Towards Efficient and Scalable Omniscient Debugging for Model Transformations

Jonathan Corley, Brian Eddy, Jeff Gray

OCTOBER 19–24
SPLASH
PORTLAND 2014

DSM
Domain-Specific Modelling

THE UNIVERSITY OF
ALABAMA
COMPUTER SCIENCE

Outline

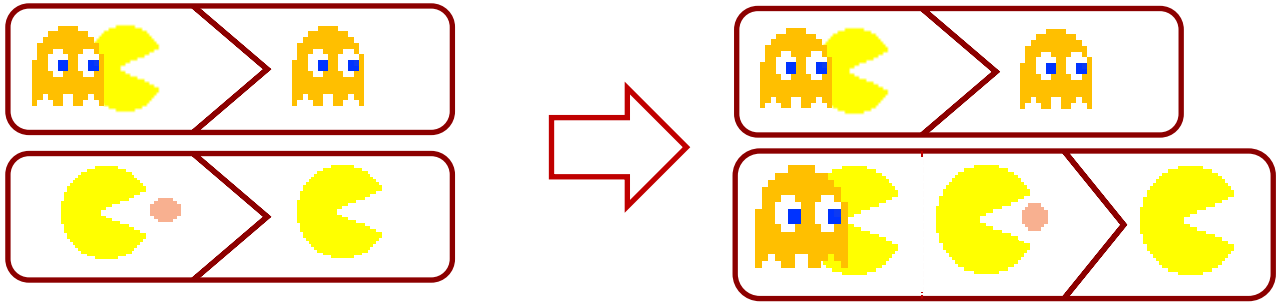
- Background and Motivation
- Omniscient Debugging for GPLs
- AToMPM Omniscient Debugger
- Future Concerns
- Comments & Questions

Model Transformations

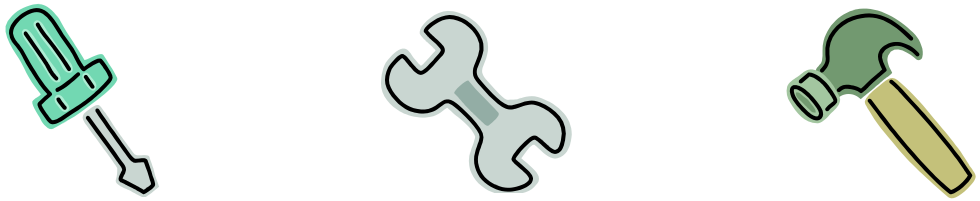
- Model Transformation
  - Endogenous vs. Exogenous MT
  - Inplace vs. Outplace MT
- Model Transformation Language
  - Declarative vs. Imperative vs. Hybrid MTLs
  - Nondeterministic Scheduling
- Higher-order Transformation

Need for MT Debugging Support

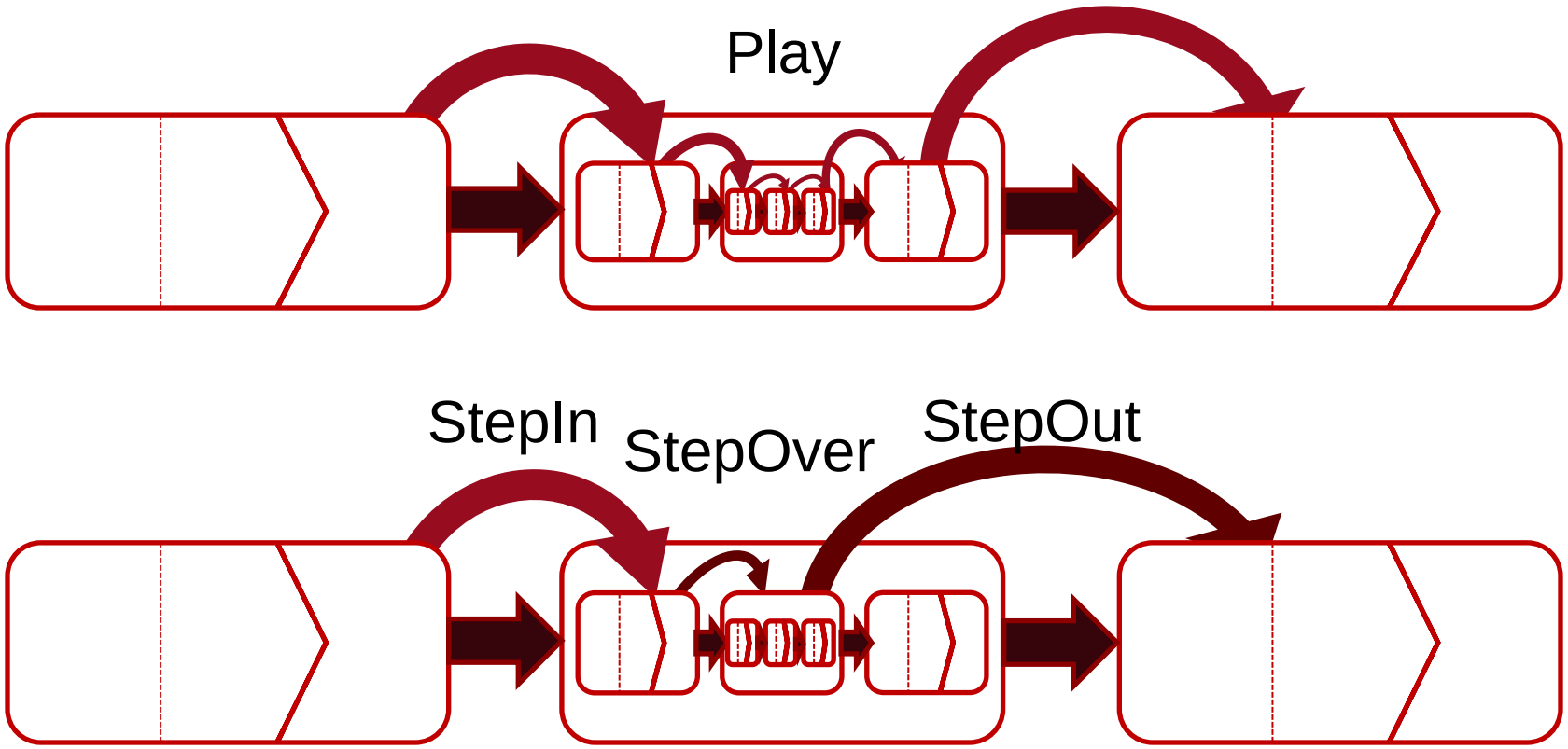
- Debugging is one of the most common tasks for software developers
- MDE practices do not eliminate all defects



- Tool support encourages adoption of MDE

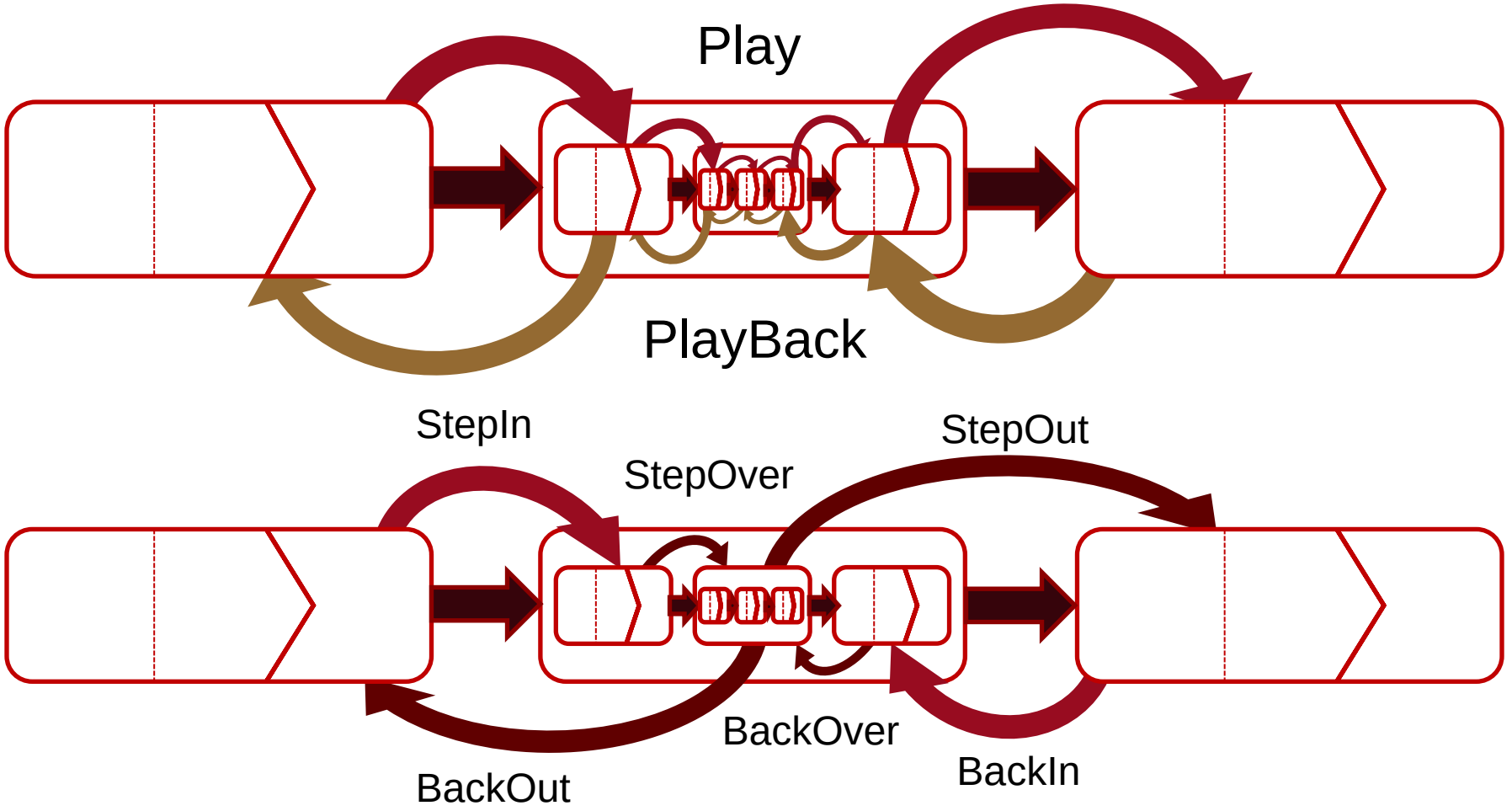


Stepwise Execution



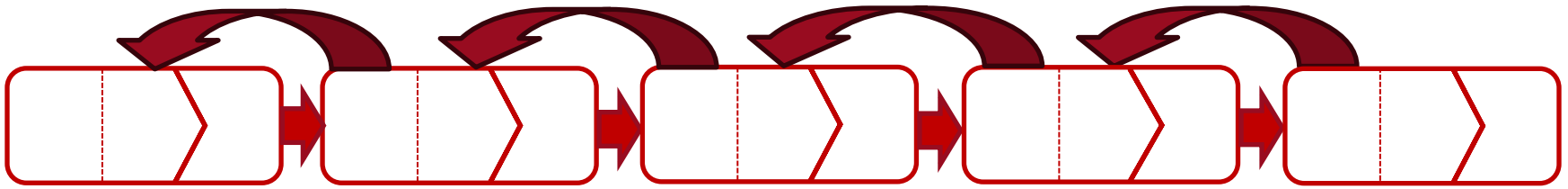
Supported by many modeling tools (e.g., ATL and TROPIC)

Extending Stepwise Execution



AToMPM Omniscient Debugger (AODB)

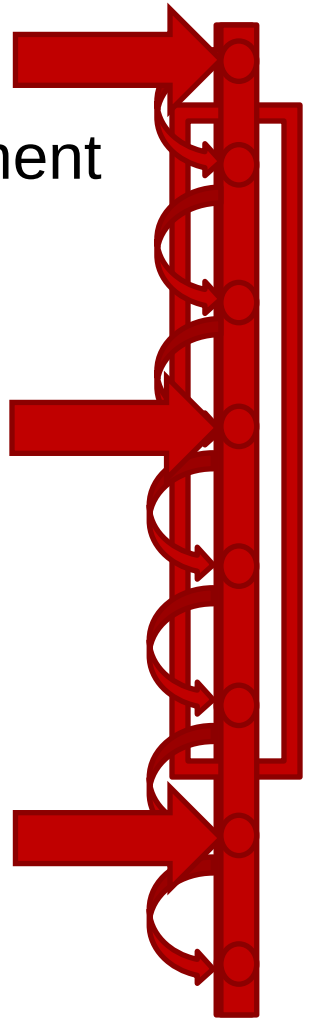
Omniscient Debugging



- Back-in-Time Debugging
- Is it needed?
  - Fault is located before the visible error
  - Failure is not present in all executions
    - Poorly understood error
    - Nondeterministic rule scheduling
  - Execution of the system is expensive
    - Time to execute
    - Manual input required

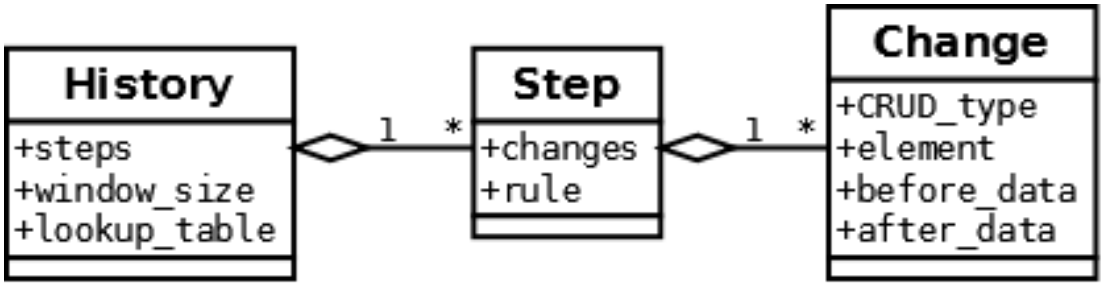
Tracing Strategies

- Challenges
  - Minimize time required to move to desired element
  - Minimize space consumption
- Previous Solutions
  - Select unimportant items to forget [1]
    - Static vs Dynamic
  - Maintain a sliding window of history [1]
  - Deltas [2]
  - Jump points [3]
  - Garbage Collection [3]



[1] Lewis “Debugging Backwards in Time” AADEBUG 2003
[2] Seifert “Opportunities and challenges for traceable graph rewriting systems” GRaMoT ‘08
[3] Lienhard et al. “Practical object-oriented back-in-time debugging” ECOOP ‘08

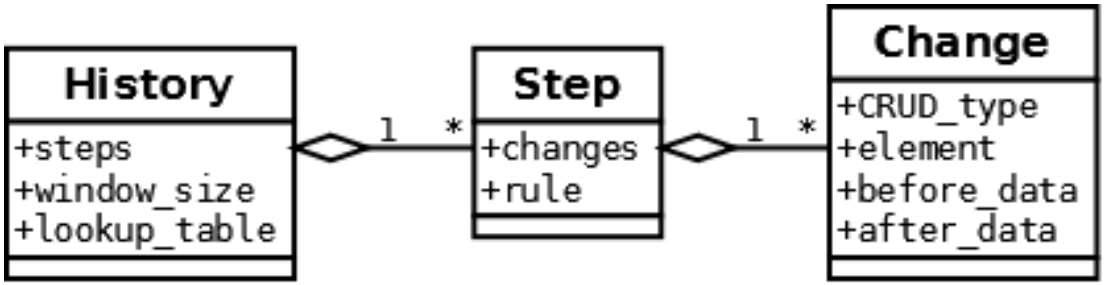
Collecting a History of Execution



How to revert execution?

- Collect trace of execution
- Step contains full state information for elements changed
- Window of history kept in memory, remainder in permanent storage

Collecting a History of Execution

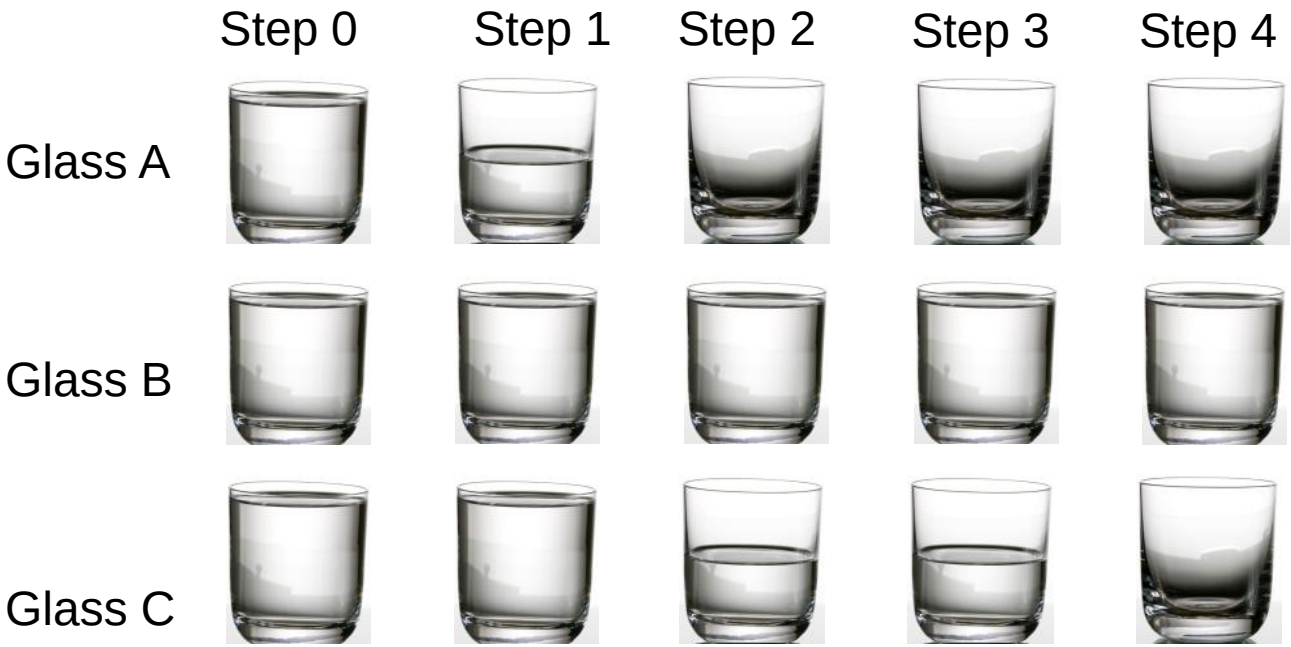


- Structure Space Complexity: O(n * (A + m*B))
  - n is number of steps
  - A is the step overhead
    - ~153 bytes in our prototype
  - m is average number of changes per step
  - B is the change overhead
    - ~407 bytes in our prototype
- Impact of Python on memory usage
  - Integer is 12 bytes
  - String is 21 bytes plus an additional byte per character

Efficient Traversal of History

Macro Steps

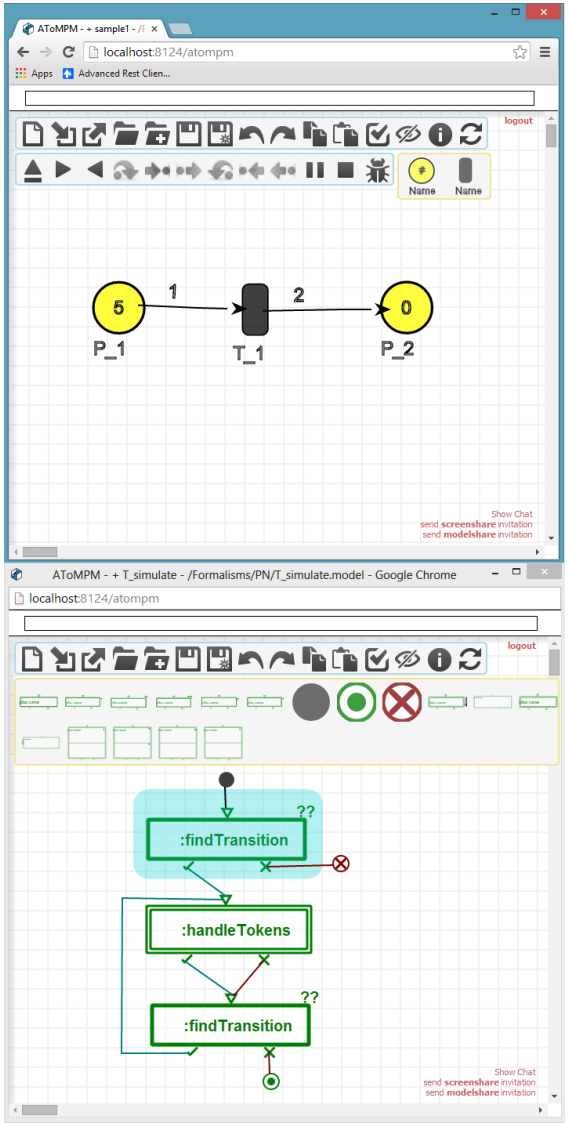
- Traverse many steps minimizing changes
- Runtime Complexity: O(n*lg(m)) vs O(n)



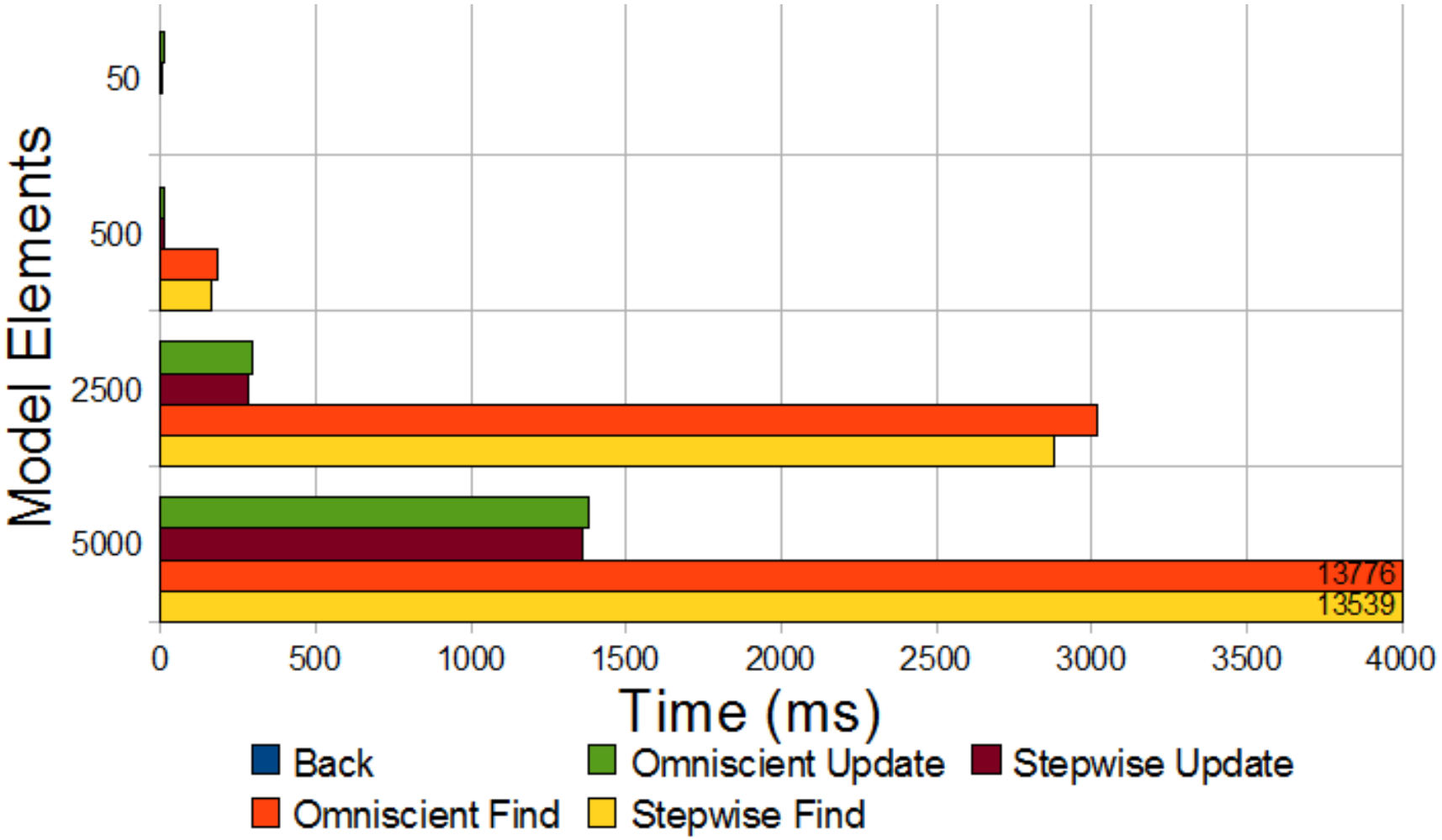
Performance Analysis

Compare to existing environment

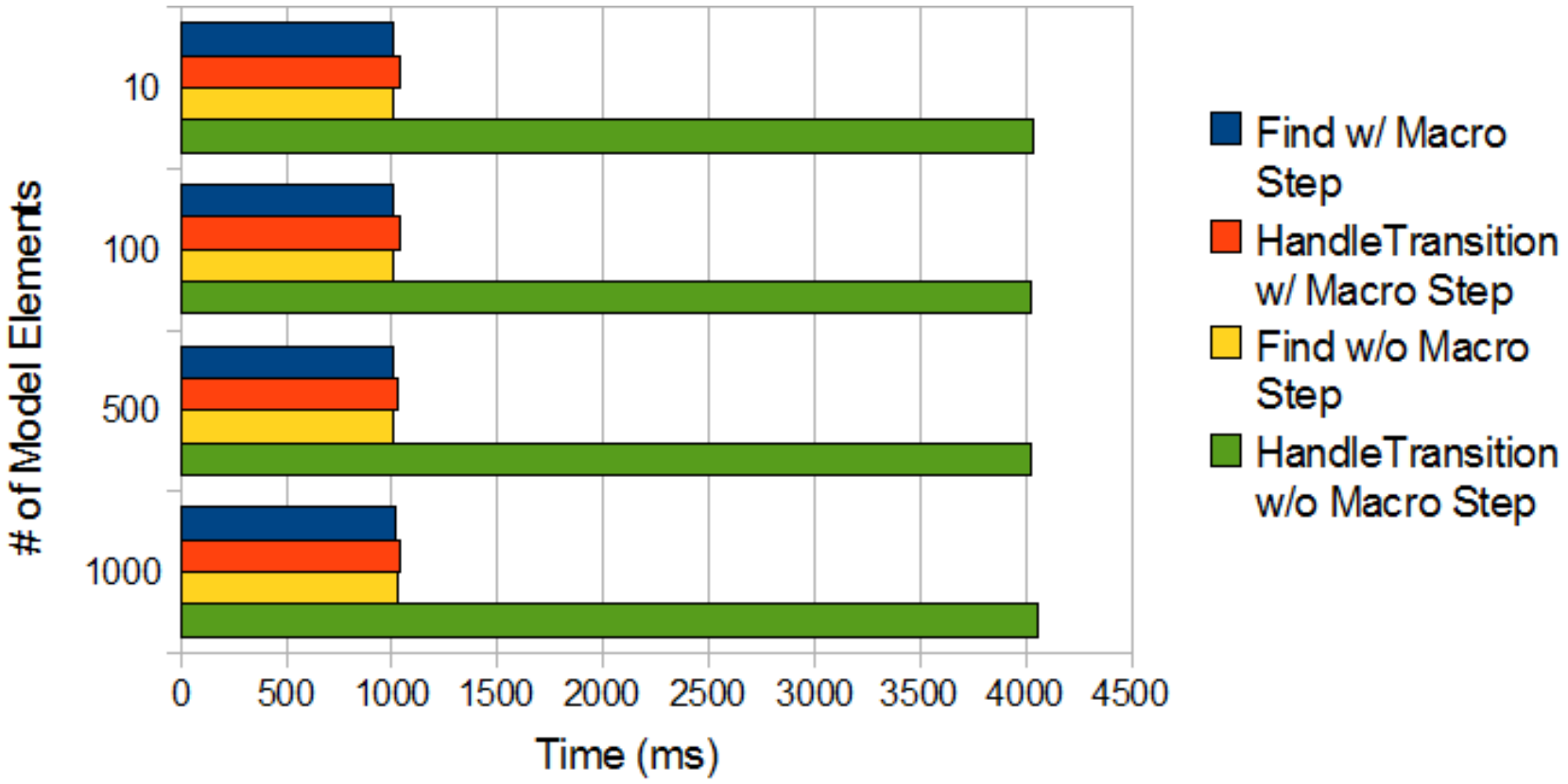
- Petri-net simulator
  - Find step
  - Update step
- Vary model (Petri-net) size
  - 50, 500, 2500, 5000
- Compare three traversal types
  - Stepwise forward
  - Omniscient forward
  - Omniscient backward



Performance Analysis Results



Further Performance Analysis



Future Concerns

- Integrating Breakpoints
- Model Everything…History
  - A view of history
- User study
  - The impact of omniscient debugging
  - Leading to query-based debugging
- Performance and Scalability Analysis
  - Larger scale and variety for models and transformations

Comments & Questions