AMOebA-RT: Run-Time Verification of Adaptive Software

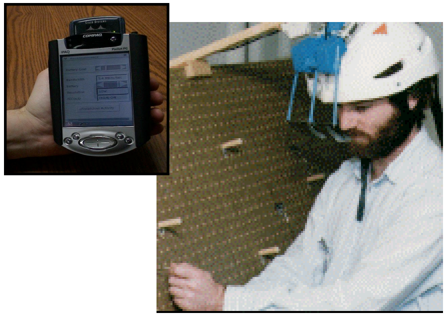
Ji Zhang, Betty H.C. Cheng, and Heather J. Goldsby

This work has been supported in part by NSF grants EIA-0000433, CNS-0551622, CCF-0541131, IIP-0700329, CCF-0750787, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Air Force Research Lab under subcontract MICH 06-S001-07-C1, Siemens Corporate Research, and a Quality Fund Program grant from Michigan State University.

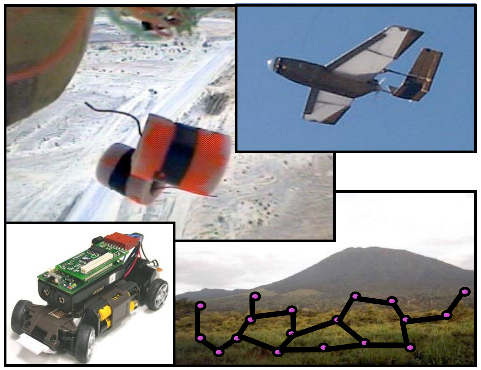
Need for Dynamically Adaptive Software

- Pervasive Computing
  - Promises anywhere, anytime access to data and computing.
- Autonomic Computing
  - Promises self-managed and long-running systems that require only limited human guidance.

Handheld/Wearable Computing



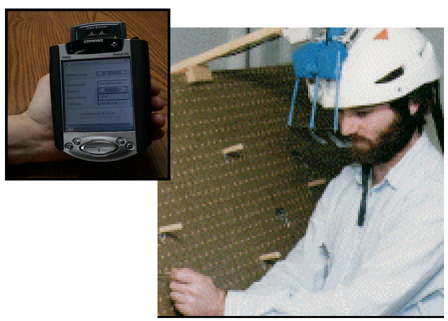
Sensor Networks



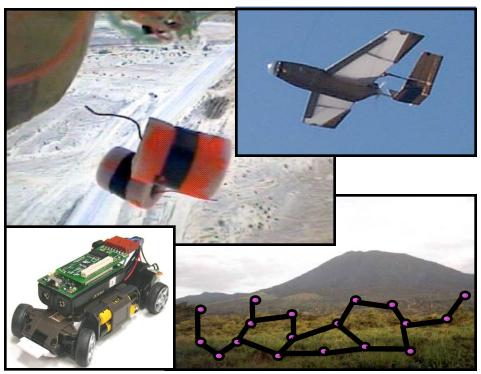
Need for Dynamically Adaptive Software

- Pervasive Computing
  - Promises anywhere, anytime access to data and computing.
- Autonomic Computing
  - Promises self-managed and long-running systems that require only limited human guidance.

Handheld/Wearable Computing



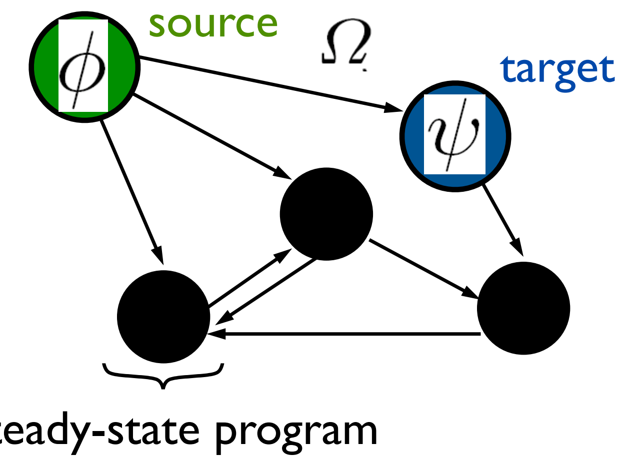
Sensor Networks



Assurance that the adaptive software satisfies its requirements is critical.

Adaptive Software

- Set of steady-state programs
  - Respond to changing environmental conditions
  - Non-adaptive

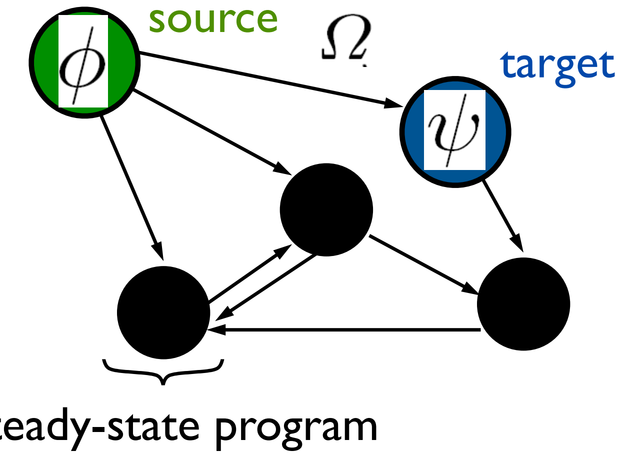


- A-LTL adaptation properties
  - Used to specify global, local, and transitional properties
  - Extends LTL with the adapt operator $\overset{\Omega}{\rightharpoonup}$
  - "$\phi \overset{\Omega}{\rightharpoonup} \psi$" Denotes an adaptation from satisfying ϕ to satisfying ψ where the adapting states satisfy Ω

MICHIGAN STATE UNIVERSITY

Adaptive Software

- Set of steady-state programs
  - Respond to changing environmental conditions
  - Non-adaptive
- A-LTL adaptation properties
  - Used to specify global, local, and transitional properties
  - Extends LTL with the adapt operator $\overset{\Omega}{\rightharpoonup}$
  - "$\phi \overset{\Omega}{\rightharpoonup} \psi$" Denotes an adaptation from satisfying ϕ to satisfying ψ where the adapting states satisfy Ω



How can a developer analyze adaptive software?

Analyzing Adaptive Software

- AMOebA (Adaptive program MOdular Analyzer)
  - Modular model checking of models of adaptive software (generally requirements or design)
  - Verifies adaptive properties specified in LTL and A-LTL
- AMOebA Limitations:
  - State explosion renders this approach *insufficient for complex adaptive software*
  - *Cannot be used to verify program code*

Analyzing Adaptive Software

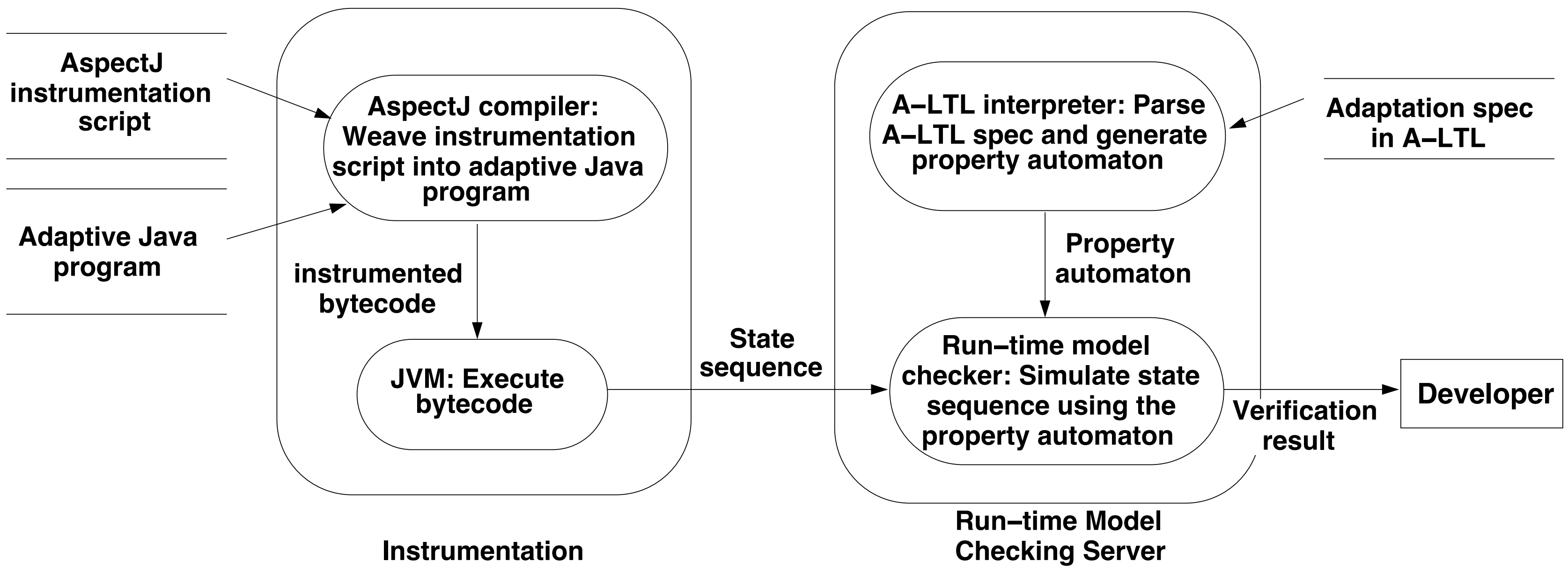
- AMOebA (Adaptive program MOdular Analyzer)
  - Modular model checking of models of adaptive software (generally requirements or design)
  - Verifies adaptive properties specified in LTL and A-LTL
- AMOebA Limitations:
  - State explosion renders this approach *insufficient for complex adaptive software*
  - *Cannot be used to verify program code*

Are there alternative, lighter weight approaches to analyzing complex adaptive software?

AMOebA-RT : A Run-time Model Checker

- Assurance
  - A-LTL specification of adaptation requirements
  - Verifies adaptive software code at run-time
- Automation
  - Non-invasive (aspect-oriented) instrumentation
  - Returns a counterexample if the verification fails
- Reduced verification complexity
  - Avoids state explosion

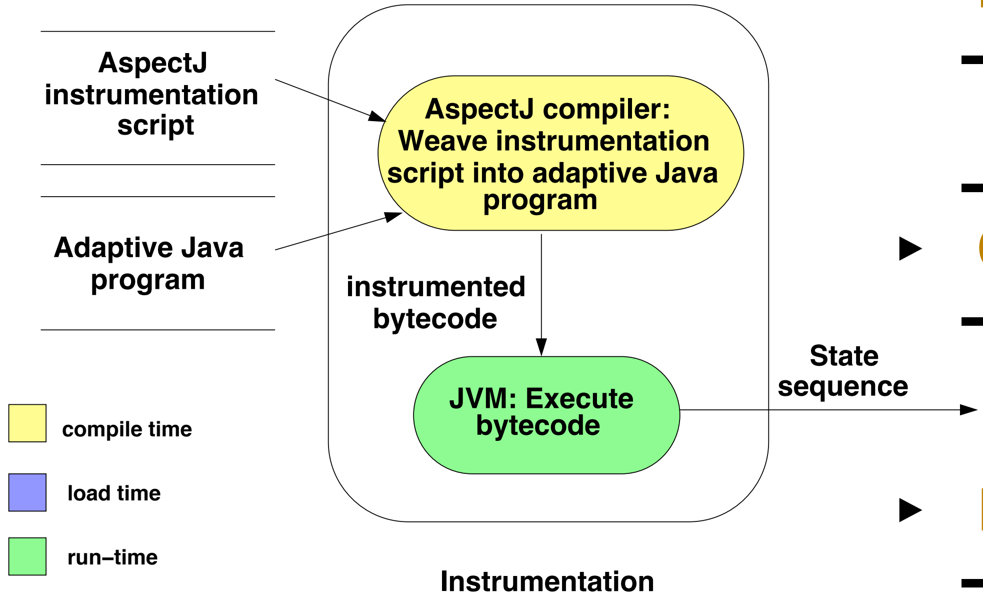
AMOebA-RT Architecture



Run-time Monitoring

Run-time Model Checking

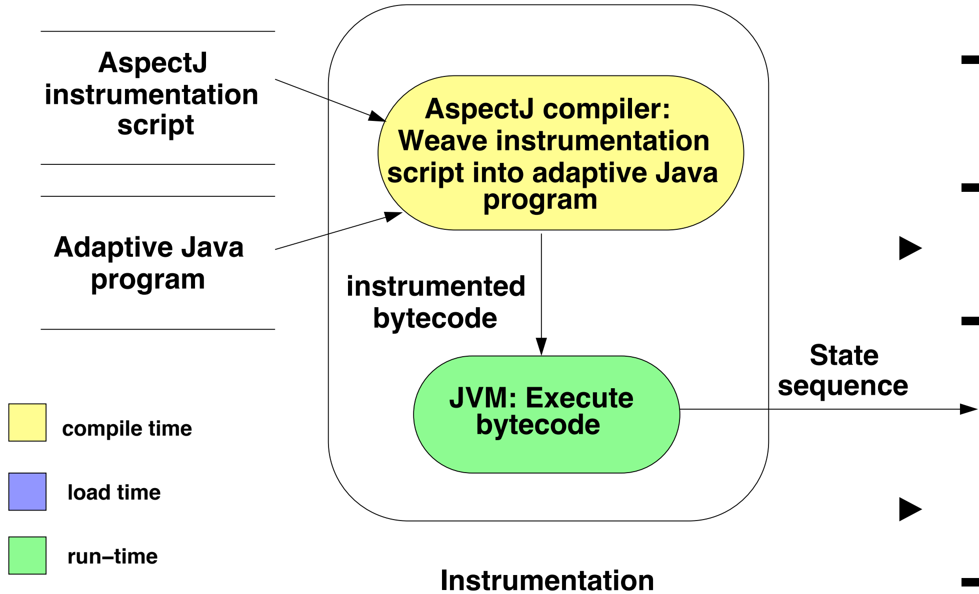
AMOebA-RT : Run-time Monitoring



- Before run-time:
  - Point-cuts identify state change locations.
  - Advice collects state information
- Compile time:
  - Weave monitoring instrumentation into the program (uses AspectJ)
- Run-time:
  - Instrumentation code produces and sends run-time state information to the model checker

Uses aspect-oriented approach

AMOebA-RT : Run-time Monitoring



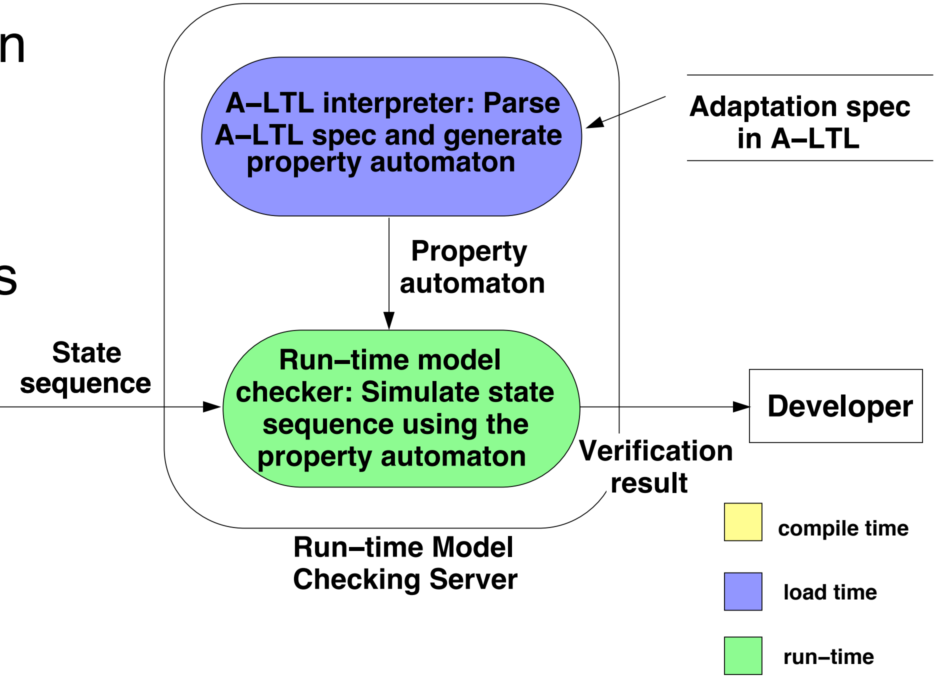
- Before run-time:
  - Point-cuts identify state change locations.
  - Advice collects state information
- Compile time:
  - Weave monitoring instrumentation into the program (uses AspectJ)
- Run-time:
  - Instrumentation code produces and sends run-time state information to the model checker

Uses aspect-oriented approach

Non-invasive: source code for the steady-state programs is not altered.

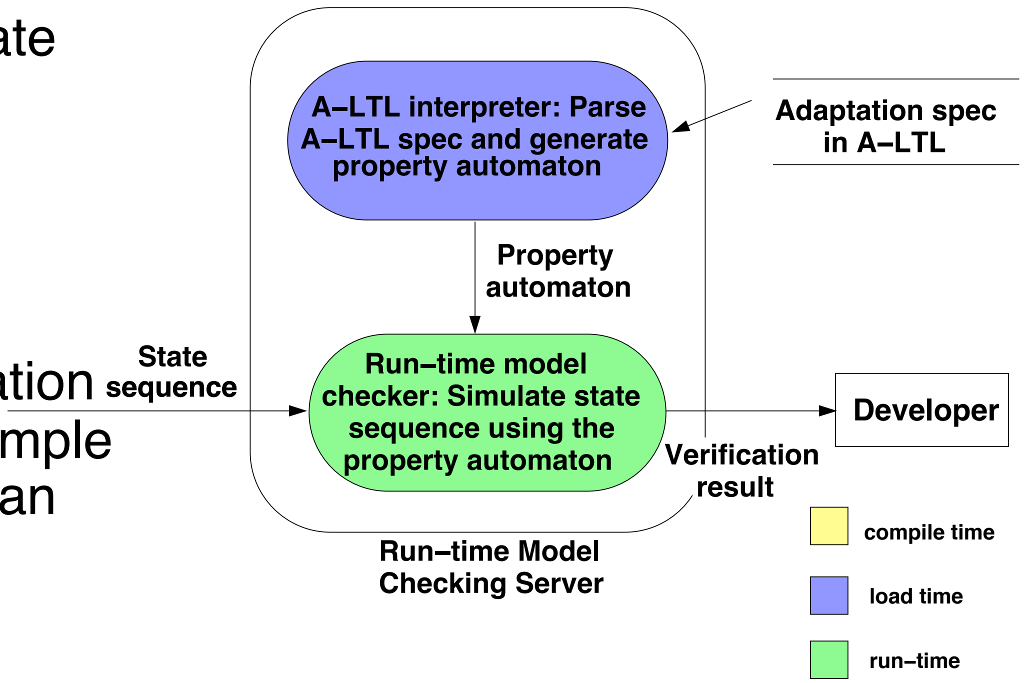
AMOebA-RT : Run-time Analysis

- Load time:
  - Construct property automaton from A-LTL specification
  - Property automata is a *run-time model*
    - Accepts all execution paths that satisfy the property
    - Each node specifies:
      - Property it satisfies
      - Property that must be satisfied by next states
    - Used to simulate adaptive system execution
      - Detect and report property violation



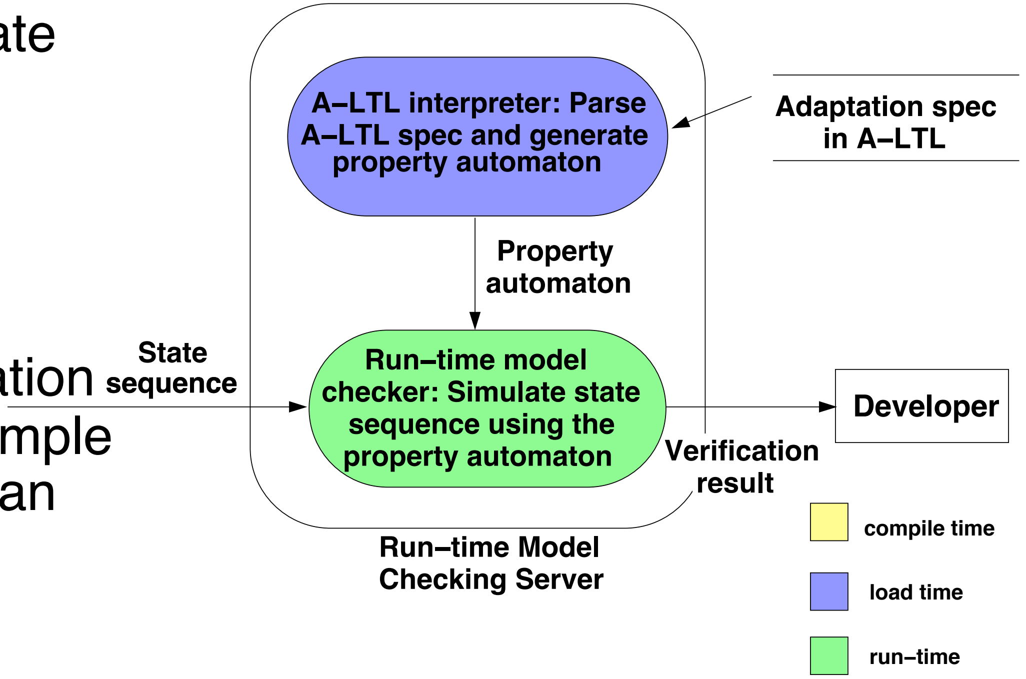
AMOebA-RT : Run-time Analysis

- Run-time:
  - Receives the run-time state information
  - Simulate execution on property automaton
- Return verification results:
  - Success if execution terminates without a violation
  - Failure and a counterexample if the simulation reaches an error state



AMOebA-RT : Run-time Analysis

- Run-time:
  - Receives the run-time state information
  - Simulate execution on property automaton
- Return verification results:
  - Success if execution terminates without a violation
  - Failure and a counterexample if the simulation reaches an error state



Avoids state explosion by exploring one execution path at a time.

Conclusions & Future Work

- AMOebA-RT
  - Run-time monitoring
    - Non-invasive instrumentation of source code
      - Separates run-time monitoring code from business logic.
  - Run-time model checking
    - Verifies adaptation properties specified using LTL and A-LTL
      - Load time: constructs automata representing LTL / A-LTL property
      - Run-time: checks execution path adheres LTL / A-LTL property
    - Avoids state explosion by verifying one execution path at a time
- Future Work
  - Applying AMOebA-RT to additional case studies
  - Use the counterexamples as decision maker input
  - Extend to isualize the run-time execution path

Thank you!

AMOebA-RT Run-Time Verification of Adaptive Software

Ji Zhang, Betty H.C. Cheng, and Heather J. Goldsby

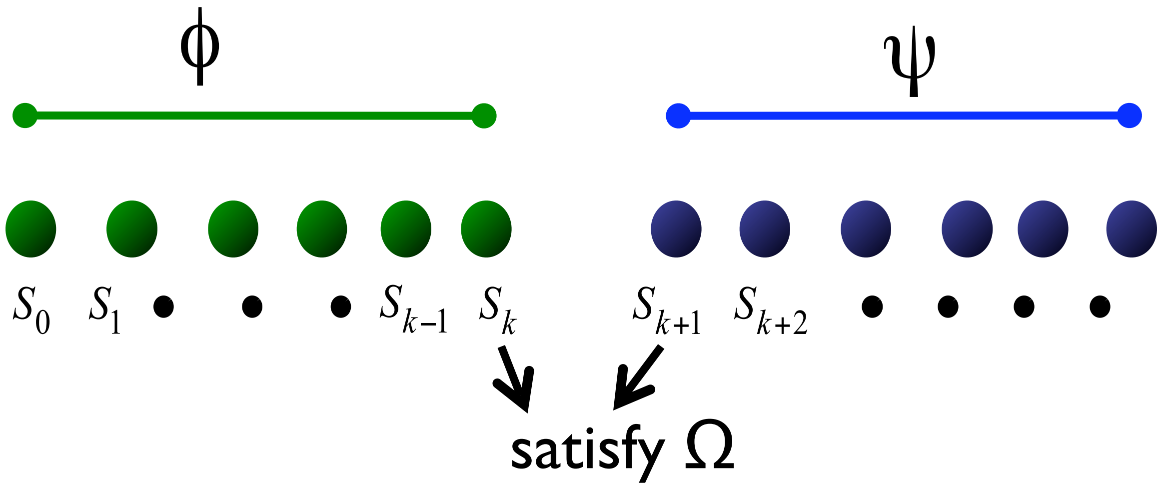
This work has been supported in part by NSF grants EIA-0000433, CNS-0551622, CCF-0541131, IIP-0700329, CCF-0750787, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Air Force Research Lab under subcontract MICH 06-S001-07-C1, Siemens Corporate Research, and a Quality Fund Program grant from Michigan State University.

Selected References

- Zhang, J., Cheng, B.H.C.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software (JSS), Architecting Dependable Systems **79**(10) (2006) 1361–1369
- Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: Proceedings of International Conference on Software Engineering (ICSE'06), Shanghai,China (2006)
- Zhang, J., Cheng, B.H.C.: Modular model checking of dynamically adaptive programs. Technical Report MSU-CSE-06-18, Computer Science and Engineering, Michigan State University, East Lansing, Michigan (2006) `http://www.cse.msu.edu/~ zhangji9/zhang06Modular.pdf`.

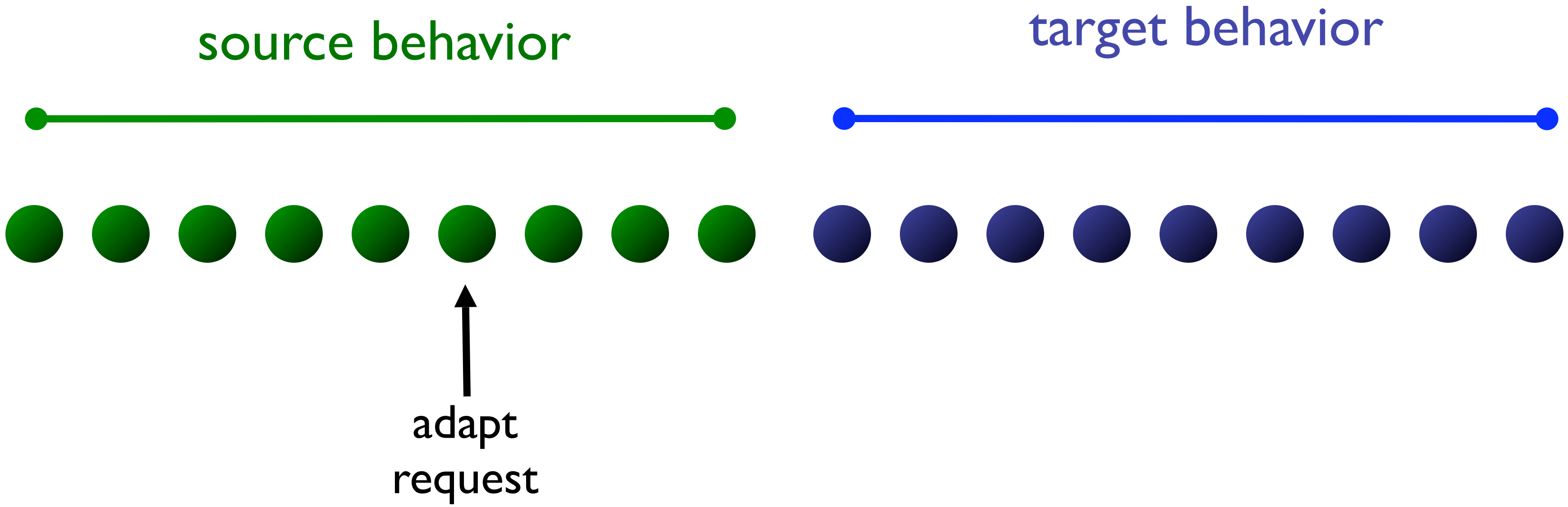
MICHIGAN STATE UNIVERSITY

A-LTL: Adapt Operator Extended LTL



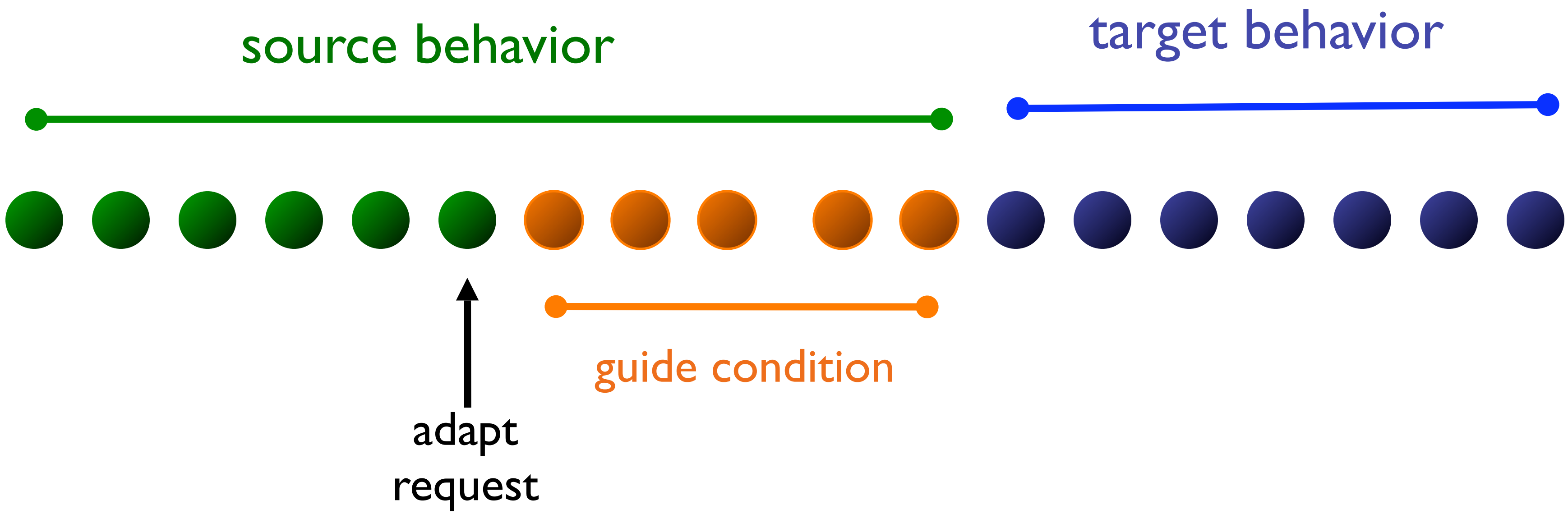
– Used to specify adaptation semantics.

One-Point Adaptation



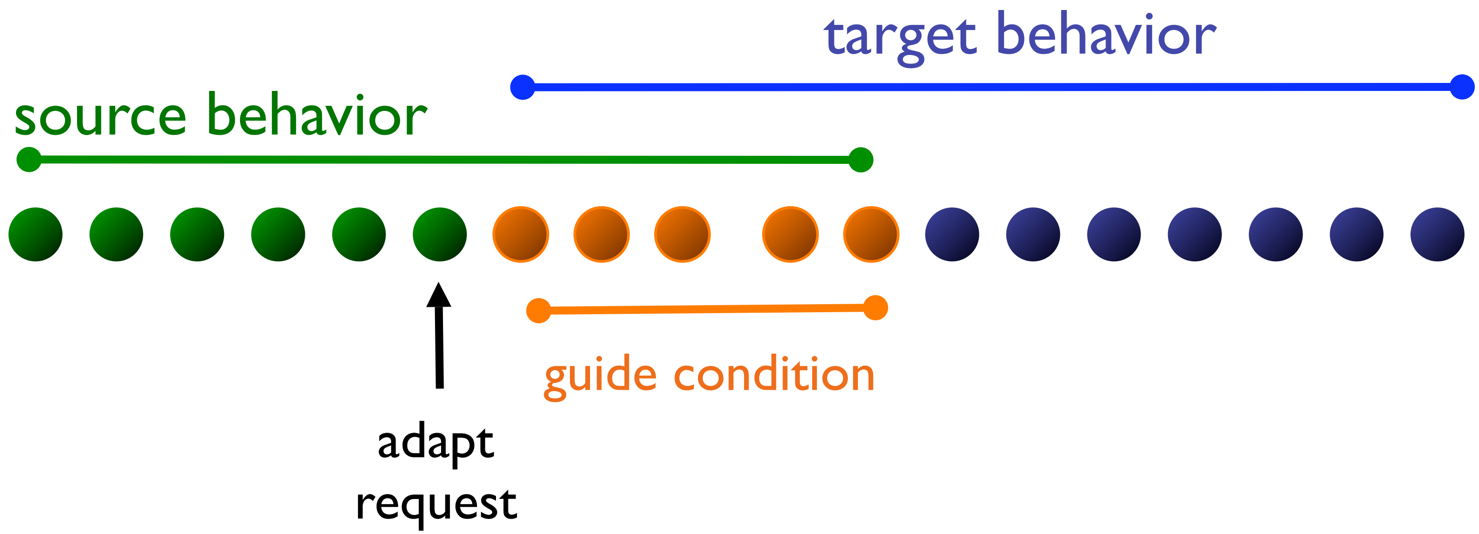
- Initially behaves as source.
- At one point after “adapt request”, starts to behave as **target.**
- Most common adaptation semantics.

Guided Adaptation



– Initially behaves as source.
– A condition guides the program to reach a **safe state**.
– Finally, the program behaves as target.
– Example: Used for hot-swapping techniques

Overlap Adaptation



- The source and target behavior may overlap.
- A condition guides the program to reach a safe state.
- Example: Adaptive Java pipeline