Architectural Goal Maintenance: Operand2

Bob Wray, John Laird, Ron Chong, Doug Pearson and Randy Jones
robert.wray@umich.edu

Soar XVII
28 June 1997

Outline

1. Motivation
   - Non-contemporaneous constraints
   - Race conditions/knowledge contention

   Problems exacerbated by interaction with an external environment
2. Solutions: New functions for goal maintenance
3. New decision cycle
4. Expected impacts
   - Knowledge development
   - Cost of solution algorithms
   - Impact on performance
5. Future work/Conclusions

Motivation: Existing Problems

A. Non-contemporaneous constraints in chunked rules (NCC)

A rule containing conditions that specify features that never occur at the same time

1. Persistent features
2. Persistent selections
3. “Elaboration” persistence

B. Race conditions between rule firings

A production result that depends on the number of production firings before the production fires (in addition to content)

1. Application and elaboration/elaboration persistence
2. Problem solving after impasse resolution
3. Knowledge contention

Are These Problems Significant?

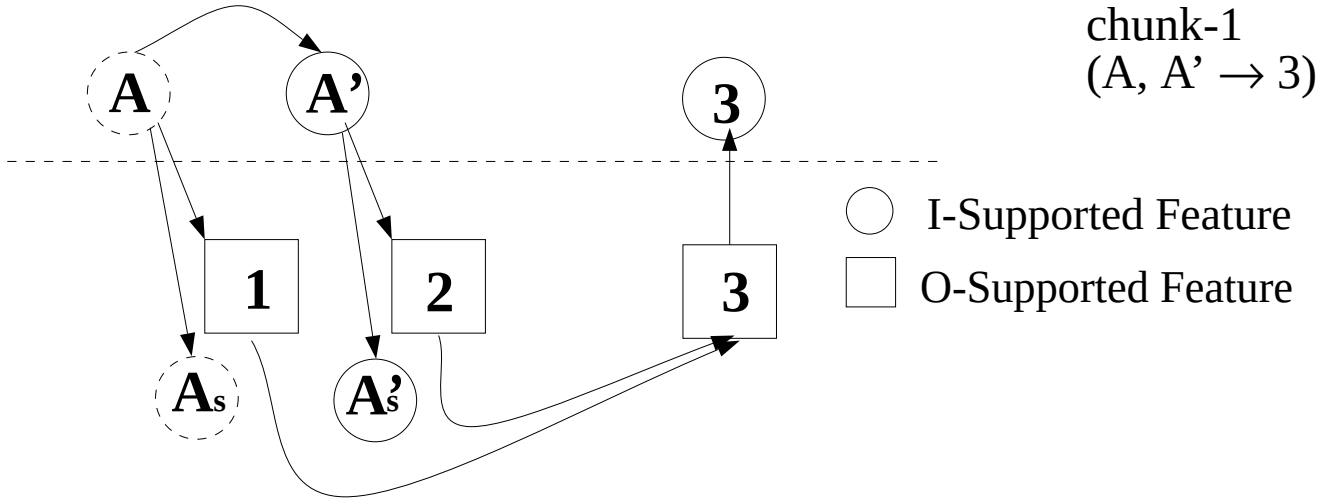
Non-contemporaneous constraints

1. Missing opportunity to learn something useful
   - Wasted processing (invoking the chunker for a useless rule)
2. “Write code with learning in mind”
   - Novice Soar users have difficulty getting their systems to chunk correctly
3. (Significantly?) increased frequency in external environment
   - Some features are changing independent of the rule knowledge

Race conditions

1. Behavior not consistent with PSCM/theory
   - “Implementation shows through”
2. Additional design/build time for agent development
3. Increased frequency in external environment
   - Activity throughout the stack with changing input

NCCs from Persistent WMEs



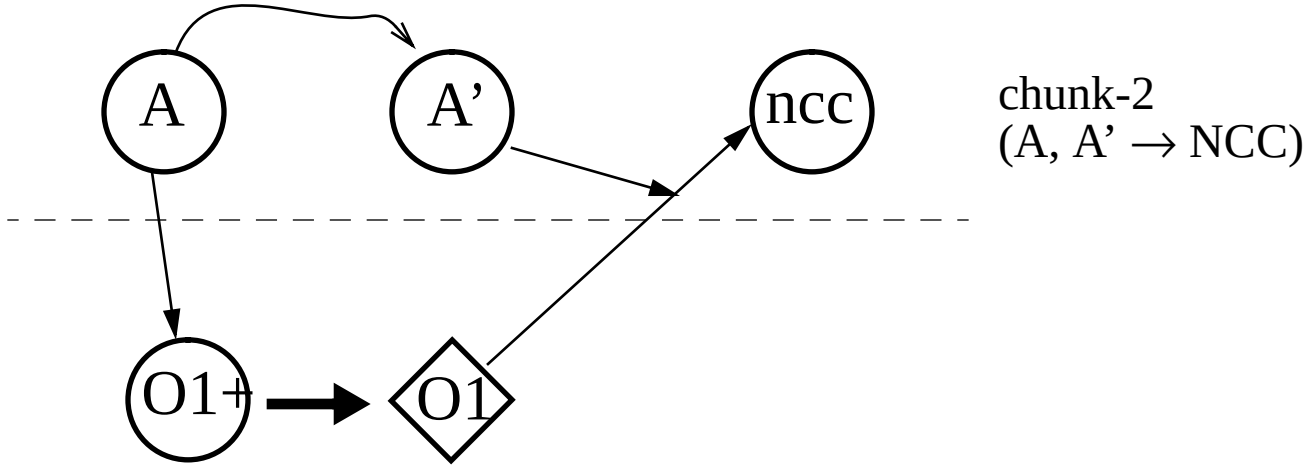
O-support: structure is maintained after instantiating conditions change

I-support: structure is retracted when instantiating conditions change

Persistent features lead to non-contemporaneous constraints when:

- instantiating features of the persistent feature change
- other, non-contemporaneous features are used in the subgoal processing
- results include both the persistent feature instantiation and subsequent features

NCCs from Persistent Decisions

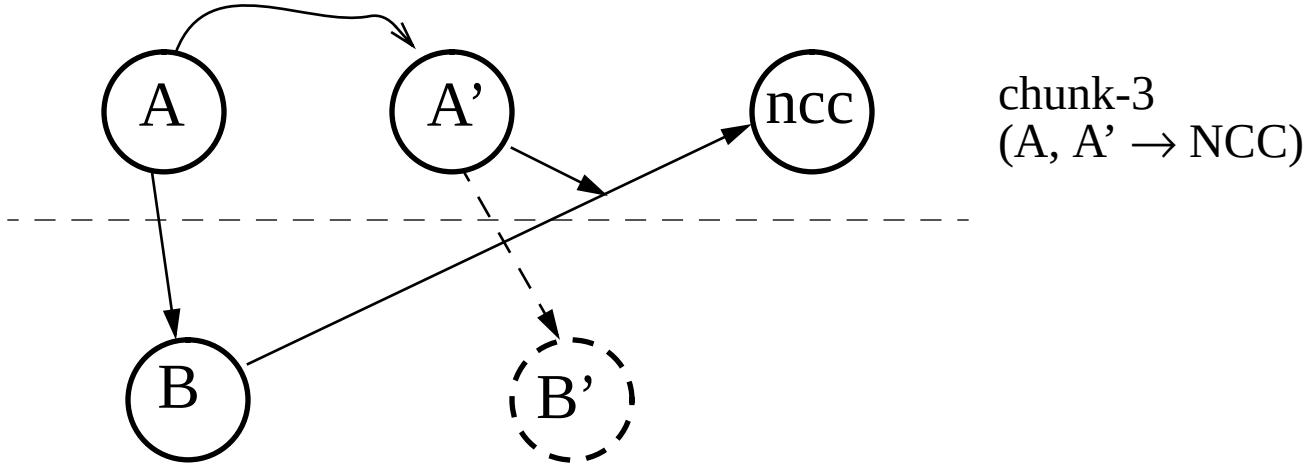


"C-support" allows operator selection to persist after proposal is lost

C-support leads to non-contemporaneous constraints when:

- An operator proposal no longer matches
- The operator creates a result (perhaps indirectly)
- The result also depends on some feature not contemporaneous with the original proposal

NCCs from Elaboration Persistence



Soar implementation: "lazy" retraction

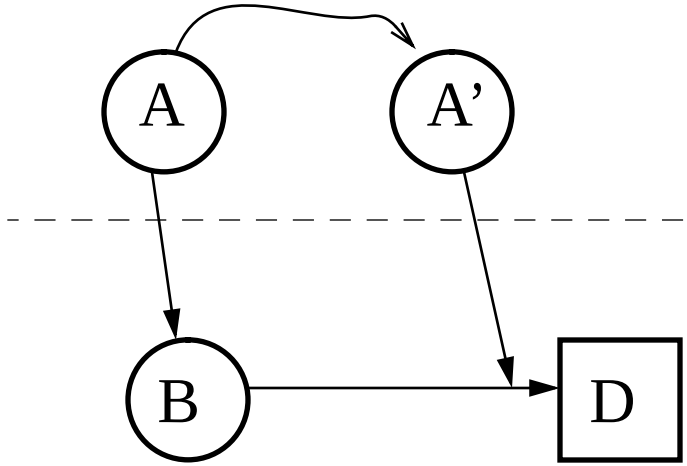
- Can result in cascade of retractions
- Elaboration persistence:

  I-supported features persist during the elaboration cycle in which their instantiation retracts.

Elaboration persistence leads to non-contemporaneous constraints when:

- Result is created that tests an i-supported item that will not be present at quiescence
- Result instantiation also tests a feature non-contemporaneous with the disappearing feature

Elaboration Persistence Race Conditions



(A → B)
(B, A' → D)

At Quiescence:
A', D (not B)

A' and B really should not appear in WM simultaneously

However, they exist simultaneously in the elaboration cycle in which B's instantiation retracts

I-supported instantiation matching against retracting WME:

- No problem (feature retracts with instantiation)

O-supported instantiation matching against retracting WME:

- May create feature based on feature(s) not present at quiescence

Problem Solving after Impasse Resolution

Problem solving continues until ‘quiescence’ regardless of whether or not the goal/impasse in which productions are firing has already been resolved.

Race Condition:

- Current preferences indicate a unique selection
- Current decision indicates an impasse (non-unique selection)
- Knowledge dependent upon one or the other condition can be retrieved

Potential for non-contemporaneous constraints

Wasted elaboration cycles

- Still retrieving knowledge when the impasse has been resolved

Knowledge Contention

```
S1:
O1: do-task
  S2: (operator no-change)
  O2: determine-what-to-do
    S3: (operator no-change)
    O3: do-subtask-4
```

chunk-1
(O1, I1 == 3 → send-output <x>)

do-subtask-4*apply*send-output
(O3, I1 == 3 → send-output <x>)

Knowledge applied simultaneously can "contend" for the same resource (e.g., output-link)

- create identical values: no problem
- create new identifiers: attribute tie
  - even if identifiers have identical structure

Architecture:

- No way of recognizing 'identical' pieces of knowledge or preferring one over another

Previous Solutions

Number of attempts to solve NCC

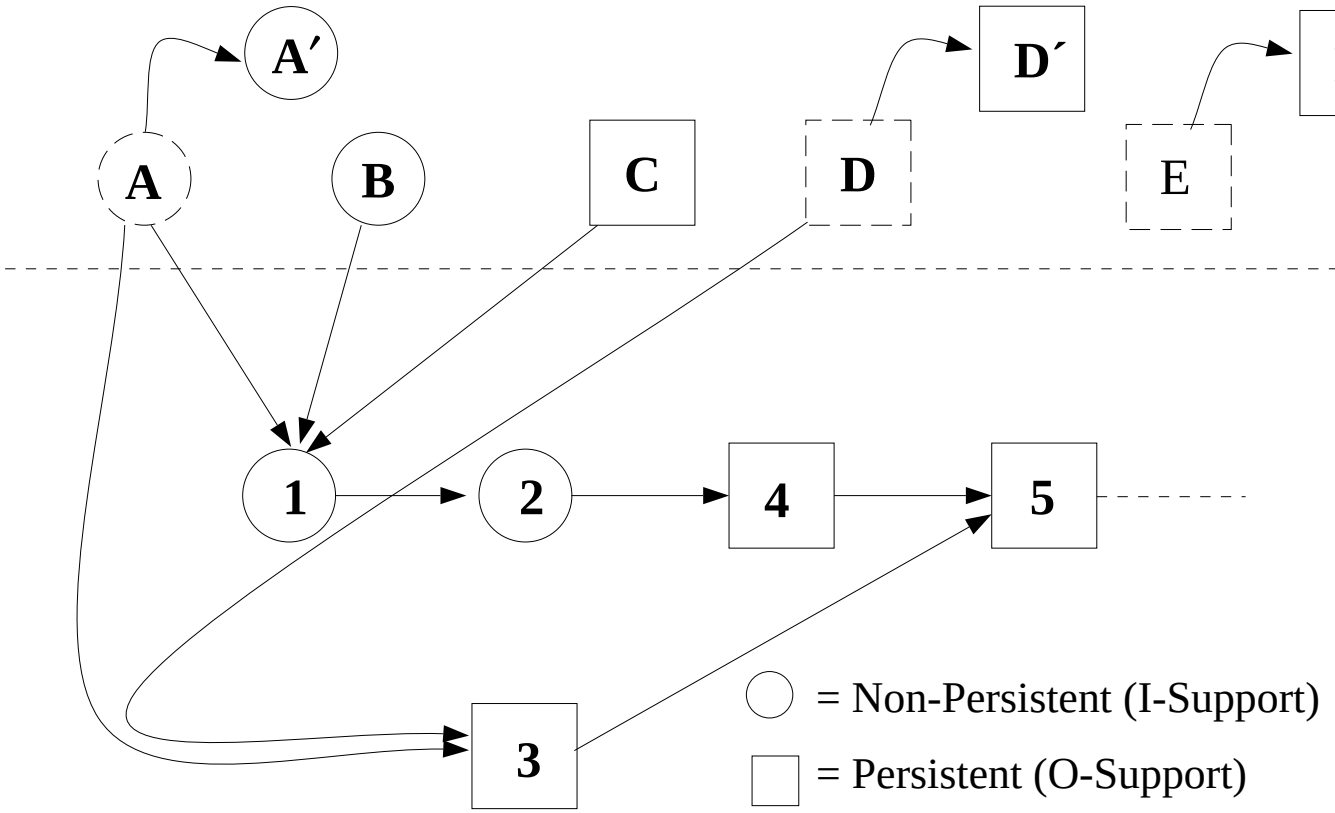
- Architectural solutions
  - S-support
  - OPERAND
  - ....
- Solutions using programming conventions
  - Neo-PEACTIDM
  - Cambridge
  - ....

  Solutions broader than just NCC

No(?) attempts to solve race conditions architecturally

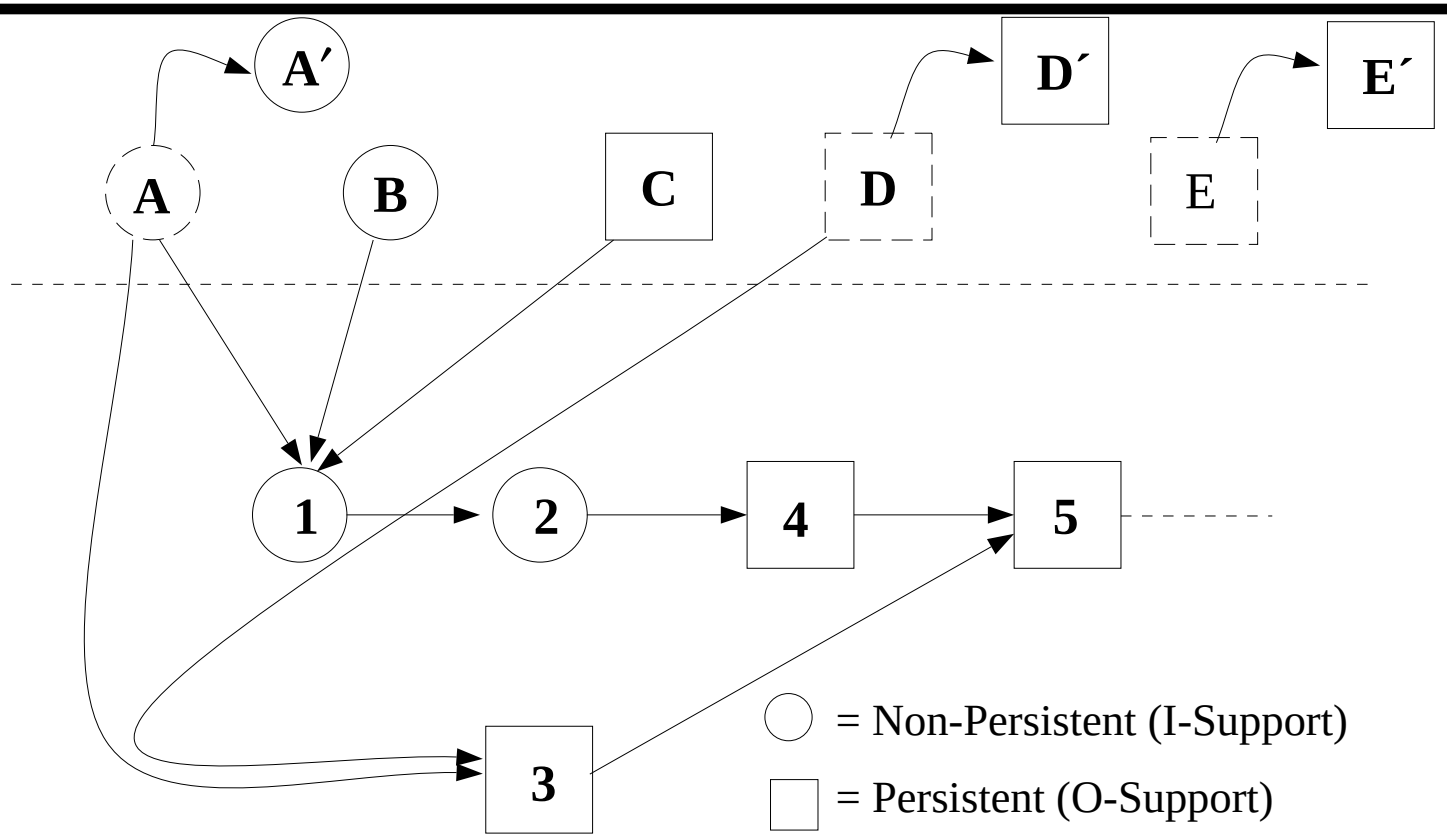
- A few attempts to document them (R. Jones)

Addressing NCC: S-Support



- D to D′ transition: Remove 3 then 5
- A to A′ transition: Remove 3, 4 and 5
- E to E′ transition: No effect
- Subgoal persists across all transitions

Addressing NCC: OPERAND



- Creation of 3, 4, and 5: Redecision of operator slot in subgoal
- D to D´ /A to A´ transition: Redecision at supergoal (subgoal removed)
- E to E´ transition: Also redecide operator slot at supergoal
- Subgoal (possibly) regenerated after supergoal persistent changes

Comparison of S-Support and OPERAND

S-Support

- Feature-centered truth maintenance
- + Conservative WME removal (minimal increase in decisions)
- - Computationally expensive
- - Potentially more race conditions

OPERAND

- Goal/Slot-centered monitoring of persistent effects
- + Computationally cheap
- + Requires architectural recognition of persistent effects
  - Changed o-support calculation
- + Eliminated need for operator terminations
- + Race Conditions:
  - Separated operator application (PE) from elaboration (IE)
  - Delayed chunking until quiescence
- - Aggressive response to persistent changes
  - Large increase in pre-chunk decisions

Operand2: Overview

New goal maintenance functions:

1. Goal-oriented truth maintenance
2. Separation of goal elaboration and operator application
3. Selection consistency checks
4. Goal-limited knowledge retrieval

+ Removes sources of non-contemporaneous constraints
+ Solves some race conditions
+ Adds constraint for knowledge design
- Additional computational cost of new functions
- Performance costs (increased decisions, elaborations)
- Another change to the architecture

1. Goal-Oriented Truth Maintenance

Associated with each goal is a “goal dependency set” (GDS)

- Creation of o-supported WMEs adds instantiating, supergoal features to the dependency set
- When a WME on a goal’s dependency set is removed, the goal is removed as well

Result:

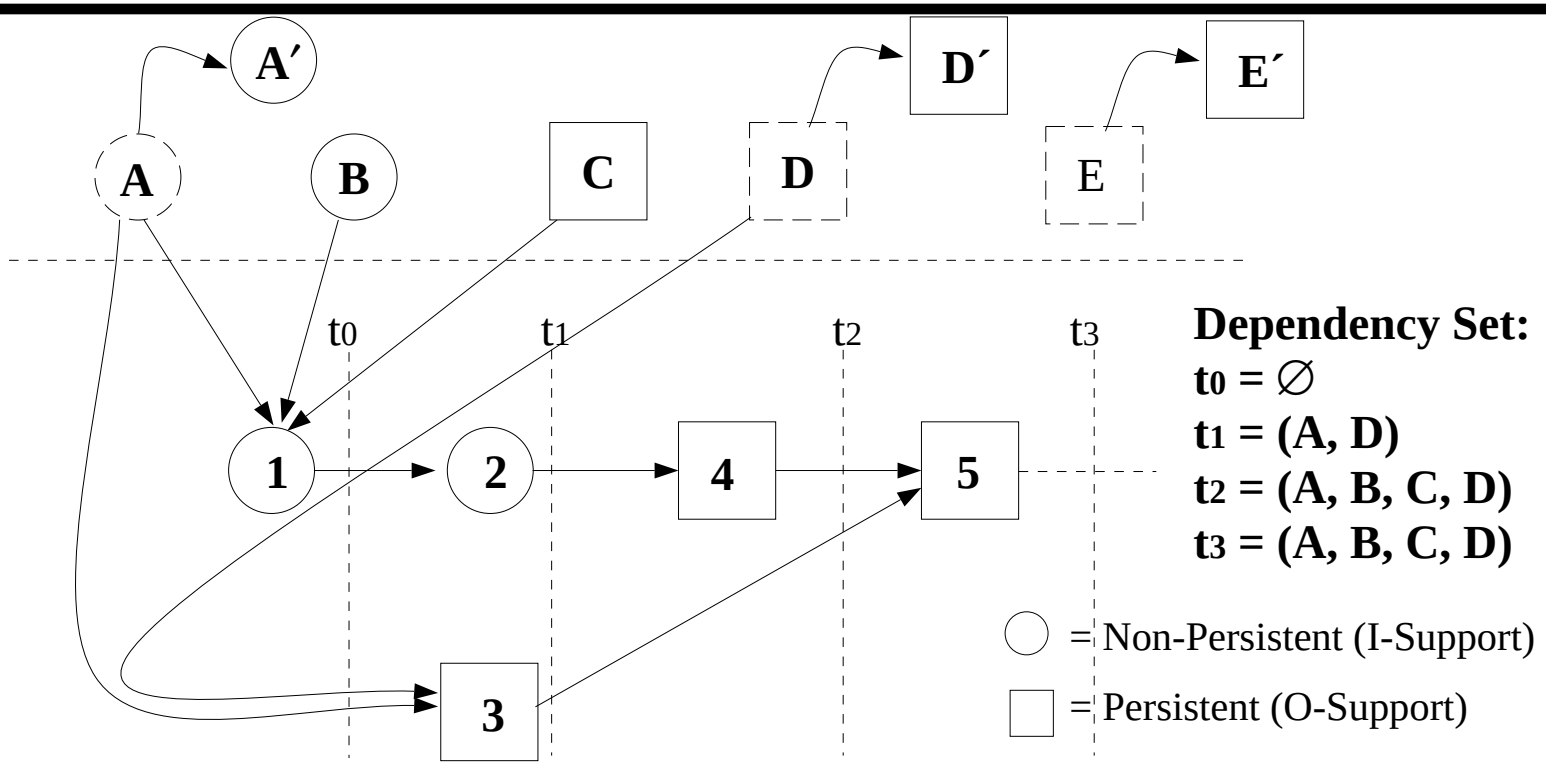
- Elimination of NCC due to persistent WMEs
- Altered strength of persistence of o-supported WMEs

  O-supported WMEs persist only as long as the WMEs instantiating supergoal features remain unchanged

Cost:

- Memory: new data structures for GDS
- Algorithm: Backtrace-like mechanism for each o-supported WME addition

Goal-Oriented Truth Maintenance



- Dependency Set maintained for o-supported features only
  - I-supported features have “built-in” dependency maintenance
- D to D´/A to A´ transitions: retraction of the subgoal
- E to E´: no retraction (not in the dependency set)

2. Separation of Elaboration and Application

Application ‘Superphase:’ (PE)

- Only o-supported assertions may fire
- Always one elaboration cycle per PE

Elaboration ‘Superphase:’ (IE)

- I-supported productions match, fire and retract
- O-supported instantiations can retract
- Multiple IE elaboration cycles per IE
- IE continues until “minor quiescence:” all i-instantiations fired

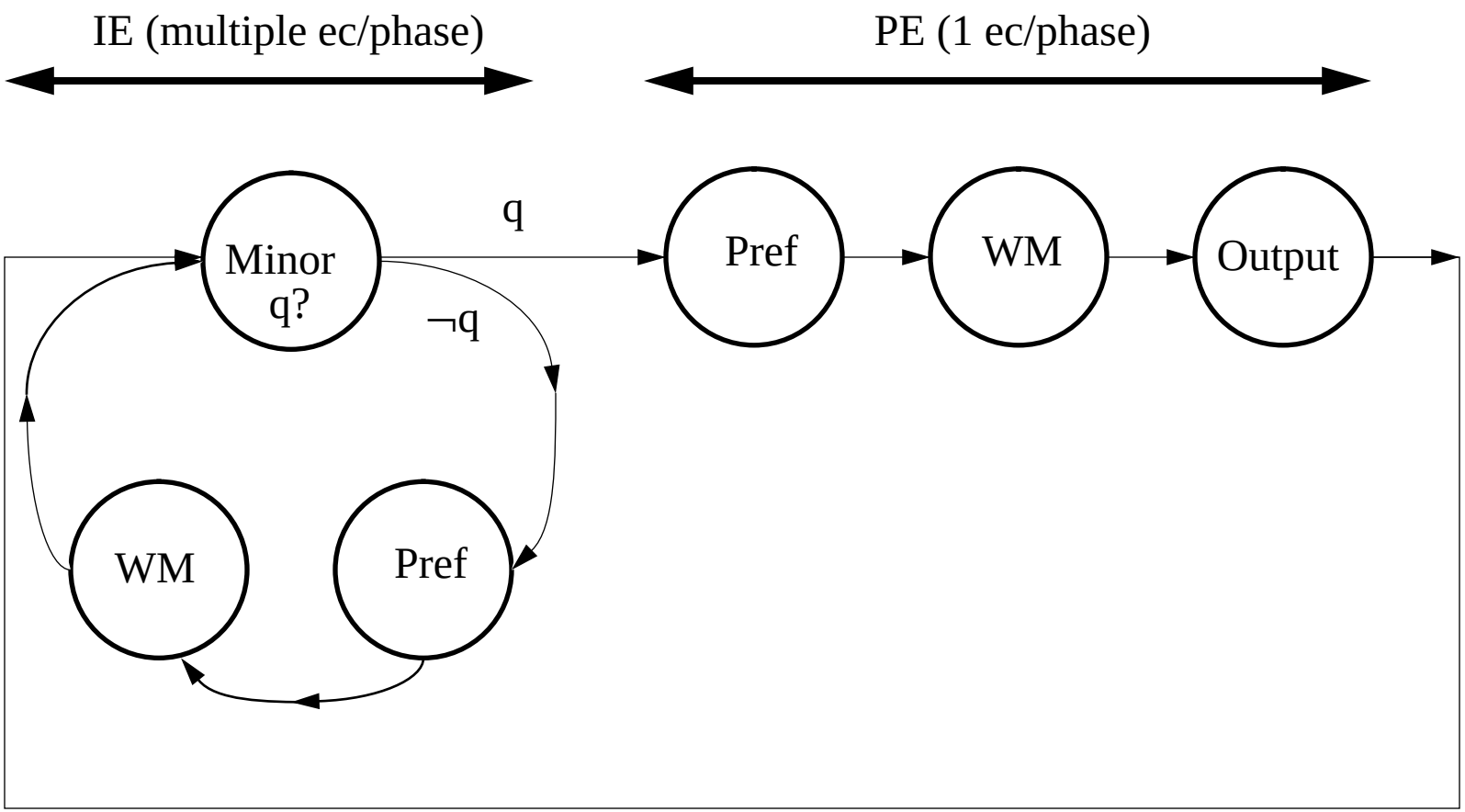
Result:

- Elimination of NCC due to persistent elaboration
- Elimination of the application/elaboration race condition

Cost:

- Loss of some parallelism
- (potentially more elaboration cycles/decision)

IE & PE Superphases



3. Selection Consistency

A. Eliminate C-support

- Operator persists only as long as its proposal condition is true
- Operator is retracted as soon as preference is lost (not at decision)

B. Consistency checks during decision

- Check current preferences against current decision
- Remove decision if not consistent with preferences (+ any substructure)
- Check at minor quiescence (before any persistent structures are created)

Result:

- Eliminate NCC from inconsistent selections
- No problem solving below an impasse after impasse resolution
- No operator terminations

Cost:

- Additional operator proposals (& possible additional complexity)
- Performance: `run-preference-semantics`
- Learnability/Usability: “Why did my operator go away?!”

Example: Decision Consistency

```
S1:
O1: do-routine-tasks
  S2: (operator no-change)
  O2: ....
```

propose*respond-to-emergency
(fire → operator <o> + >)

When “fire” is detected, operator respond-to-emergency is proposed

Operand2: (Minor quiescence)

- Preferences have changed
- Check preferences against current decision
- Result is not consistent
  - respond-to-emergency is best choice
- Install respond-to-emergency

Soar 7:

- O1 must be terminated before new operator can be selected

Interruptions

Soar 7:

- Decisions were removed only in the decision phase
- Operator selections were removed only when reconsidered
- Always reach quiescence in each goal in the stack

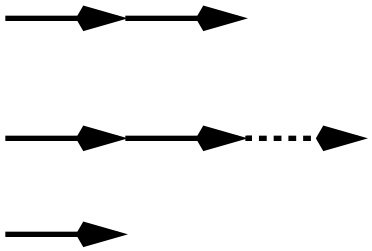
Operand2:

- Remove:
  - Goal when a WME in its GDS changes (immediate)
  - Operator selection whenever proposal is lost (immediate)
  - Decisions when inconsistent with changed preferences (minor q.)
- Decisions will not necessarily persist until (Soar 7) quiescence

Repercussions: A New Race Condition

Subgoal processing can be interrupted at any time

```
S1:
O1: operator-1
  S2: (operator no-change)
  O2: operator-2
    S3: (operator no-change)
    O3: operator-3
```



Assume O1 is retracted after its second serial application:

- O2 is interrupted before firing a third application
- O3 was able to terminate because it had only one application

Race Condition:

- Amount of problem solving in the subgoal in a given decision becomes a function of *number* of serial operator applications (as opposed to the content of the applications).

4. Goal-Limited Knowledge Retrieval

Only productions that match in the ‘active’ goal fire/retract

- Activity proceeds from top state to the bottom state (“Waterfall”)
- Results can trigger activity higher in the stack

Result:

- Ensures that all supergoal states are quiescent and their decisions are consistent before any subgoal processing proceeds.

  Make all progress possible in one state before proceeding to another
- Eliminates knowledge contention race condition

Cost:

- Less parallelism
  - Potentially many more elaboration cycles/decision
- Algorithm: Sort assertions and retractions

Knowledge Contention Solution

```
S1:
O1: do-task
  S2: (operator no-change)
  O2: determine-what-to-do
    S3: (operator no-change)
    O3: do-subtask-4
```

chunk-1
(O1, I1 == 3 → send-output <x>)

do-subtask-4*apply*send-output
(O3, I1 == 3 → send-output <x>)

Goal-limited knowledge retrieval

- Imposes conflict resolution for knowledge contention
- Productions matching goals higher in the stack are preferred to those matching goals lower in the stack

Architecture:

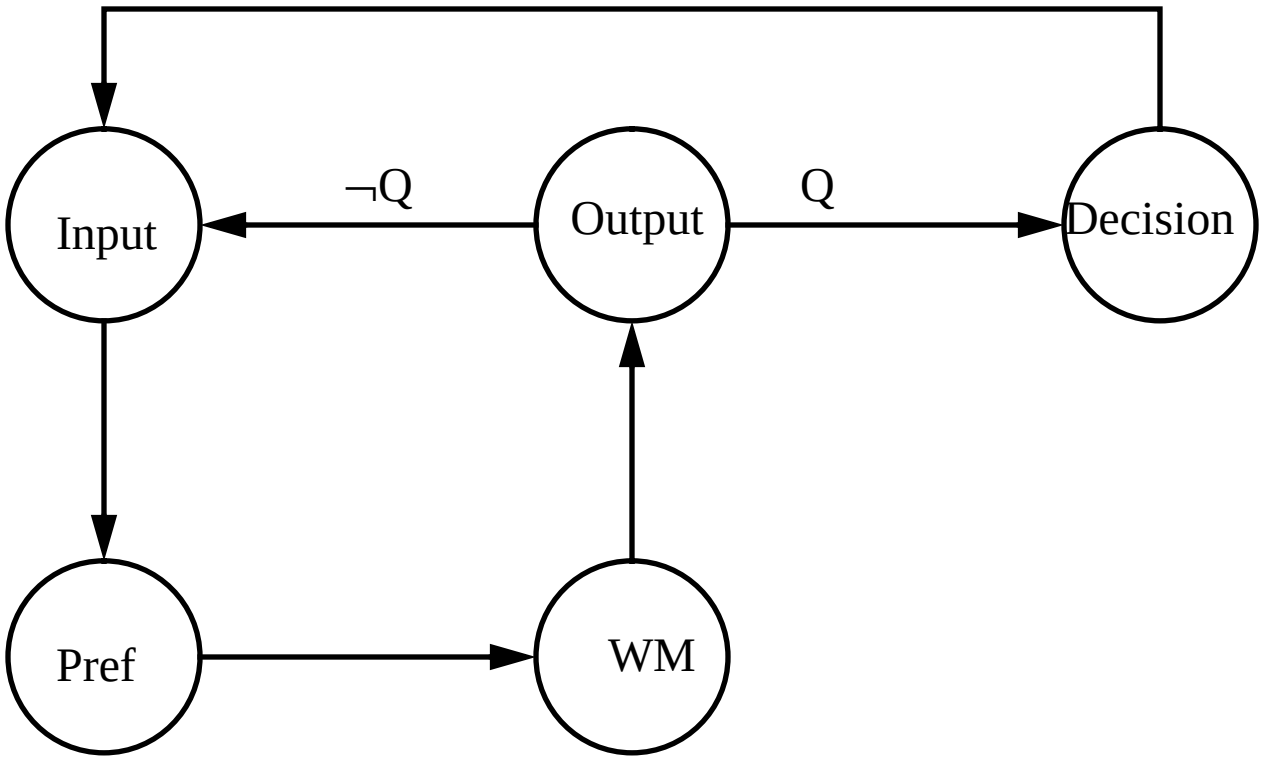
- Given a mixture of learned and deliberate behaviors, prefer the learned behaviors

Solution:

- Requires that knowledge test resource
- (or chunk leads to new decision)

do-subtask-4*apply*send-output
(O3, I1 == 3, !send-output
→ send-output <x>)

Soar 7 Decision Cycle



Changes to the Decision Cycle

New phase: Determine Level

- Checks for Quiescence
- Determines active production type (IE or PE)
- Determines active level
- If minor-quiescence, makes consistency check

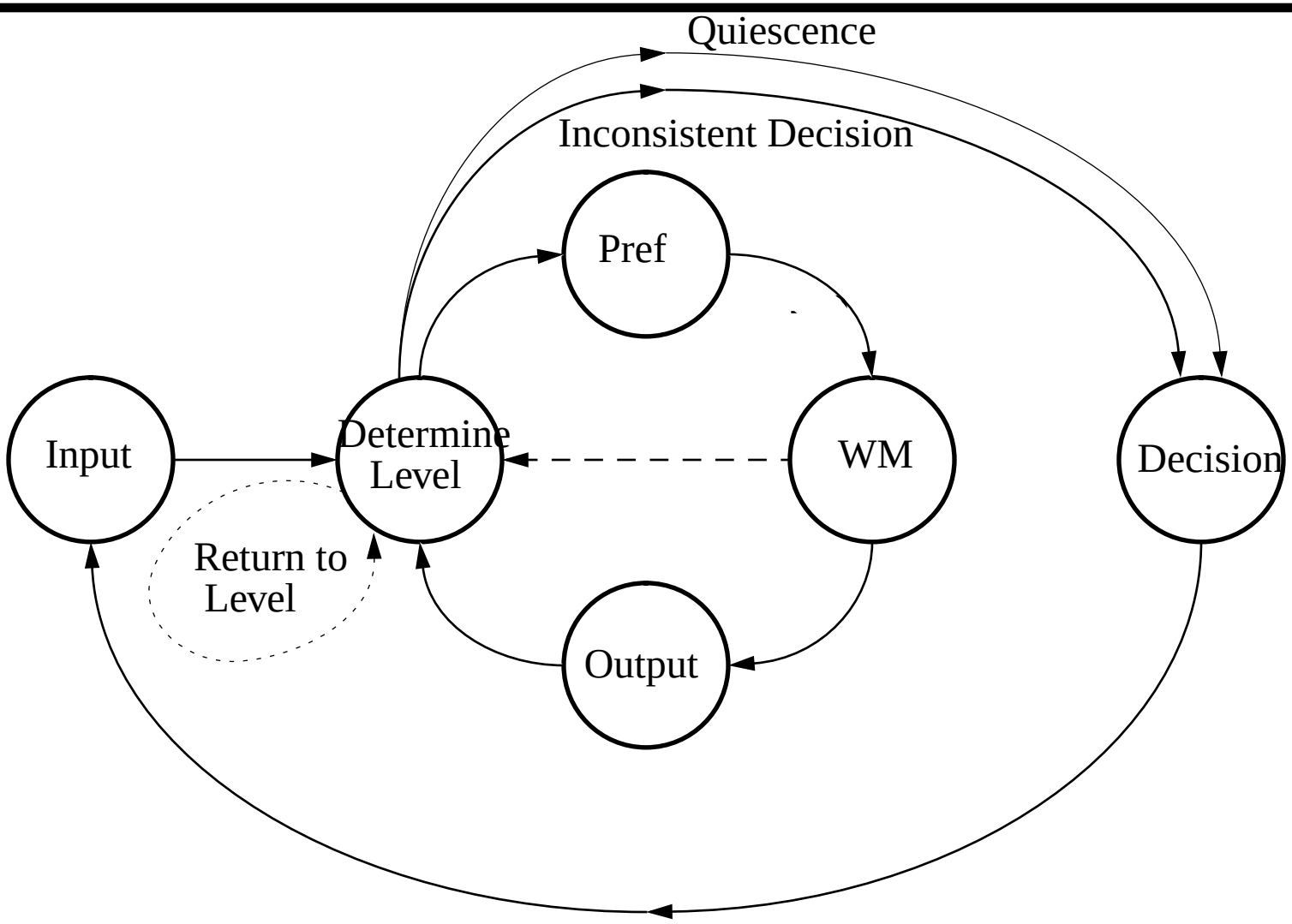
Input Phase: once per decision

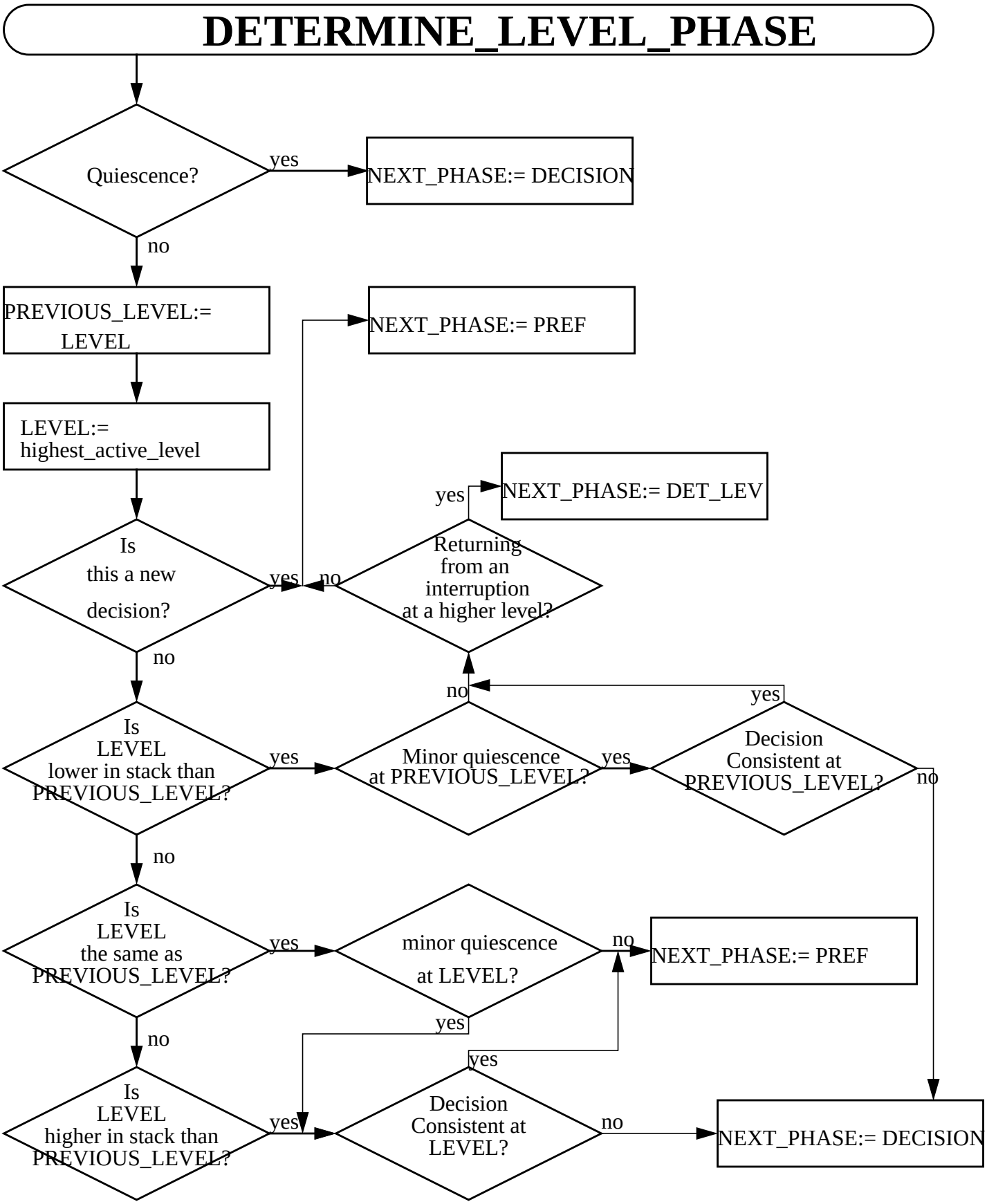
- Continuously changing input could make it impossible to ever reach bottom level
- Consistent with experience in other systems (e.g., TacAir-Soar)
- Can a system ever reach quiescence while also accepting input?

Production-type (IE/PE) for WM and Preference Phase

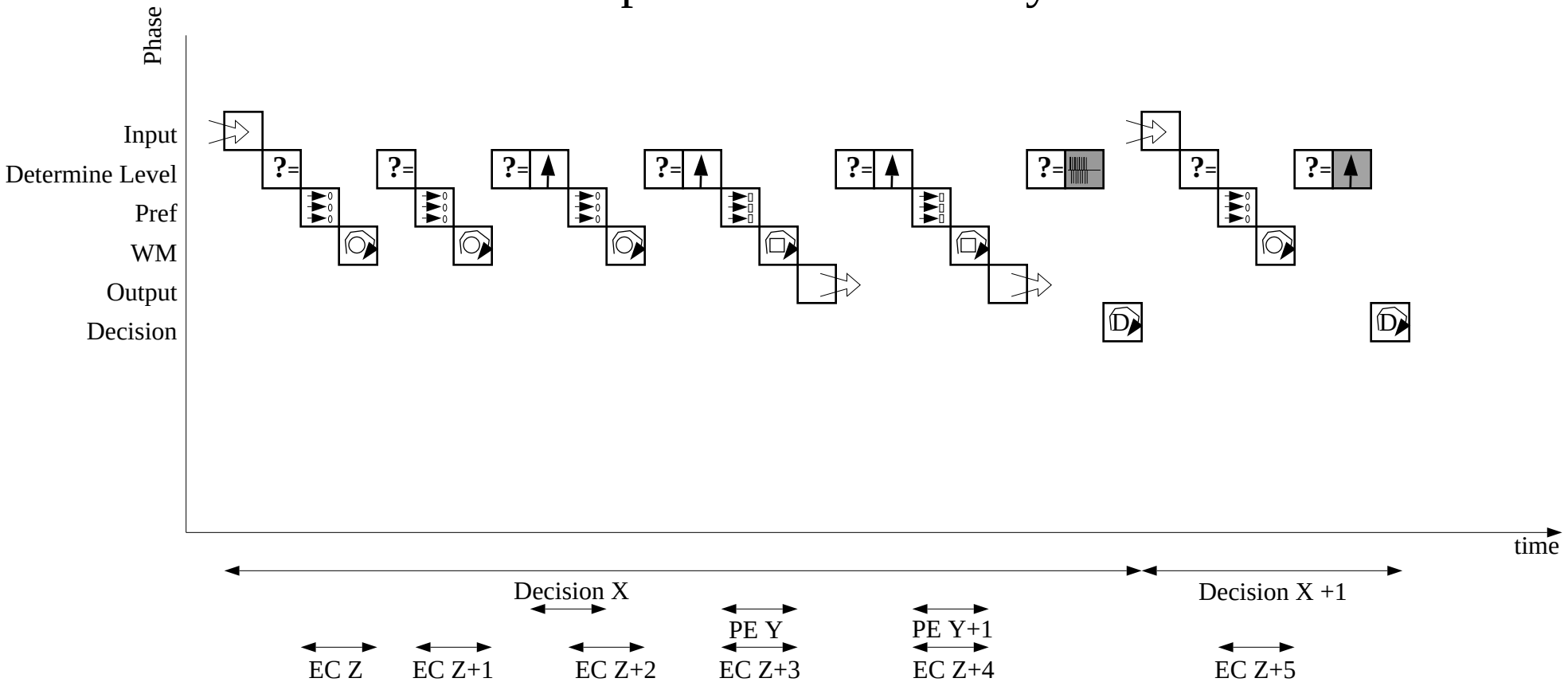
- Set in Determine Level Phase

Operand2/Soar Decision Cycle




DETERMINE_LEVEL_PHASE
Quiescence?
yes
NEXT_PHASE:= DECISION
no
PREVIOUS_LEVEL:= LEVEL
NEXT_PHASE:= PREF
LEVEL:= highest_active_level
yes
NEXT_PHASE:= DET_LEV
Is this a new decision?
yes
no
Returning from an interruption at a higher level?
no
yes
Is LEVEL lower in stack than PREVIOUS_LEVEL?
yes
Minor quiescence at PREVIOUS_LEVEL?
yes
Decision Consistent at PREVIOUS_LEVEL?
no
no
Is LEVEL the same as PREVIOUS_LEVEL?
yes
minor quiescence at LEVEL?
no
NEXT_PHASE:= PREF
yes
no
yes
Is LEVEL higher in stack than PREVIOUS_LEVEL?
yes
Decision Consistent at LEVEL?
no
NEXT_PHASE:= DECISION

The Operand2 Decision Cycle



Legend:

- Production Firings for O-Supported Preferences
- Production Firings for I-Supported Preferences
- Additions to Working Memory (o-supported)
- Additions to Working Memory (i-supported)
- Input Entering Working Memory
- Working Memory Passed to Output
- Decision Procedure
- Quiescence (No Productions Ready to Fire)
- Determine Active Level
- Consistency Check (dark: inconsistent decision)

Impact: Soar Theory, Systems, & Users

How will Operand2 impact:

- Knowledge design?
  - Existing systems
  - New systems
- Soar metrics?
- Overall performance?
- Timing Studies/Cognitive modeling?
  - Ron Chong (others?)
- Learnability and usability?
  - Conclusions

Impact: Knowledge Design

General Expectation:

More constraint ($\rightarrow$ less debugging)

Specific Expectations:

- Greater care in what one makes persistent (o-support)
- Greater care in proposal conditions
- Greater care in impasse conditions
+ Less need to design explicitly for learning
+ No race conditions

Impact on Existing Systems: μTAS

Micro TacAir-Soar

- Subset of the IFOR/TacAir productions
- Goal hierarchy of execution knowledge (little internal reasoning)
- Simulates 2-v-1 tactical air combat
  - Knowledge for both lead and wing
- 588 total productions (32 operators)
- Not possible to run with learning on

  Results in rules with NCC/knowledge contention failures

Preliminary results of conversion to Operand2:

- No learning data: Chunking works but chunks are over-specific
- Not tested with Goal-Limited Knowledge Retrieval (Waterfall)
- Behavior not validated by “expert”
- Non-deterministic domain
  - very hard to make performance and behavior comparisons

μTAS Conversion Data

Soar 7: 588 productions

Operand2: 550 productions

Deletions:	
DT: delete termination production	- 33
DSP: delete the ‘suggest-proposal’	- 11
DP: deleted production	- 5
Additions:	
API: additional proposal for intermediate state	+ 1
ATM: additions for timing (must be done in top state)	+ 5
NO: New operator added to goal hierarchy	+ 3
NP: New Productions	+ 2
Modifications:	
CT: change termination (action in addition to reconsider)	1
CP: change proposal (modify preconditions)	5
MPE: modification of OA due to change of precondition element	3
MIE: modification of application due to internal element (GDS)	0
M: Miscellaneous modifications	11
Total number of changes:	80
“Easy” changes:	44

Bad news: 36 modifications were hard work

- domain/knowledge analysis

Good news:

- Over 86% of the knowledge required entirely no modification
- Decrease in total production knowledge for task (-6.5%)

μTAS Conversion: DSP

DSP: Delete Suggest Proposals

- Actual operator preconditions used to create “suggest-proposal” structure
- Operator proposal: tests for suggest-proposal structure
- Operator termination: tests for absence of suggest-proposal

General way of having operators terminate when proposal is lost

Unnecessary in Operand2

- Operators are architecturally retracted whenever the proposal is retracted

μTAS Conversion: New Operators

Soar:

- Multi-step operators (without implementation subgoals)
- Proposal conditions based on first step of process
- Termination condition based on completion of last step of process

Operand2:

- Difficult to write multi-step operators
- Proposal condition must match throughout the entire operator
- Solution: break a multi-step operator into multiple, single-step operators
- Example of additional constraint:
  - Really want simple operators
  - debugging, composability, (psychological plausibility), etc.

New Operator: push-fire-button

Soar 7:

- push-fire-button proposal: no missile waiting to clear aircraft
- Action1:

  Output command to push the plane's fire button

  Leads to creation of a missile waiting to clear aircraft
- Action2:

  Count for a number of seconds until missile has cleared
- Terminate when missile has cleared (in flight)

Operand2:

- Problem: Action1 leads to immediate retraction of push-fire-button
- Solution: 2 Operators

  Action1: push-fire-button

  Action2: wait-for-missile-to-clear

Timers will generally require separate operators (GDS-generated retractions)

μTAS Conversion: Clean Up

Soar 7:

- Operators never interrupted in mid-decision
- Write items on state, clean-up before termination
- suggest-proposal structure allowed recognition of necessity of clean-up

Operand2:

Not possible to guarantee deliberate clean-up

Solutions:

A. Store local data under i-supported structure

- Build i-supported structure with similar conditions as operator proposal
- Will get removed with operator (similar to suggest-proposal)

B. Store anything that needs to be cleaned up on the operator itself

- Retraction of operator results in automatic (architectural) clean-up

Performance Comparisons

μTAS

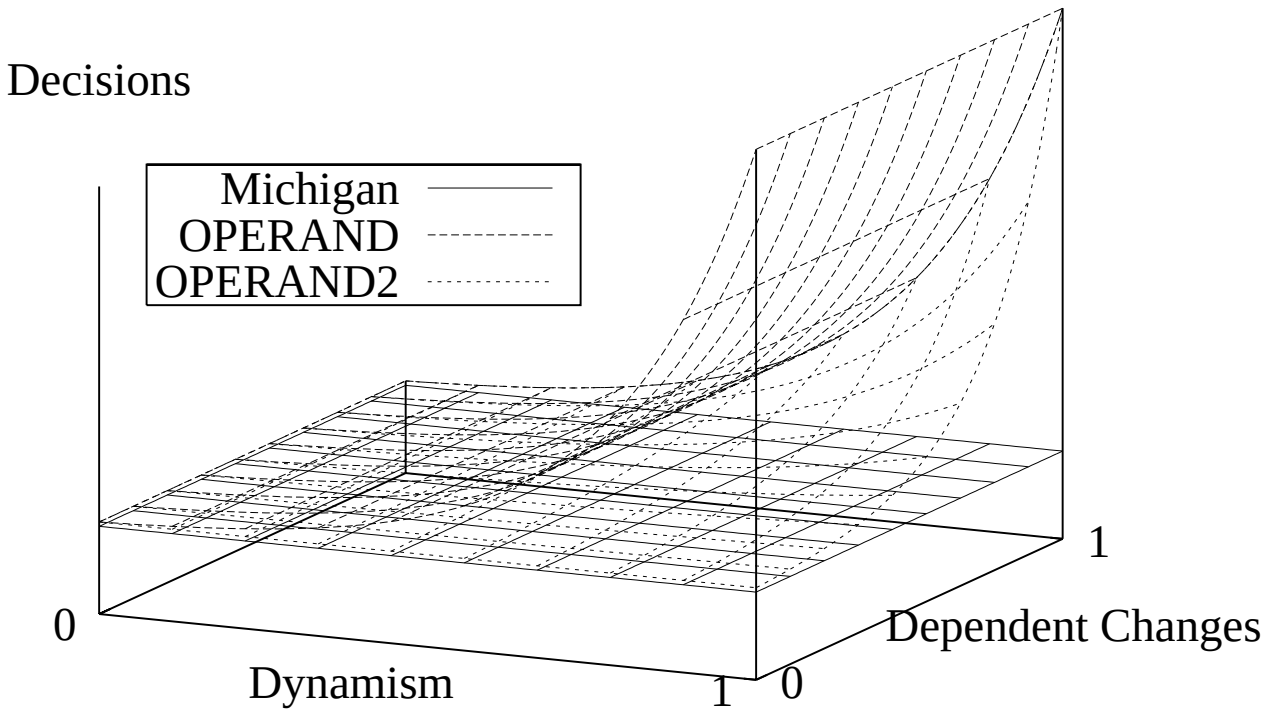
- Difficult to make exact comparisons due to non-determinism
- Rough comparison (5min run)

Soar 7		Operand2 1.0	
Decisions:	3172	Decisions:	3180
Elaboration Cycles:	4906	Elaboration Cycles:	12572
Production Firings:	7074	Production Firings:	18711
Kernel Time:	18.6 sec	Kernel Time:	23.9 sec
Total CPU Time:	45.5 sec	Total CPU Time:	42.7 sec

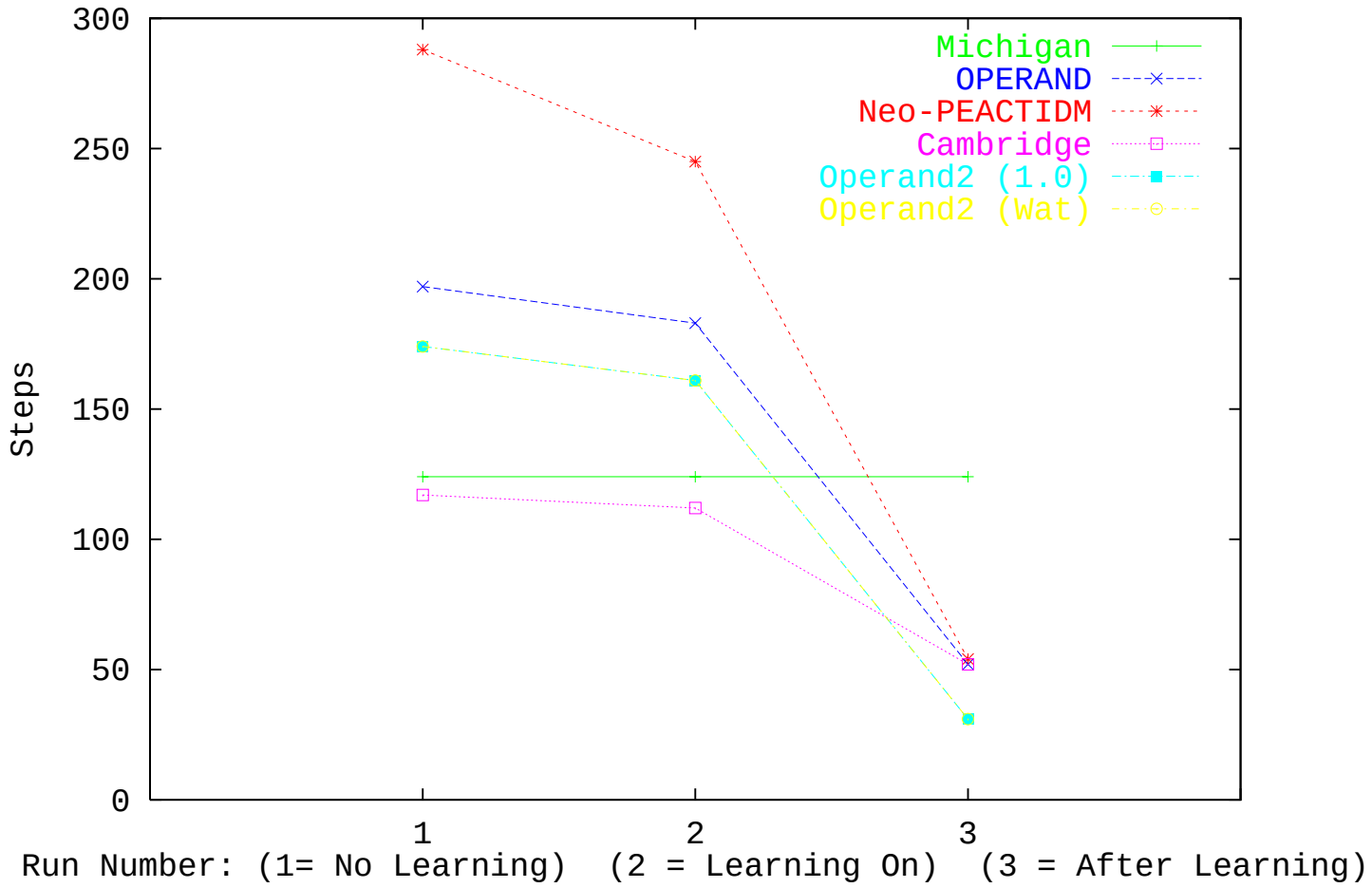
Dynamic Blocks World Test Bed

- Compare performance for simple, deterministic tower-building task
  - Soar 7
  - Operand2 1.0 (no Goal-limited knowledge retrieval)
  - Operand2/Waterfall (Goal-limited knowledge retrieval)

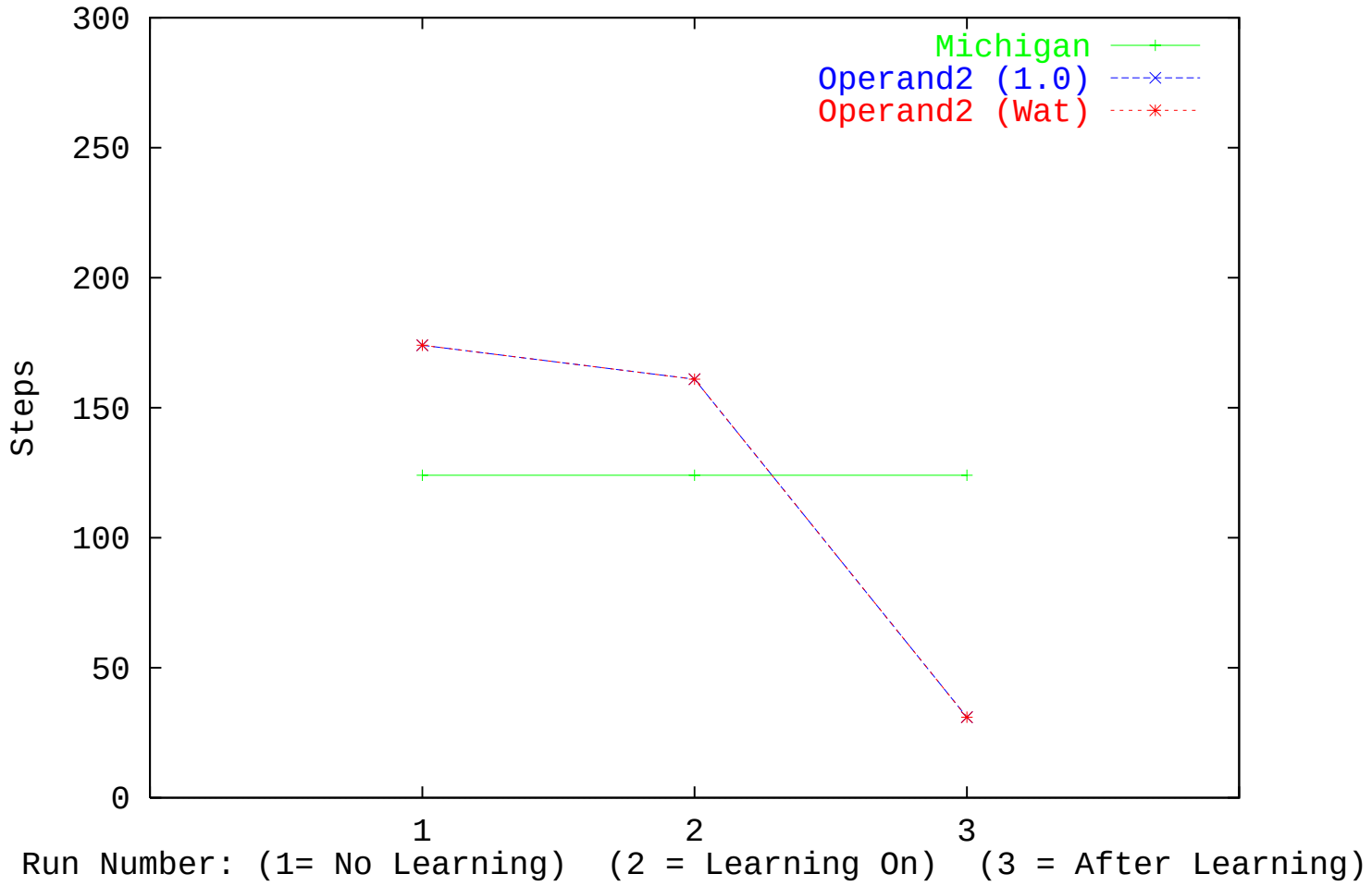
Expected Decision Differences



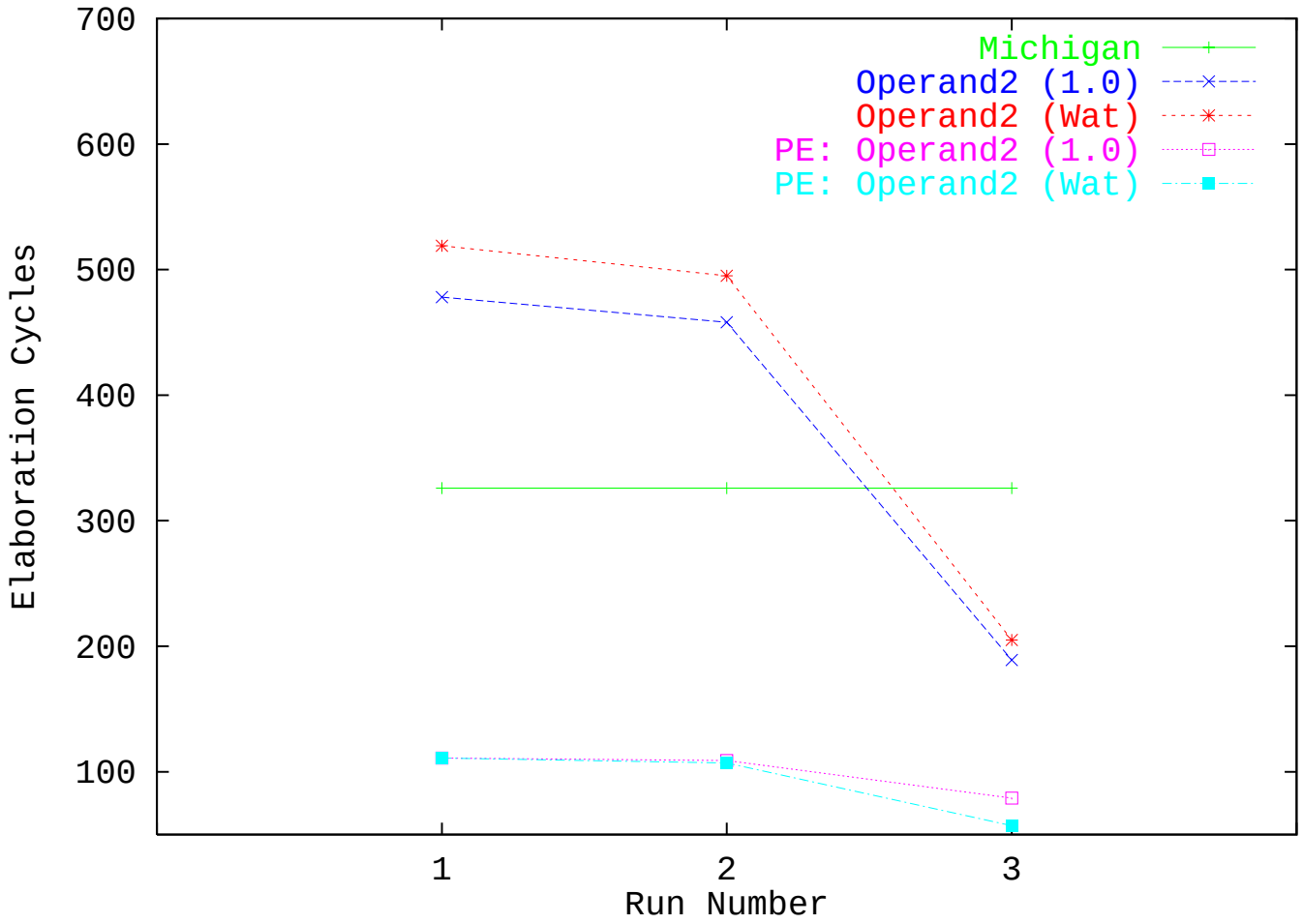
Decision Cycle Comparison



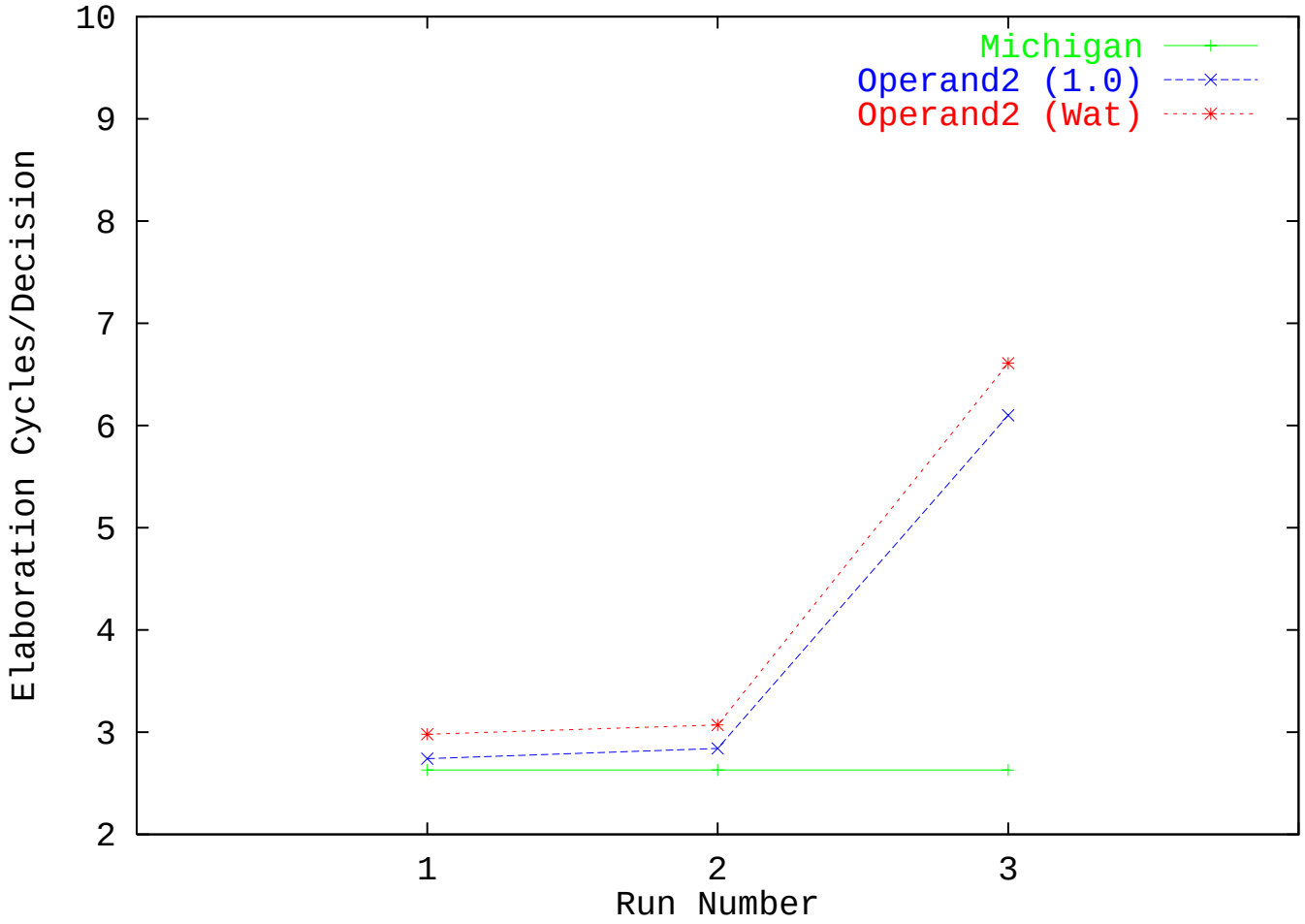
Decision Comparison



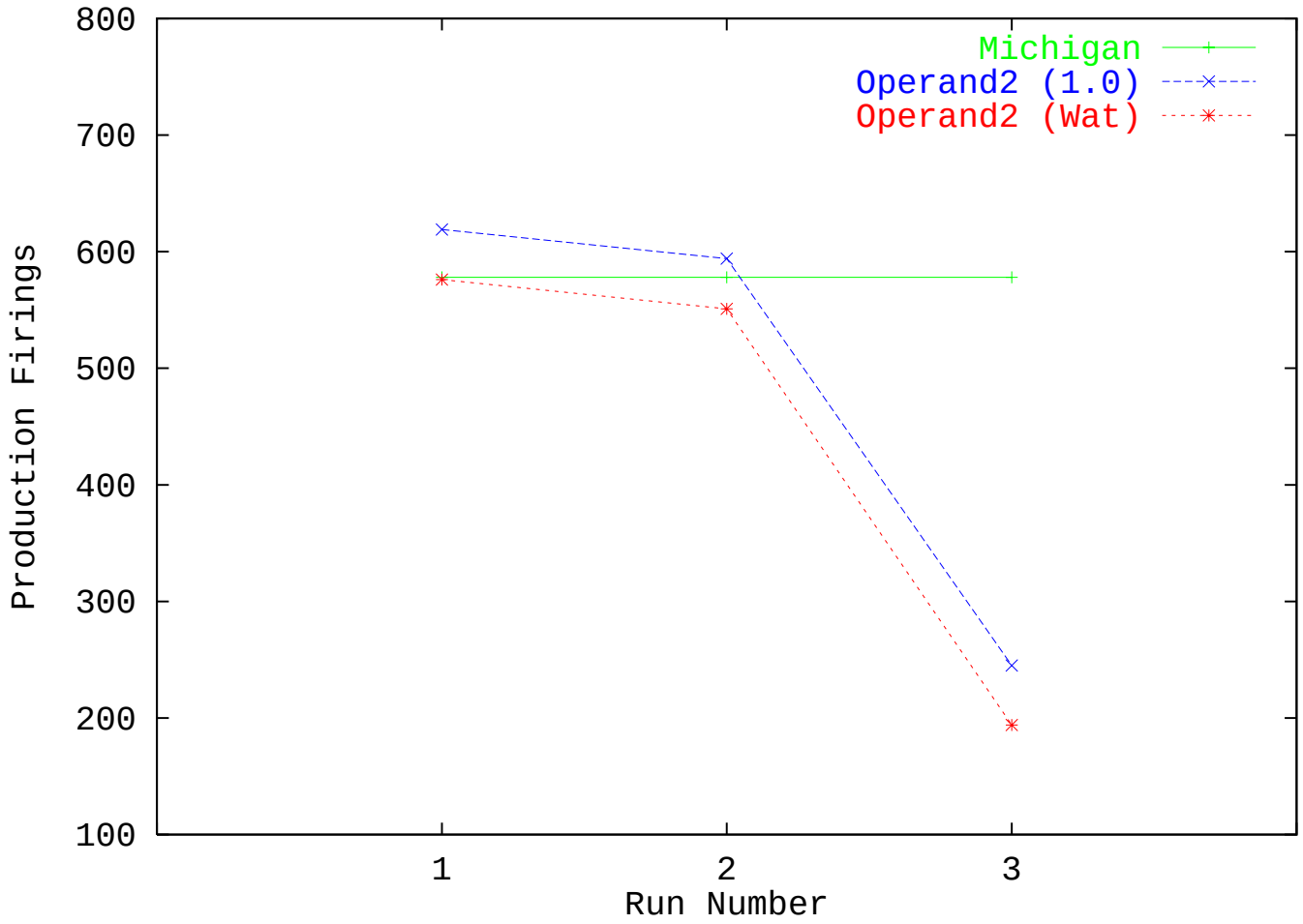
Elaboration Cycles



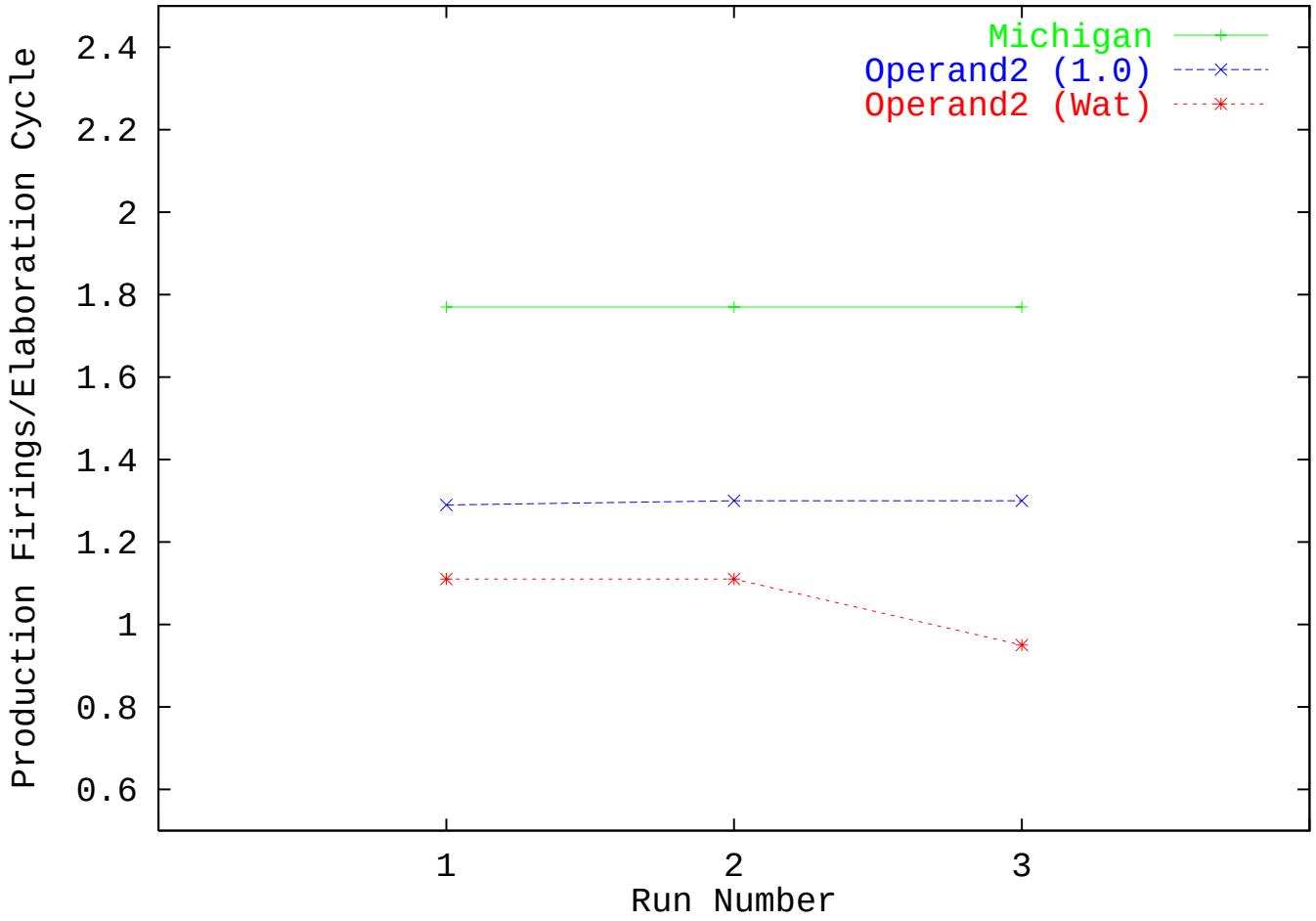
Elaboration Cycles per Decision



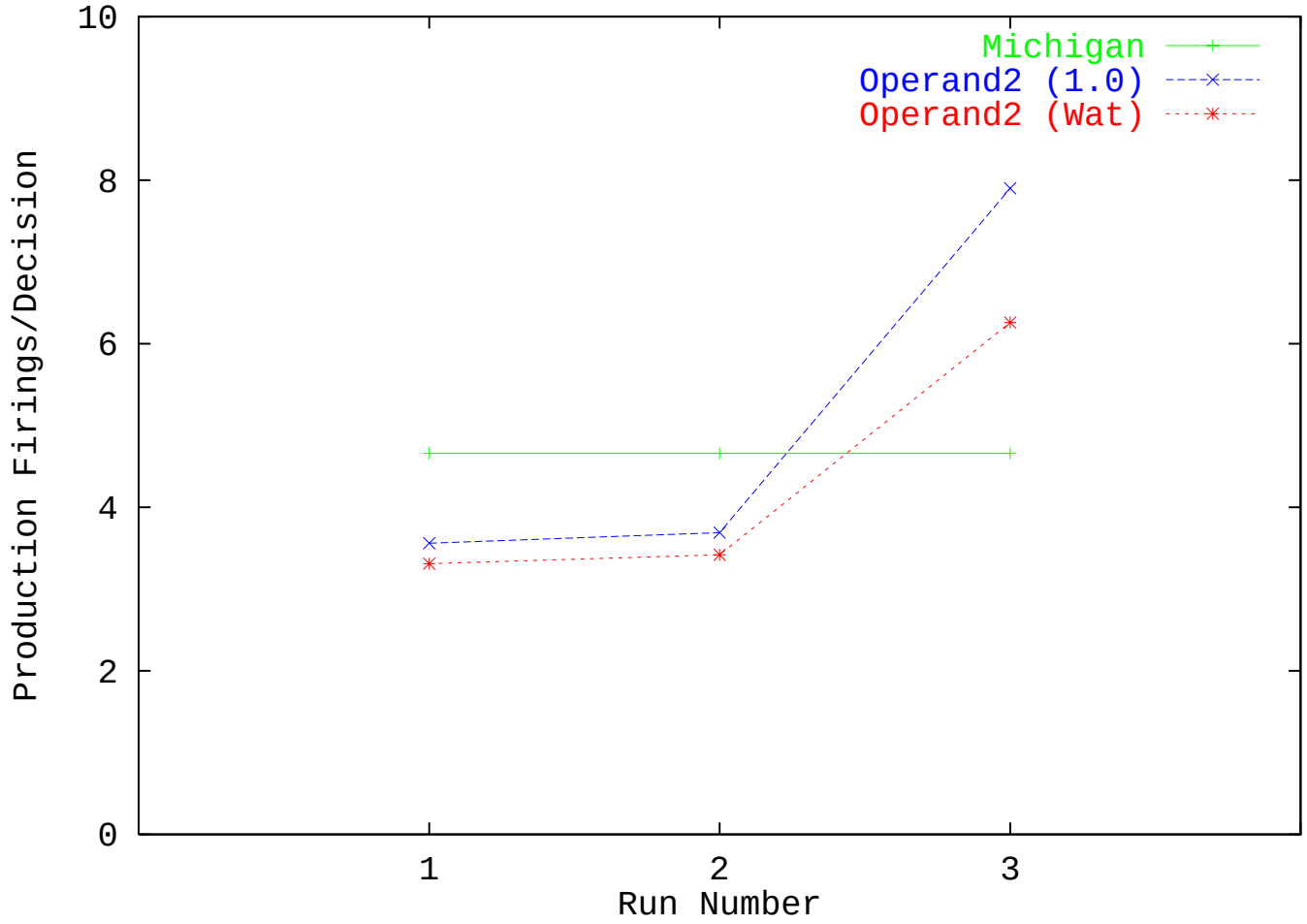
Production Firings



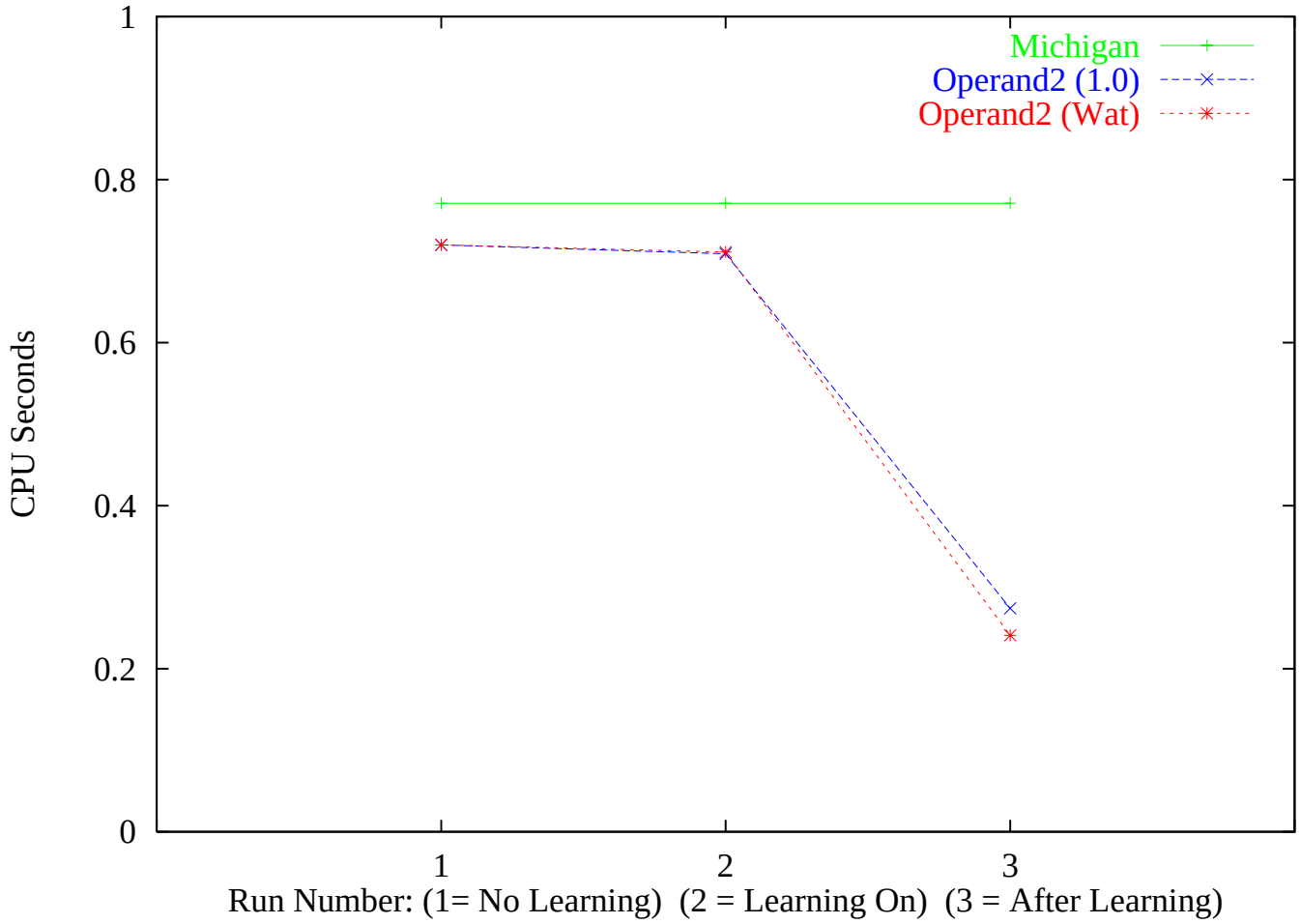
Production Firings/Elaboration Cycle



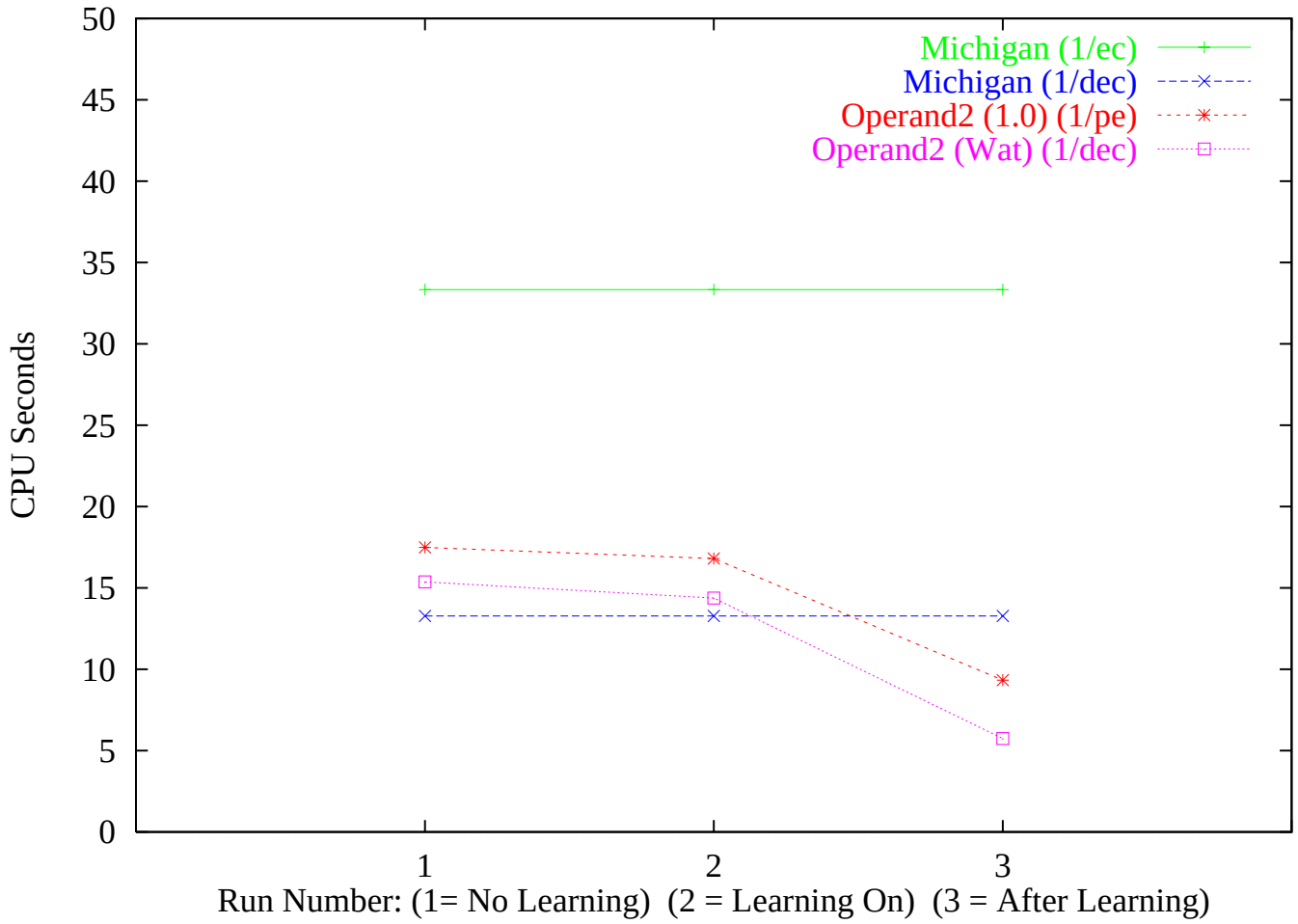
Production Firings/Decision



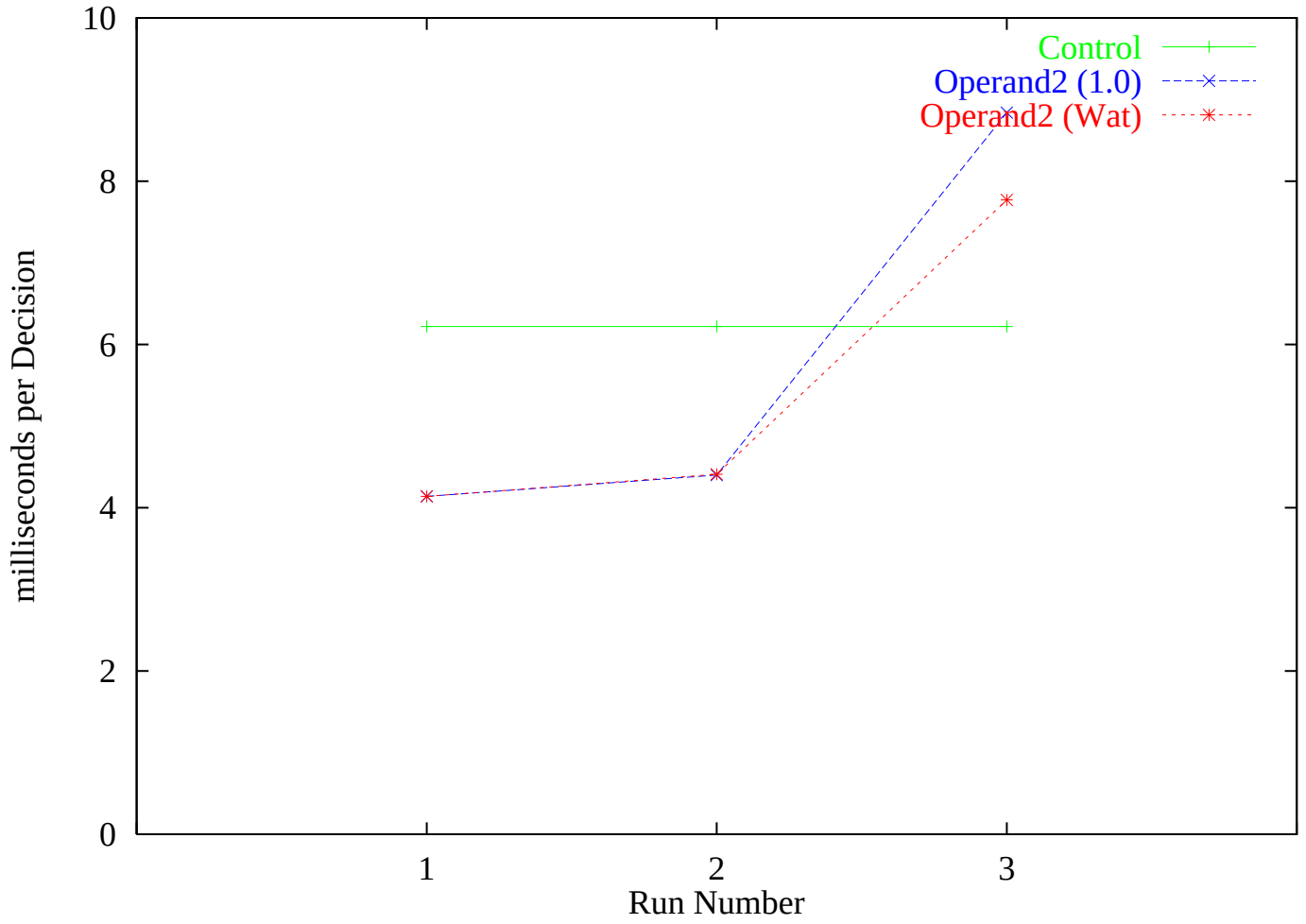
Kernel Time


Michigan
Operand2 (1.0)
Operand2 (Wat)
CPU Seconds
1
0.8
0.6
0.4
0.2
0
1
2
3
Run Number: (1= No Learning) (2 = Learning On) (3 = After Learning)

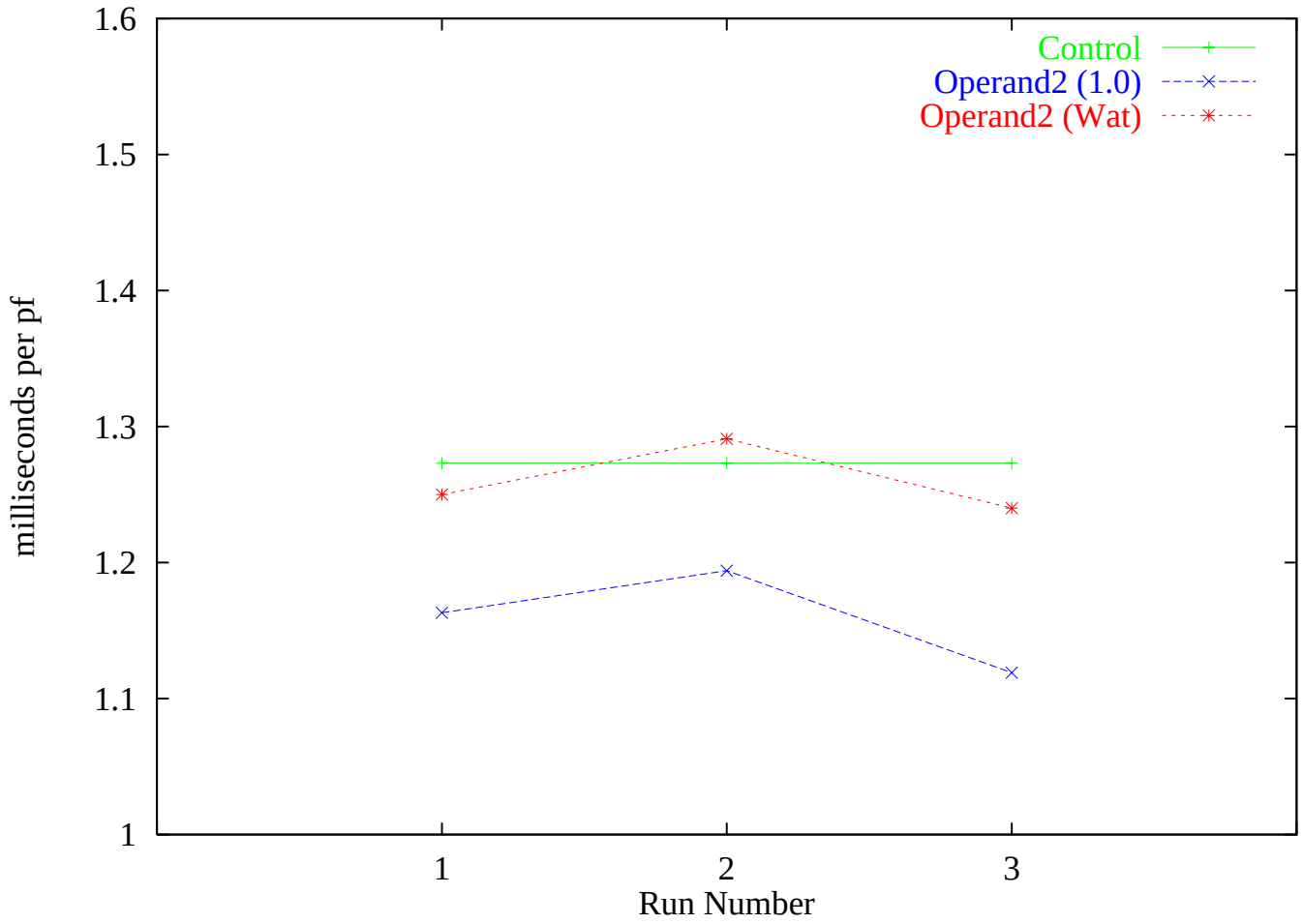
Total Execution Time



Time (ms) per Decision



Time (ms) per Production Firing



Synopsis of Performance Results

Decisions:

- Increase: GDS removals

Elaborations:

- Increase: Split of application and elaboration
- Increase: Goal-limited knowledge retrieval
- [Decrease: single input/decision]

Total production firings:

- Increase: Elaboration of new decisions
- Decrease: No knowledge retrieval after impasse resolution

  Block stacking task: net decrease

Execution time:

- Increase: Additional functions
- Decrease: More efficient support calculations

  Block stacking task: net decrease also due to decrease in production firings

Future Work

Implementation:

- More efficient algorithm for goal-limited knowledge retrieval
- Negations for goal dependency set
- Annotations of trace for improved understandability
- *Operator elaboration support
  - Should not be able to search through different operator elaborations?

Testing and validation:

- More tasks in blocks world
- Finish microTacAir-Soar conversion
- Conversion of an application/model with internal reasoning

Documentation:

- Tech Report (in progress) detailing changes:

  http://ai.eecs.umich.edu/soar/soar8/index.html

Conclusions

Operand2:

+ solves to problems it was designed to solve:
    - No observed non-contemporaneous constraints
    - No elaboration persistence race conditions
    - No knowledge contention
    - Decision terminates when impasse is resolved
- can increase the number of decisions & elaborations (wrt Soar 7)
    - Extra decisions: GDS removals
    - Extra elaborations: reduced parallelism
- exhibits moderate increases in CPU time/decision
    - Additional functionality

Conclusions (cont.)

Operand2:

? should have little impact on Soar as UTC

- Mostly implementation-level changes
- Existing models consistent with 1 PE = 50 ms hypothesis?

? Previous systems

- potentially difficult conversion process

+ additional constraint points to problems in code

? Learnability/Usability

- more complex system

+ additional constraint: less debugging, faster total development time