

# FORMAL METHODS OF SOFTWARE DEVELOPMENT: A BRIDGE TO ARTIFICIAL INTELLIGENCE THROUGH SPARK ADA

**BY NIKITA WAGLE & KAVITHA BHASKAR**

IN PARTIAL FULFILLMENT OF THE COURSE CSE 841 FORMAL  
METHODS OF SOFTWARE DEVELOPMENT  
INSTRUCTOR: LAURA K. DILLON



# INTRODUCTION

- SPARK Ada Approach in Formal Methods of Software Development to ensure high integrity using traditional symbol-oriented approach
- NuSPADE Approach as a viable solution to lack of program analysis and incapability to develop exception freedom proofs in SPADE Simplifier
- Analysis of the merits and demerits of Symbol-oriented Approach with Developmental Approach in Artificial Intelligence

# BACKGROUND

- Mental development of an artificial agent characterized by :
  - Symbol -oriented Approach
  - Developmental Approach
- Formal Methods of Software Development employ traditional symbol-oriented approach to program autonomous agents
- Developmental approach, a topic of current and future study in Artificial Intelligence

# FORMAL METHODS OF SOFTWARE DEVELOPMENT- SYMBOL-ORIENTED APPROACH

- Symbol-oriented Approach develop high integrity program using mathematical logic and formal reasoning.
- Symbol-oriented Approach categorized as :
  - Logic -based Approach
  - Rule- based Approach
- Merits of Rule -based Approach over Logic-based Approach

# SYMBOL-ORIENTED APPROACH - LOGIC-BASED

- **Format** of Logic-Based Approach :  
*For all  $X$ ,  $P(X)$*
- **Merits** of Logic-based Approach:
  - Clarity
  - Lack of ambiguity
  - Pre-existence of technical mathematical theories about logic.

# SYMBOL-ORIENTED APPROACH - LOGIC-BASED

- **Demerits** of Logic-based Approach:
  - *For all  $X$ ,  $P(X)$*  makes AI researches seek universal validity
  - Not applicable in real world as no guarantee that assumptions will always be correct
- **Reframing** of Logic-based Approach:

*“For anything  $X$  being considered in current context, the assertion  $P\{X\}$  is likely to be useful for achieving goals, provided we apply it with other heuristically appropriate inference methods.”*

# SYMBOL-ORIENTED APPROACH – RULE-BASED

- Rule-based represents each fragment of knowledge by *if-then* rule
- Whenever a description of current problem situation precisely matches rule's antecedent *if* condition, system performs action described by rule's *then* consequent
- If no antecedent condition matches, programmer adds another rule
- More successful in real world Artificial Intelligence environment

# SYMBOL-ORIENTED APPROACH- MERITS

- Program itself is not the sole executer of the task
- Human programmer who frames the logic and rules of program and the program itself act together as a joint task
- Human programmer is holistically aware central controller and artificial agent is task executer



# SYMBOL-ORIENTED APPROACH- DEMERITS

- **Skull closed**  
Holistically aware central controller
- **Task specific**  
Logic pertinent to particular function
- **Domain restricted**  
Accumulation of increasing rules make system work in an unexpected manner.



# FORMAL METHODS OF SOFTWARE DEVELOPMENT - SPARK ADA

- One of the most important techniques that can improve security in critical software
- Spark programming language, a subset of Ada
- Expressive enough for industrial applications but restrictive enough to support rigorous analysis throughout the development process
- Formal verification ensures that program satisfies its specifications

# SPARK ADA - SPADE

- Spade simplifier in Spark Ada successful to a certain extent in automating exception freedom proofs or getting rid of run-time errors
- Two problems associated with SPADE
  - Fails to prove certain classes of problems
  - Becomes necessary to strengthen program specification in order to complete proofs
- Need to minimize the amount of user interaction required in performing formal verification to make it useful to Artificial Intelligence

# SPARK ADA - SPADE

- Proof Planning is used to overcome drawbacks of SPADE and to increase the level of automated reasoning
- Proof Planning in Artificial Intelligence is a technique for guiding tactic-based theorem provers
- “Given a particular proof obligation, functions are used by planning system to automatically construct customized tactic”

# SPARK ADA- NuSPADE

NuSPADE combines proof planning theorem with program analysis

- *Proof Planning* to identify additional hypothesis to the bounds on variables
- *Program Analysis* to strengthen default loop invariant and introduce required hypothesis
- Proof tactics and revised run-time check verification condition suitable for execution within the *Proof Checker*

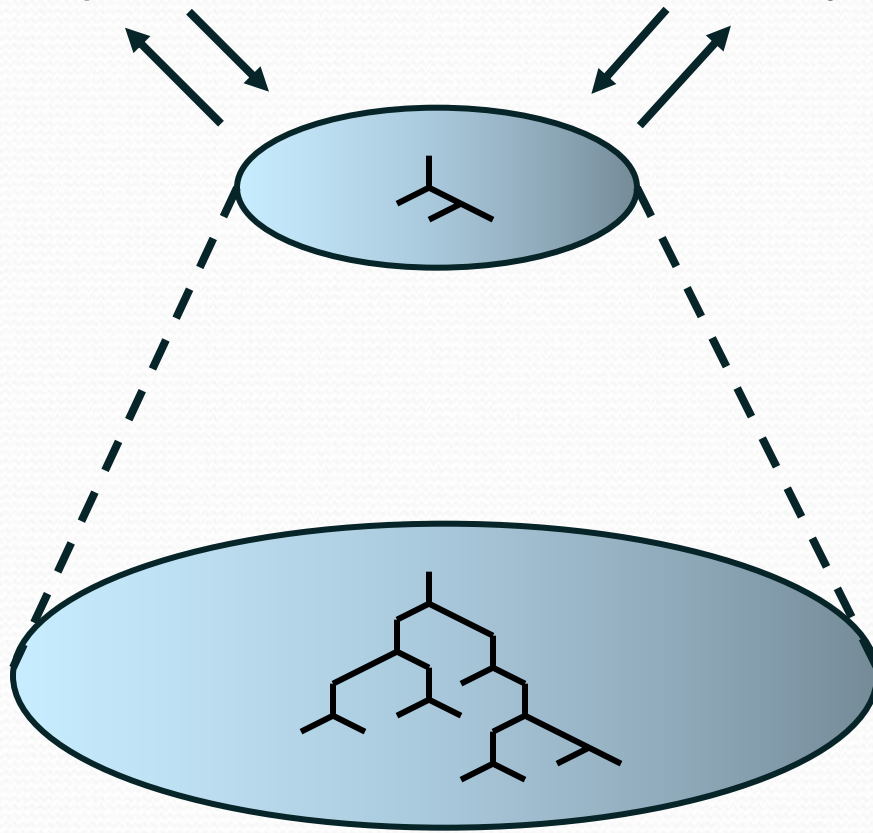
# NuSPADE – PROOF PLANNING

- Proof Plan and associated proof critiques prove that variable does not exceed its legal bounds
- Proof by transitivity identifies bounds of expression using decomposition
- Each failure of proof plan is identified by a patch using four new proof critiques
- Two proof critiques used to identify why verification condition is improvable
- Counter-example generated is used to search for auxiliary program property that progress search for a proof

# PROOF PLANNING

**Conjectures**

**Theory**



**Proof planning:**  
Methods + Critics

**Proof checking:**  
Tactics

# NuSPADE – PROGRAM ANALYSIS

- Program analysis automatically finds interesting properties about source code
- Program analysis in practice:
  - Flow analysis
  - Performance analysis
  - Discovering constraints on variables or abstract interpretation
  - Discovery code properties or invariant discovery

# NuSPADE – PROGRAM ANALYSIS

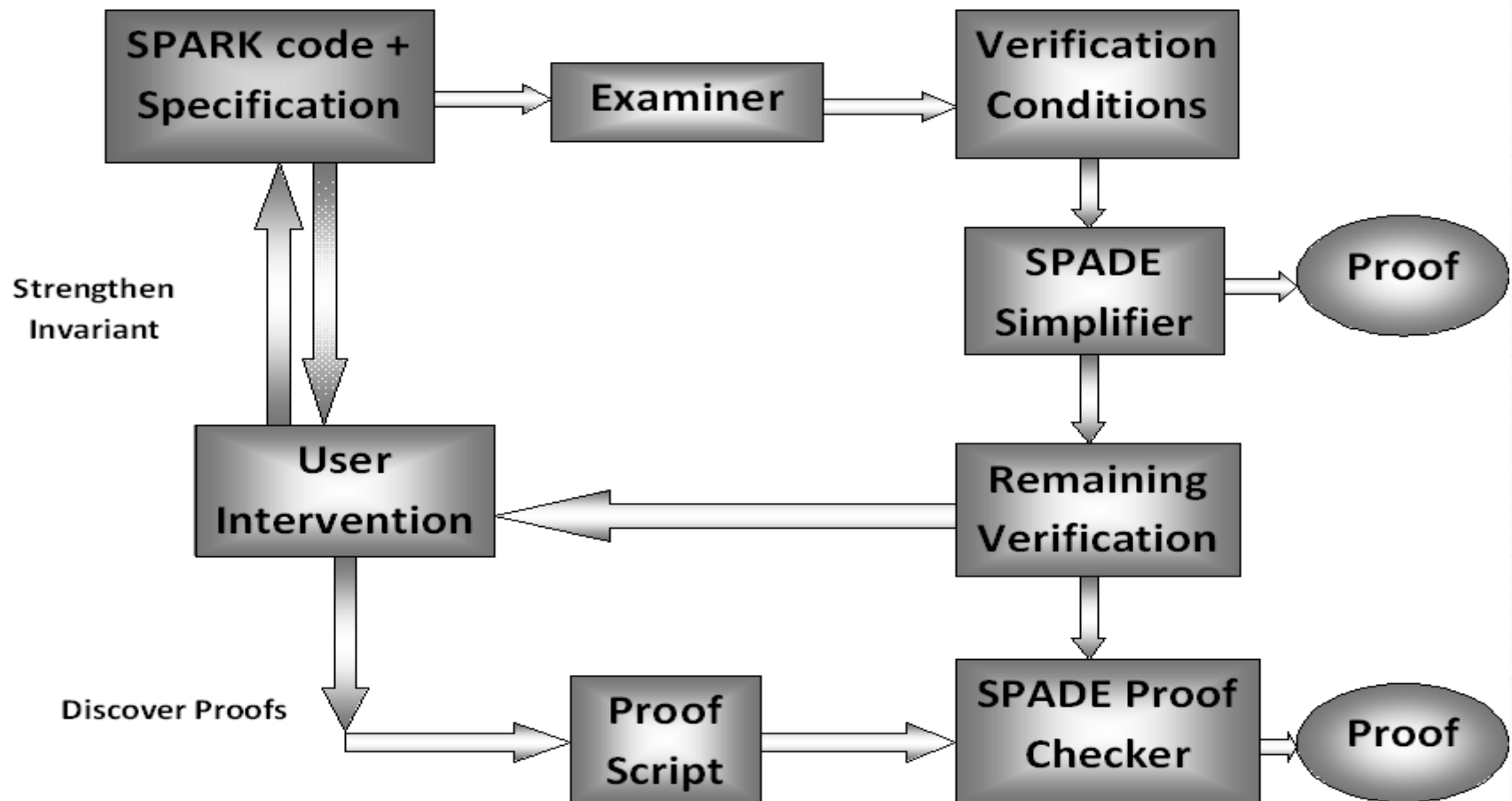
Program Analysis phase includes two systems :

- AutoGap supports two properties:
  - Generates simple loop invariants required to complete the proof of another
  - Generates information about program variables
  
- PropGen extends property generation properties of AutoGap as:
  - PropGen targets exception freedom proofs that SPADE simplifier fails in doing
  - PropGen translates SPARK source code to conventional flow graphs to generate properties that support exception freedom proof

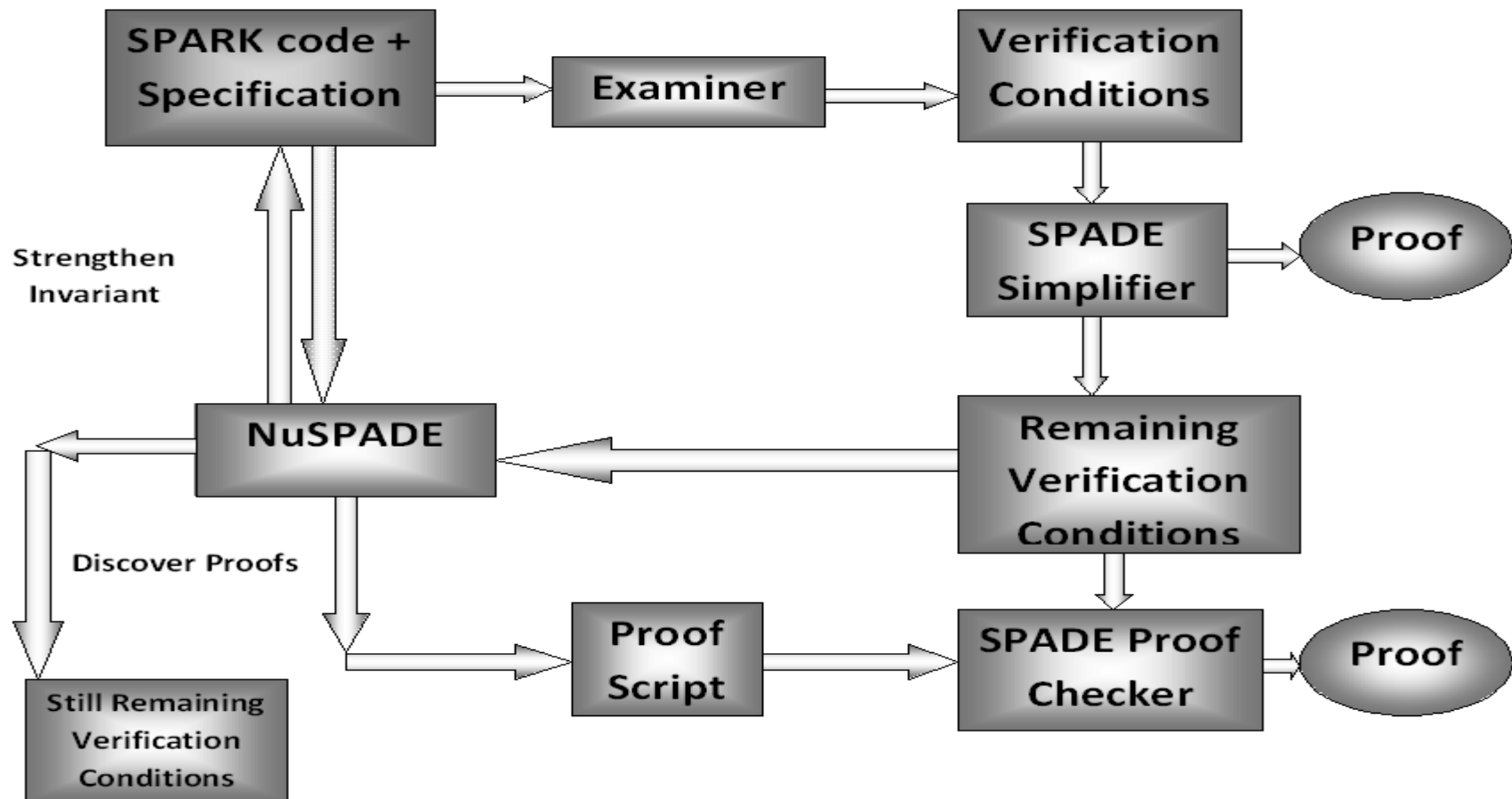
# NuSPADE – PROOF CHECKER

- Proof Checker does more than proof checking , it attempts to automate many small steps within development of a proof
- Automation of small steps within development of a proof are useful if proof is developed interactively
- If proof checker is used for its sole purpose, small steps complicate proof planning process

# SPARK ADA - SPADE



# SPARK ADA - NuSPADE



# NuSPADE- FUTURE WORK

- Strengthen recurrence relation solver and experiment with other external reasoners, e.g. inequality reasoning
- Explore program analysis as a basis for bug finding (ongoing)
- Explore program analysis as a basis for reducing VC complexity



# PROGRAM VERIFICATION – A BRIDGE TO ARTIFICIAL INTELLIGENCE

- Program verification involves mathematical logic and formal reasoning techniques to develop high integrity programs
- Program verification finds widespread application in Artificial Intelligence
- Program verification has turned its focus from full function-based integrity to property-based integrity
- Bridges the gap between program specification and program execution

# SYMBOL-ORIENTED APPROACH VERSUS DEVELOPMENTAL APPROACH

- Symbol-oriented Approach
  - Task specific
  - Can only perform clean tasks
  - Skull Open
  - Holistically aware central controller
- Developmental Approach
  - Non-task specific
  - Can also perform muddy tasks
  - Skull closed
  - No holistically aware central controller

# CONCLUSION

- Artificial Intelligence demands the use of several different representations
- Each particular data structure has its own virtues and deficiencies and none by itself is adequate
- Each representation has domains of competence and efficiency .  
Hence, one may work while another fails
- Combination of symbol-oriented as well as development approach necessary for successful autonomous development of artificial agent

# REFERENCES

- Marvin Minsky, “*Logical versus Analogical or Symbolic versus Connectionist or Neat versus Scruffy*”, AI Magazine Volume 12 Number 2 (1991) (© AAI)
- Andrew Ireland, “*Towards Increased Verification Automation for High Integrity Software Engineering*”, EPSRC Research Assistant Industrial Secondment, Final Report, Collaborative Training Account GR/T11289, November 21, 2005
- Andrew Ireland, “*Automatic Guidance for Formal Verification of High Integrity Ada*”, Final Report of EPSRC Grant GR/R24081
- Bill J. Ellis and Andrew Ireland, “*Automation for exception freedom proofs*”, School of Mathematical & Computer Sciences, Heriot-Watt University, Edinburgh, Scotland, UK

# REFERENCES

- Juyang Weng, “*Task Muddiness, Intelligence Metrics, and Necessity of Autonomous Mental Development*”, *Minds and Machines* (2009) 19:93-115, DOI 10.1007/s11023-008-9127-1
- “*An Integration of Program Analysis and Automated Theorem Proving*” presented at the fourth international conference on integrated formal methods, Canterbury, 2004, [www.macs.hw.ac.uk/nuspade/presentations](http://www.macs.hw.ac.uk/nuspade/presentations)