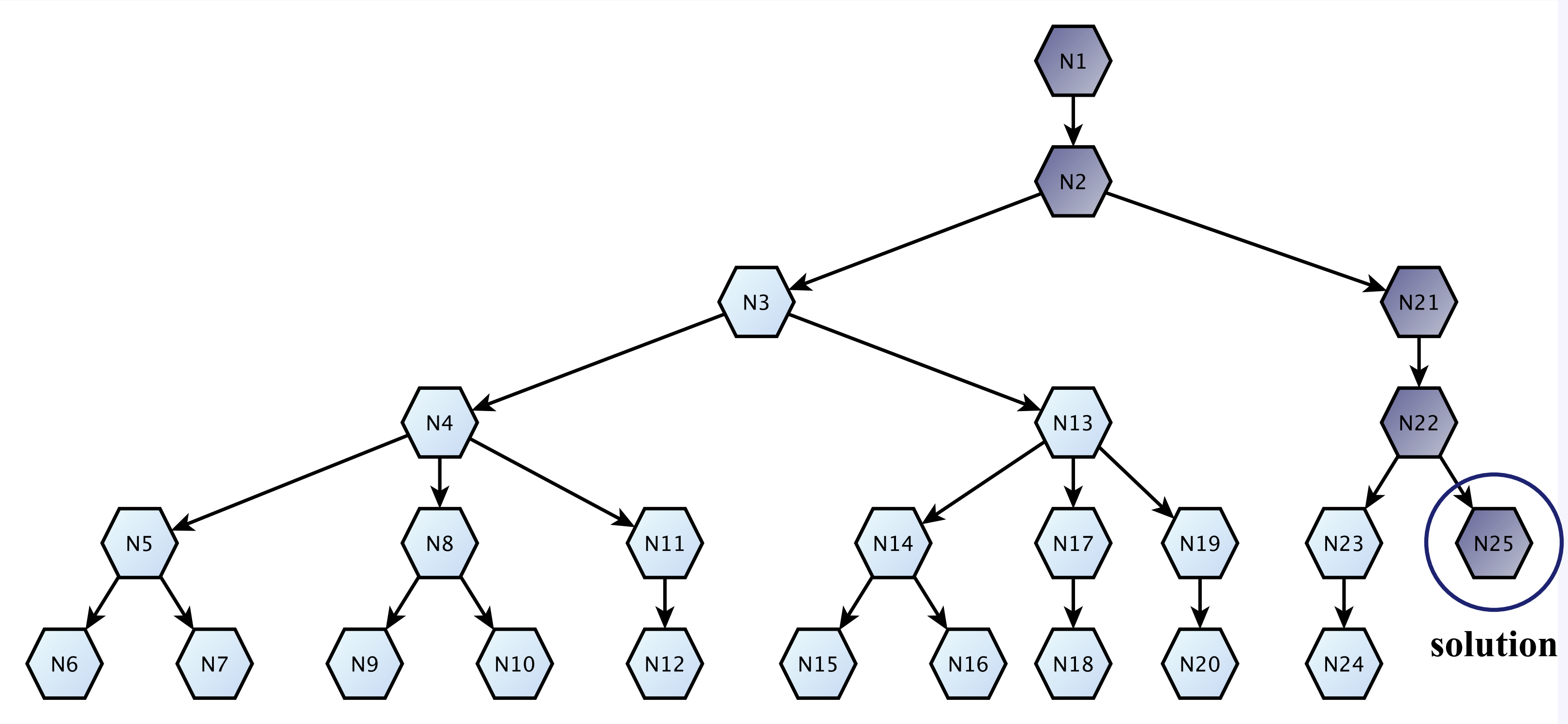
S1: (on B C) (ontable A D) (ontable C D)
S2: (holding B) (ontable A D) (ontable C D)
S3: (ontable B D) (ontable A D) (ontable C D)
S4: (holding C) (ontable B D) (ontable A D)
S5: (on C A) (ontable B D) (ontable A D)

Goal descriptions

G1: (on C A)
G2: (not (on ?any A)) (holding C)
G3: (ontable C ?y) (not (on ?any C))
(not (holding ?other))
G4: (on B C) (not (holding ?any))
G5: (holding B)

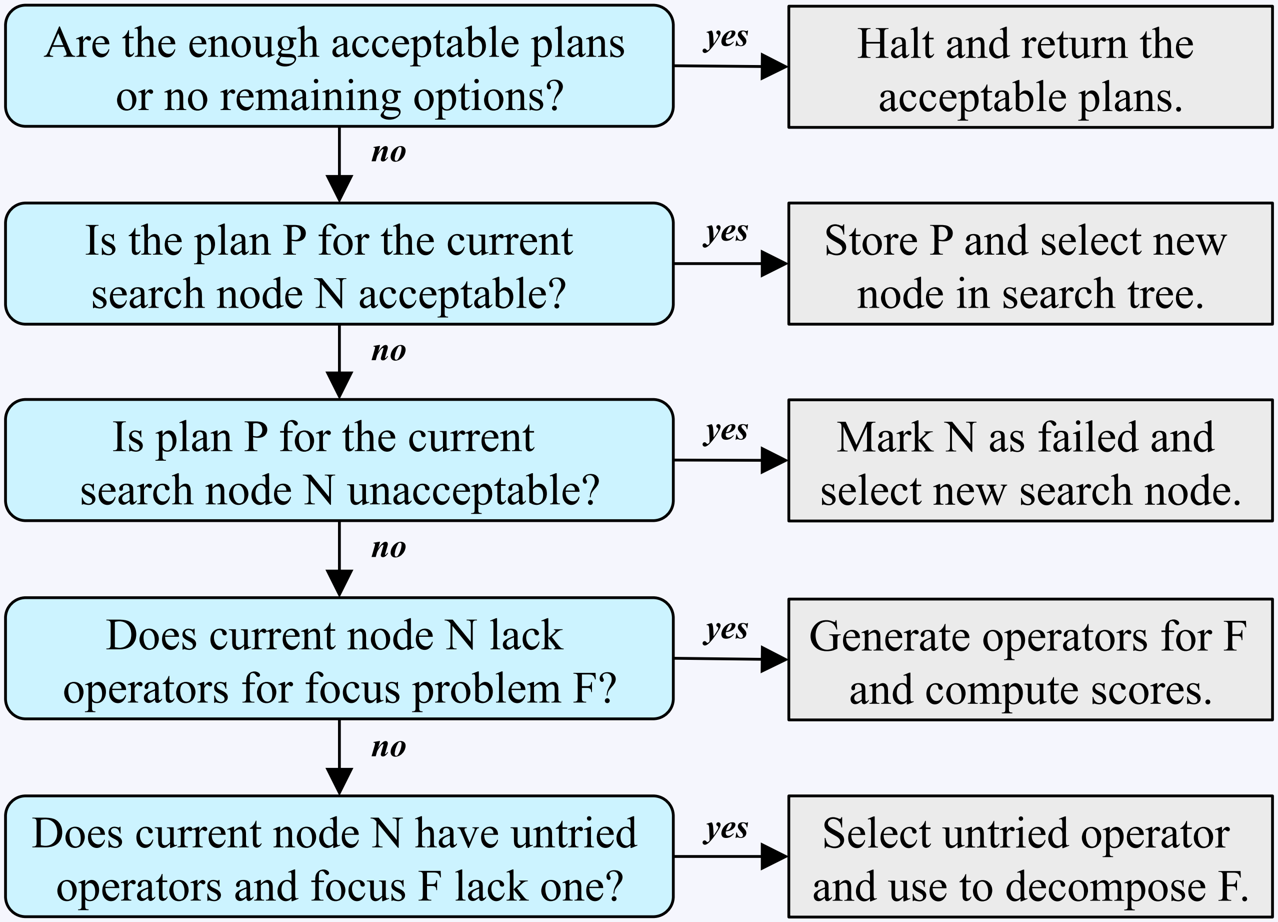
Organizing Nodes into a Search Tree

Here is a search tree HPS generates during its problem solving, with nodes along a successful path shaded.



Each node (partial plan) elaborates its parent by adding a new subproblem decomposition. Numbers reflect generation order.

The HPS Problem-Solving Cycle



Are the enough acceptable plans or no remaining options?

Halt and return the acceptable plans.

Is the plan P for the current search node N acceptable?

Store P and select new node in search tree.

Is plan P for the current search node N unacceptable?

Mark N as failed and select new search node.

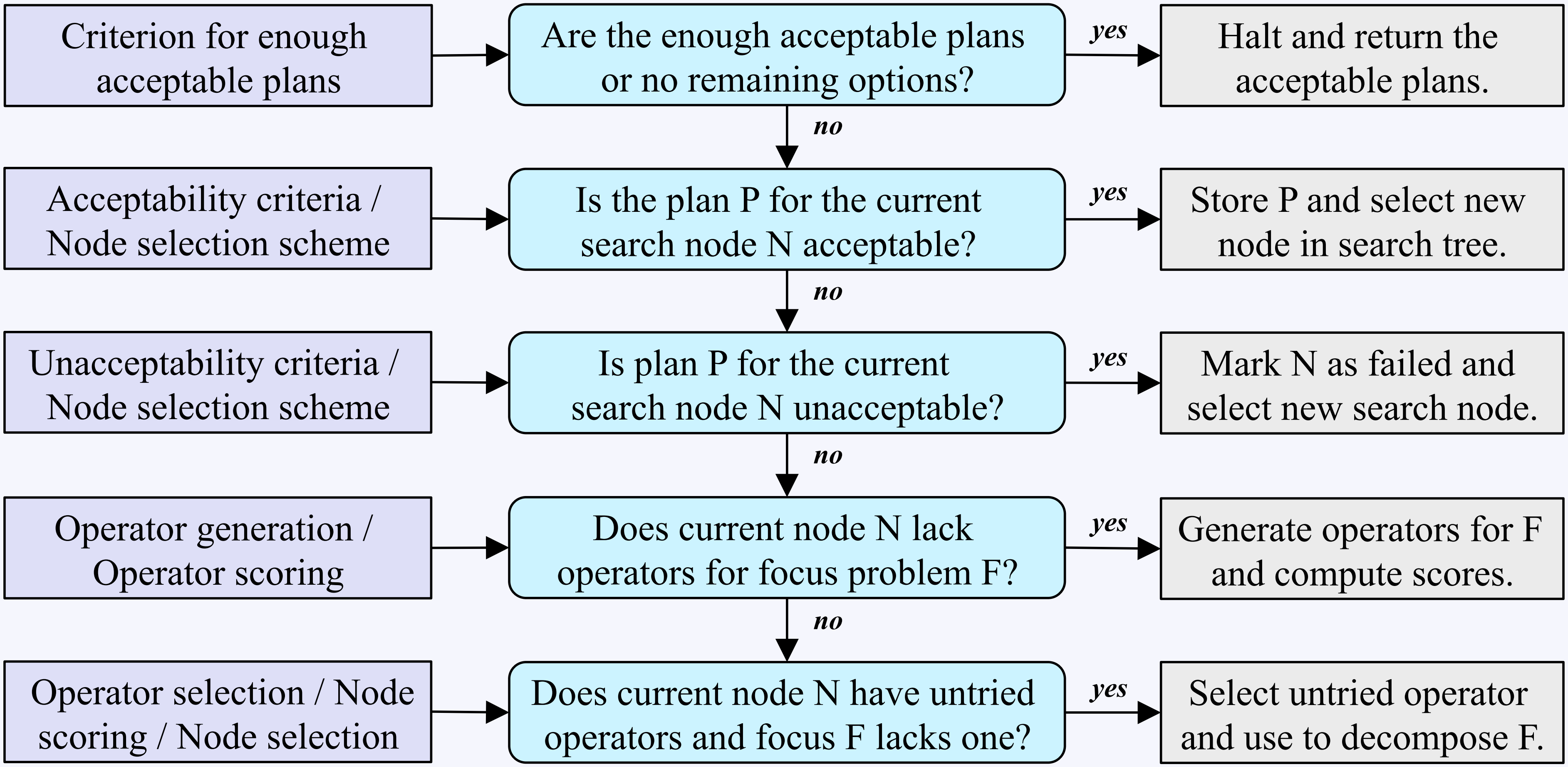
Does current node N lack operators for focus problem F?

Generate operators for F and compute scores.

Does current node N have untried operators and focus F lack one?

Select untried operator and use to decompose F.

The HPS Problem-Solving Cycle



Criterion for enough acceptable plans

Are the enough acceptable plans or no remaining options?

Halt and return the acceptable plans.

Acceptability criteria / Node selection scheme

Is the plan P for the current search node N acceptable?

Store P and select new node in search tree.

Unacceptability criteria / Node selection scheme

Is plan P for the current search node N unacceptable?

Mark N as failed and select new search node.

Operator generation / Operator scoring

Does current node N lack operators for focus problem F?

Generate operators for F and compute scores.

Operator selection / Node scoring / Node selection

Does current node N have untried operators and focus F lacks one?

Select untried operator and use to decompose F.

Strategic Parameters

Parameter settings are intrinsically *composable*, so different combinations can reproduce many distinct strategies.

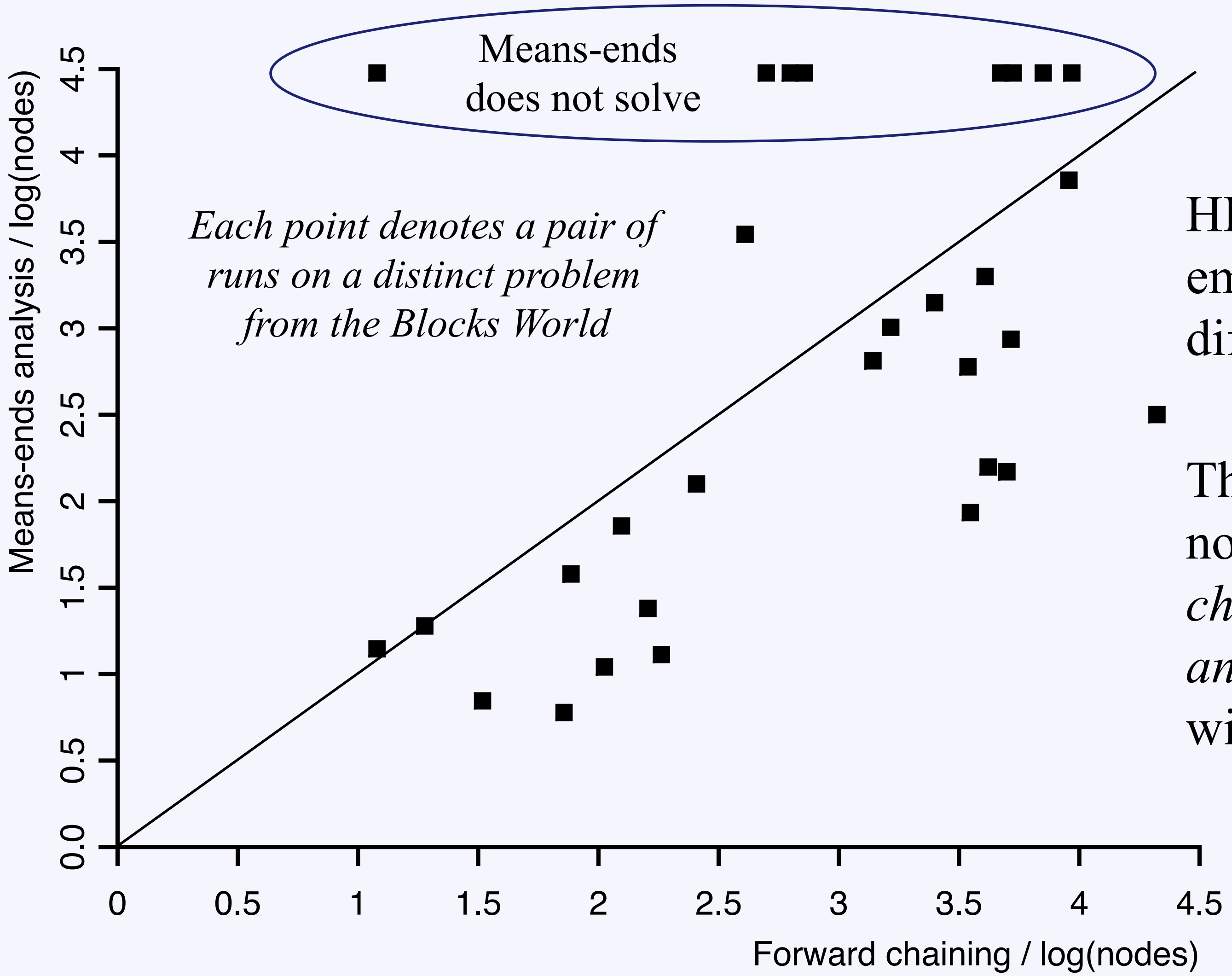
Initial Tests of HPS Architecture

We have demonstrated HPS's ability to solve novel problems on two familiar domains:

- *Blocks World* – finding plans that produce desired configurations:
  - Solutions to problems ranged from four to 12 steps.
  - Depth-first forward chaining solved 30 out of 30 tasks.
  - Depth-first means ends solved only 23 of these problems.
- *Kinship inference* – deducing complex relations from simple ones:
  - Problems required from one to eight inference steps.
  - Forward chaining could not solve more complex tasks.
  - Goal-driven means-ends analysis had little difficulty.

These basic results show that strategy effectiveness interacts with domain characteristics.

Forward Chaining vs. Means-Ends Analysis

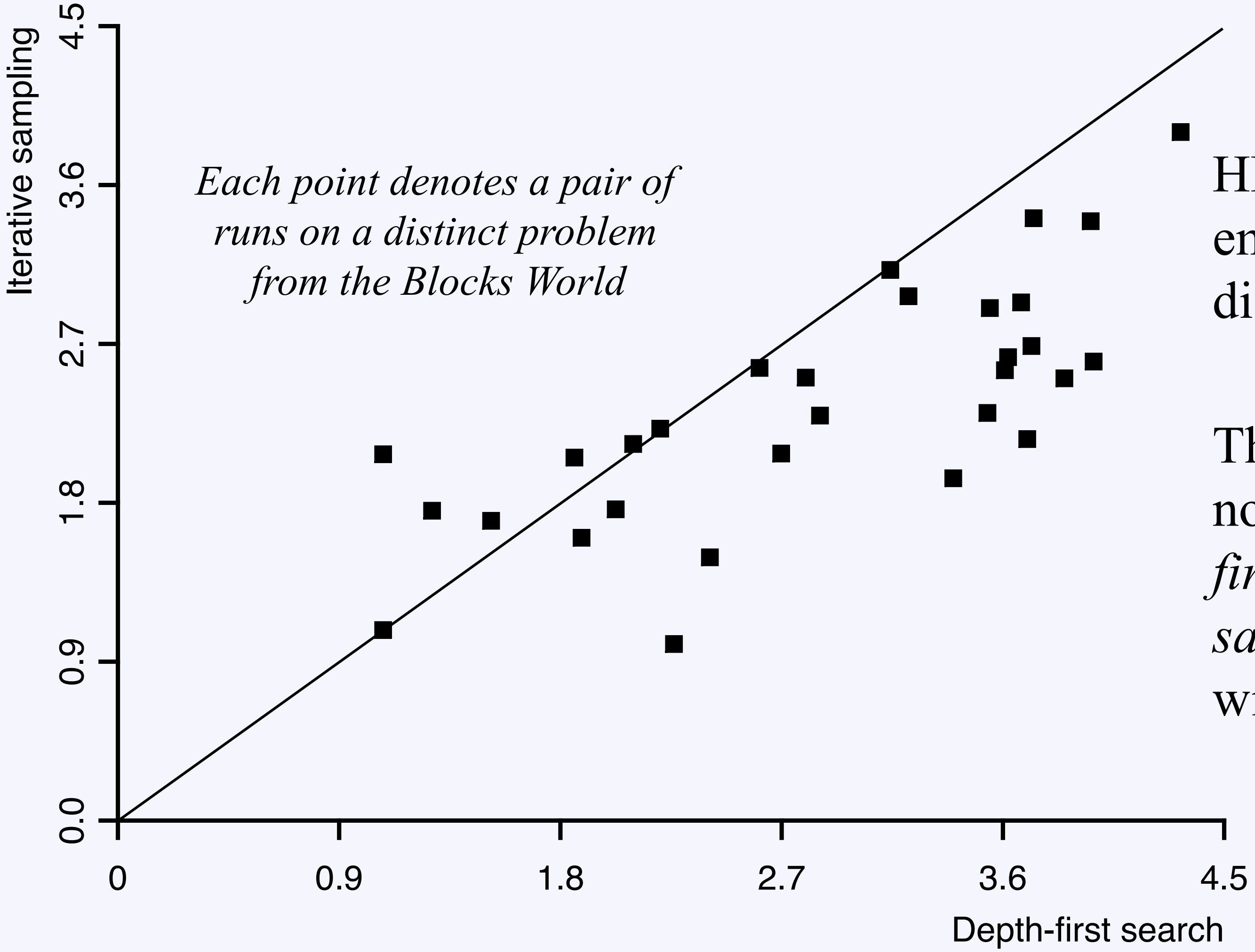


Each point denotes a pair of runs on a distinct problem from the Blocks World

HPS parameters support empirical comparison of different strategies.

This study compares the nodes visited by *forward chaining* and *means-ends analysis* when combined with depth-first search.

Depth-First Search vs. Iterative Sampling



Each point denotes a pair of runs on a distinct problem from the Blocks World

HPS parameters support empirical comparison of different strategies.

This study compares the nodes visited by *depth-first search* and *iterative sampling* when combined with forward chaining.

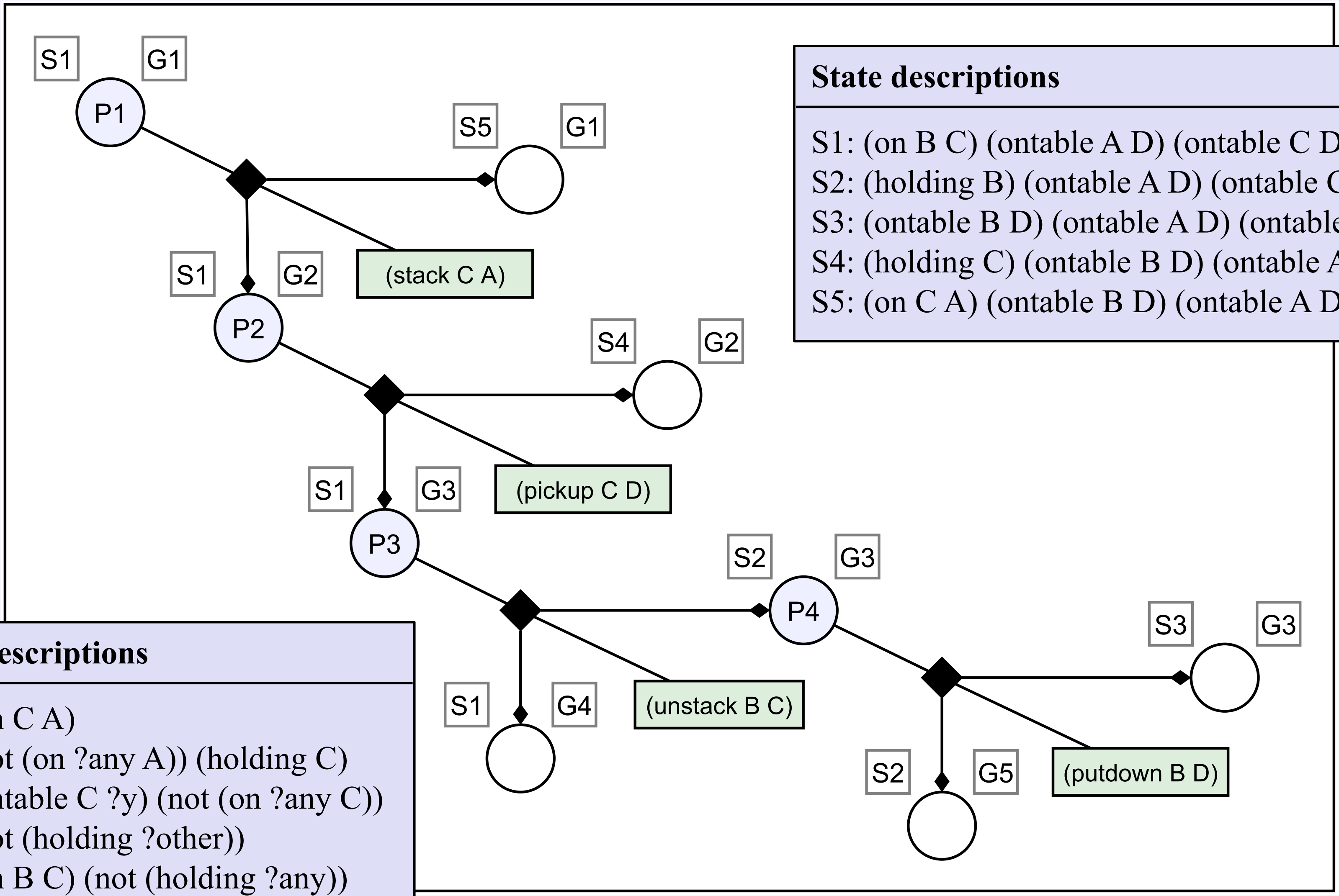
An Account of Plan Execution

Humans also *execute* their plans to achieve goals. Our account of plan execution makes four claims:

- Plans (and skills) are stored as *hierarchical decompositions* like those produced by problem solving.
- Plan execution *traverses these decomposition trees*, top to bottom and left to right, with physical actions at terminal nodes.
- This process relies on a cognitive cycle with steps for *sensing* to check operator conditions and effects.
- *Strategic parameters* govern decisions on each cycle to produce different behaviors (e.g., closed vs. open loop control).

We have embedded these assumptions in an HPS module for hierarchical plan execution.

Executing a Hierarchical Plan



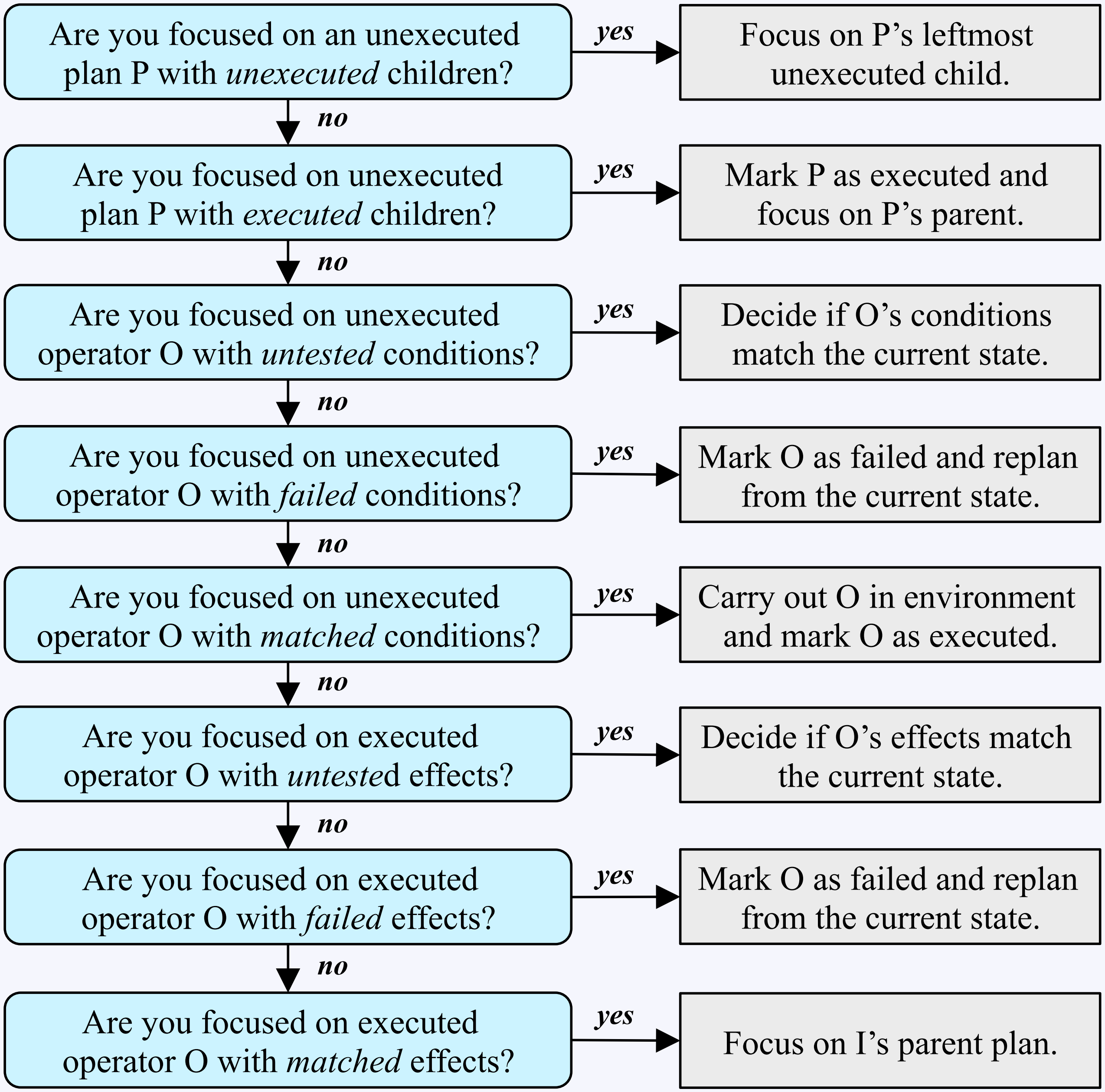
State descriptions

S1: (on B C) (ontable A D) (ontable C D)
S2: (holding B) (ontable A D) (ontable C D)
S3: (ontable B D) (ontable A D) (ontable C D)
S4: (holding C) (ontable B D) (ontable A D)
S5: (on C A) (ontable B D) (ontable A D)

Goal descriptions

G1: (on C A)
G2: (not (on ?any A)) (holding C)
G3: (ontable C ?y) (not (on ?any C))
(not (holding ?other))
G4: (on B C) (not (holding ?any))
G5: (holding B)

The HPS Execution Cycle



Are you focused on an unexecuted plan P with *unexecuted* children?

yes Focus on P’s leftmost unexecuted child.

no

Are you focused on unexecuted plan P with *executed* children?

yes Mark P as executed and focus on P’s parent.

no

Are you focused on unexecuted operator O with *untested* conditions?

yes Decide if O’s conditions match the current state.

no

Are you focused on unexecuted operator O with *failed* conditions?

yes Mark O as failed and replan from the current state.

no

Are you focused on unexecuted operator O with *matched* conditions?

yes Carry out O in environment and mark O as executed.

no

Are you focused on executed operator O with *untested* effects?

yes Decide if O’s effects match the current state.

no

Are you focused on executed operator O with *failed* effects?

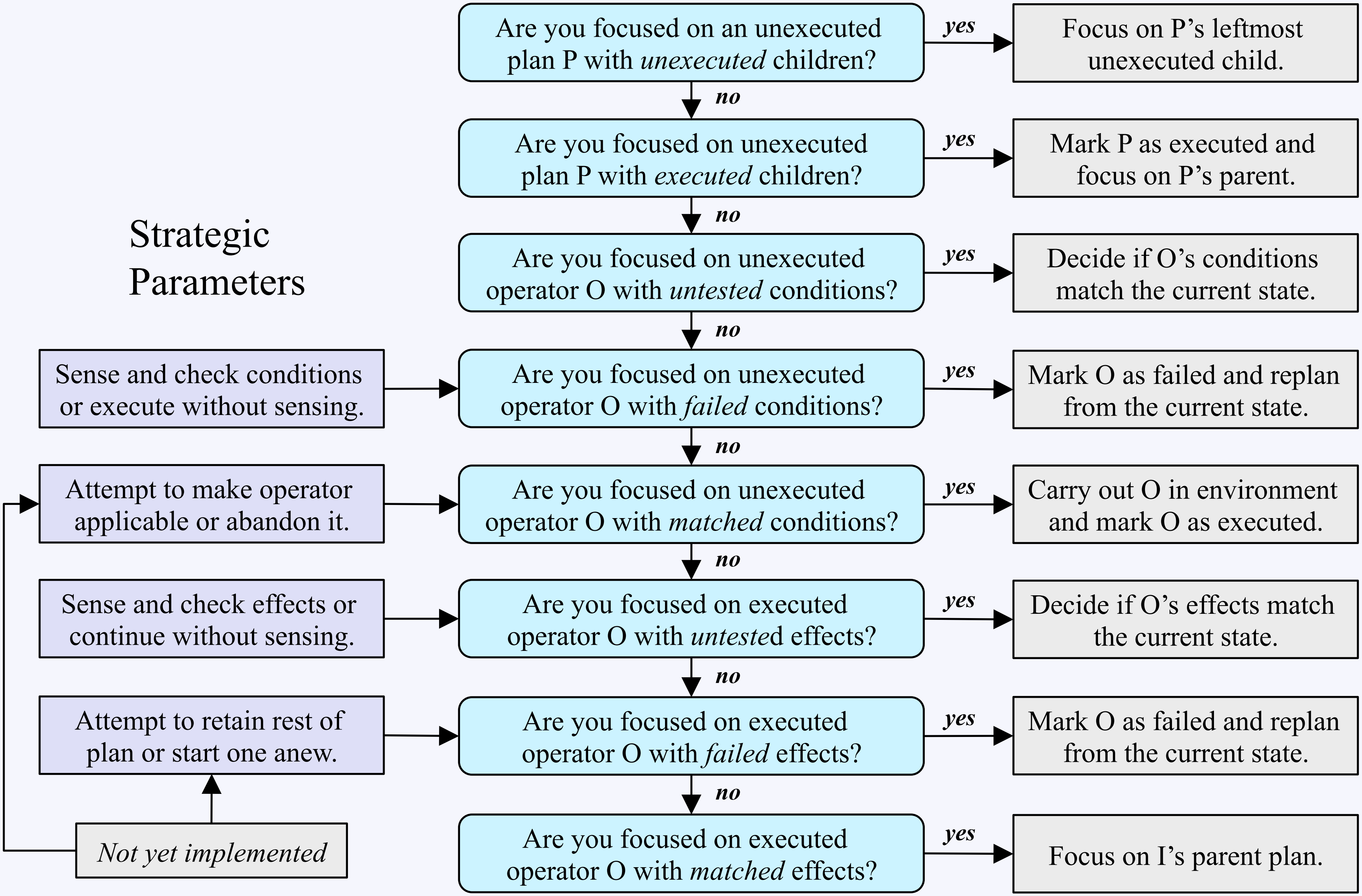
yes Mark O as failed and replan from the current state.

no

Are you focused on executed operator O with *matched* effects?

yes Focus on I’s parent plan.

The HPS Execution Cycle



Are you focused on an unexecuted plan P with *unexecuted* children?

yes → Focus on P's leftmost unexecuted child.

no

Are you focused on unexecuted plan P with *executed* children?

yes → Mark P as executed and focus on P's parent.

no

Are you focused on unexecuted operator O with *untested* conditions?

yes → Decide if O's conditions match the current state.

no

Are you focused on unexecuted operator O with *failed* conditions?

yes → Mark O as failed and replan from the current state.

no

Are you focused on unexecuted operator O with *matched* conditions?

yes → Carry out O in environment and mark O as executed.

no

Are you focused on executed operator O with *untested* effects?

yes → Decide if O's effects match the current state.

no

Are you focused on executed operator O with *failed* effects?

yes → Mark O as failed and replan from the current state.

no

Are you focused on executed operator O with *matched* effects?

yes → Focus on I's parent plan.

Strategic Parameters

Sense and check conditions or execute without sensing.

Attempt to make operator applicable or abandon it.

Sense and check effects or continue without sensing.

Attempt to retain rest of plan or start one anew.

Not yet implemented

Interleaving Planning and Execution

We are extending the architecture to *interleave* problem solving and execution with parameters that state:

- When to stop elaborating a plan and start executing it:
  - E.g., after plan complete, after solving subproblem, after one step.
- Which elements of plan to retain when it fails during execution:
  - E.g., attempt to make operator applicable or abandon it.

We hope to mimic strategies from the literature, including:

- Open-loop execution after finding a complete plan;
- Closed-loop (reactive) control with one-step lookahead;
- N-step lookahead alternating with one action (game playing).

We expect each strategy's effectiveness to interact with domain characteristics in both HPS and in humans.

Strategy Variation and Adaptation

HPS offers an architecture-level account of strategy variations, but it does not specify when they occur.

We hypothesize that they arise in response to task demands, and that one can model them by:

- Collecting meta-level data about local problem features:
  - E.g., relative branching factors, reliability of operators
- Making parameter settings conditional on these statistics:
  - E.g., selecting direction of search, frequency of sensing

The result with be an *adaptive* architecture that explains both *how* strategy variations occur and *when* people use them.

Relation to Earlier Research

Our framework draws ideas from many earlier research efforts, including:

- Problem solving as search for decompositions (Newell et al., 1961)
- Planning as incremental refinement (Kambhampati et al., 1995)
- Strategic rules support variation in problem solving (Laird, 1984)
- Execution as test-operate-test processing (Miller et al., 1959)
- Continuum of open-loop to closed-loop control (Iba & Langley, 1987)
- Meta-level cognitive control by inspecting mental traces (Cox, 2007)

HPS incorporates these ideas but also *combines* them to offer an extended theory of human problem solving.

Summary Remarks

In this talk, we presented an architectural account of variation in problem solving and execution that:

- Retains key ideas from the *standard theory* of problem solving
- Drops means-ends analysis but retains *hierarchical decomposition*
- Adds execution as incremental *traversal over hierarchical plans*
- Introduces *strategic parameters* that together determine behavior
- Supports strategies for *interleaving* planning and execution
- Suggests that strategy variation is an *adaptation* to task demands

This theory, and its implementation in HPS, go beyond other cognitive architectures in breadth and coverage.

End of Presentation