Interactive fixes for software configuration

Yingfei Xiong, Peking University

Cooperation with:

Peking University	Bo Wang, Hansheng Zhang, Jie Wang, Haiyan Zhao, Wei Zhang
University of Waterloo	Leonardo Passos, Steven She, Krzysztof Czarnecki
University of Namur	Arnaud Hubaux

2013

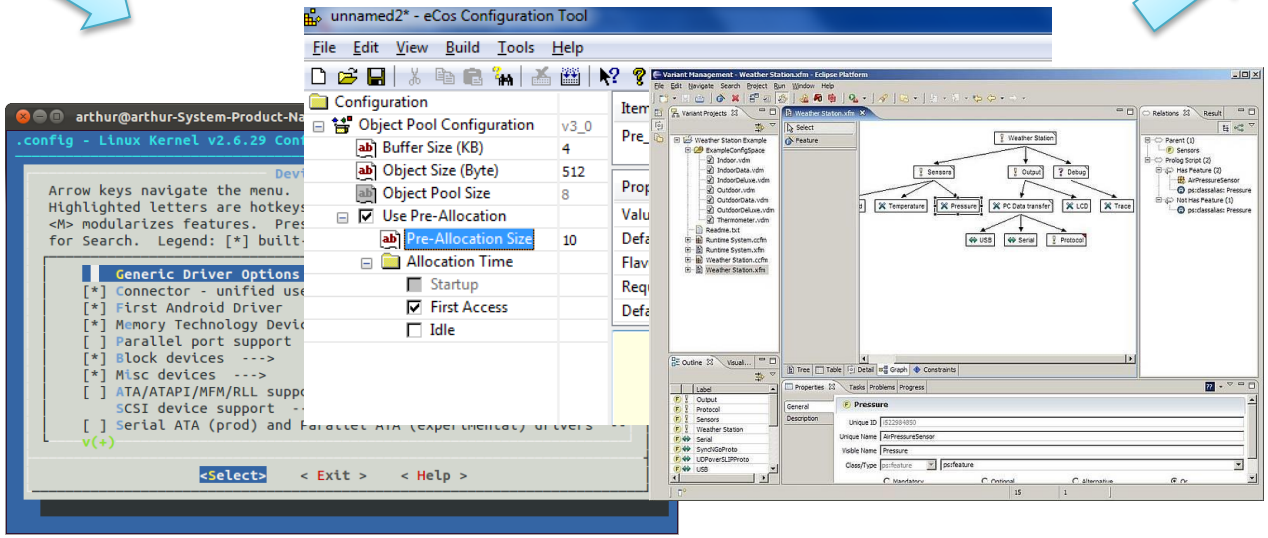
北京大学软件工程研究所

- 国内最早开展软件工程研究、规模最大、最有影响力的软件工程研究团队
- 院士三名（含双聘一名），博士生导师10名，硕士生导师13名
- 在软件工程顶级会议发表论文数占大陆总数约三分之二*
- 获得ACM SIGSOFT杰出论文奖三次，大陆共获奖四次，香港共获奖一次

*统计截止2011年

Variability Models & Configurators

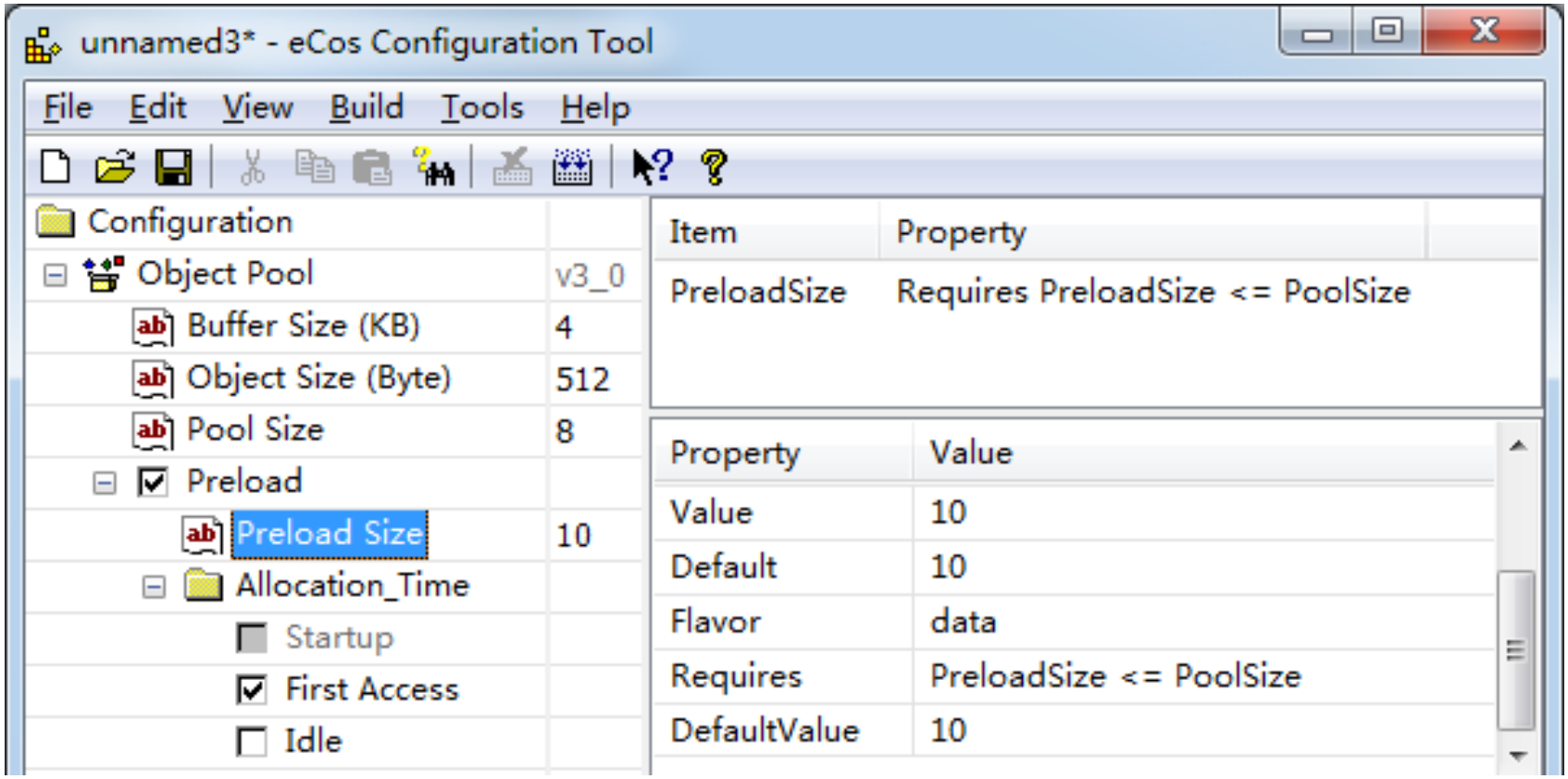
Configuration



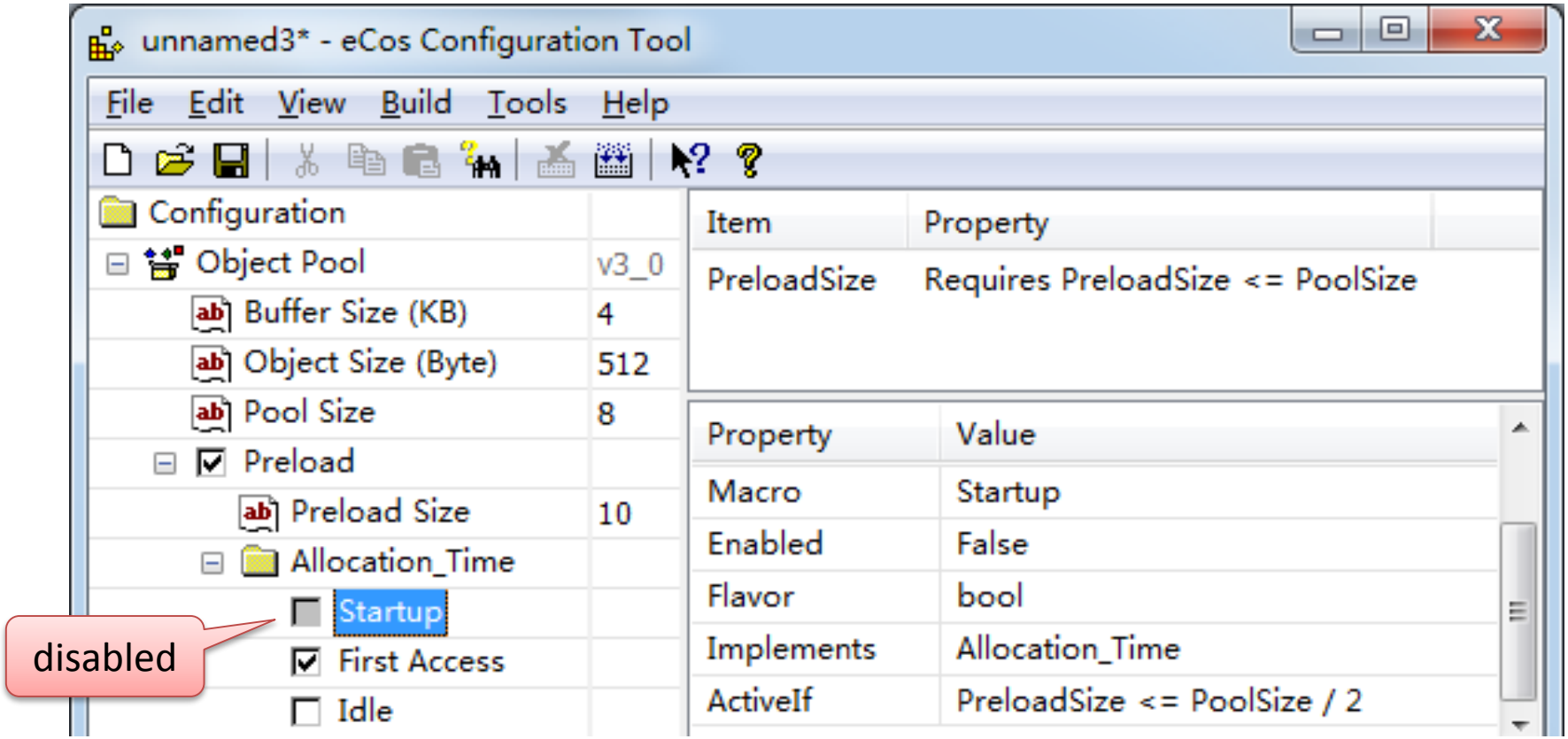
Linux Kconfig,
eCos CDL,
pure::variants,
…

Variability Models

eCos Configurator - Errors



eCos Configurator - Inactive Options

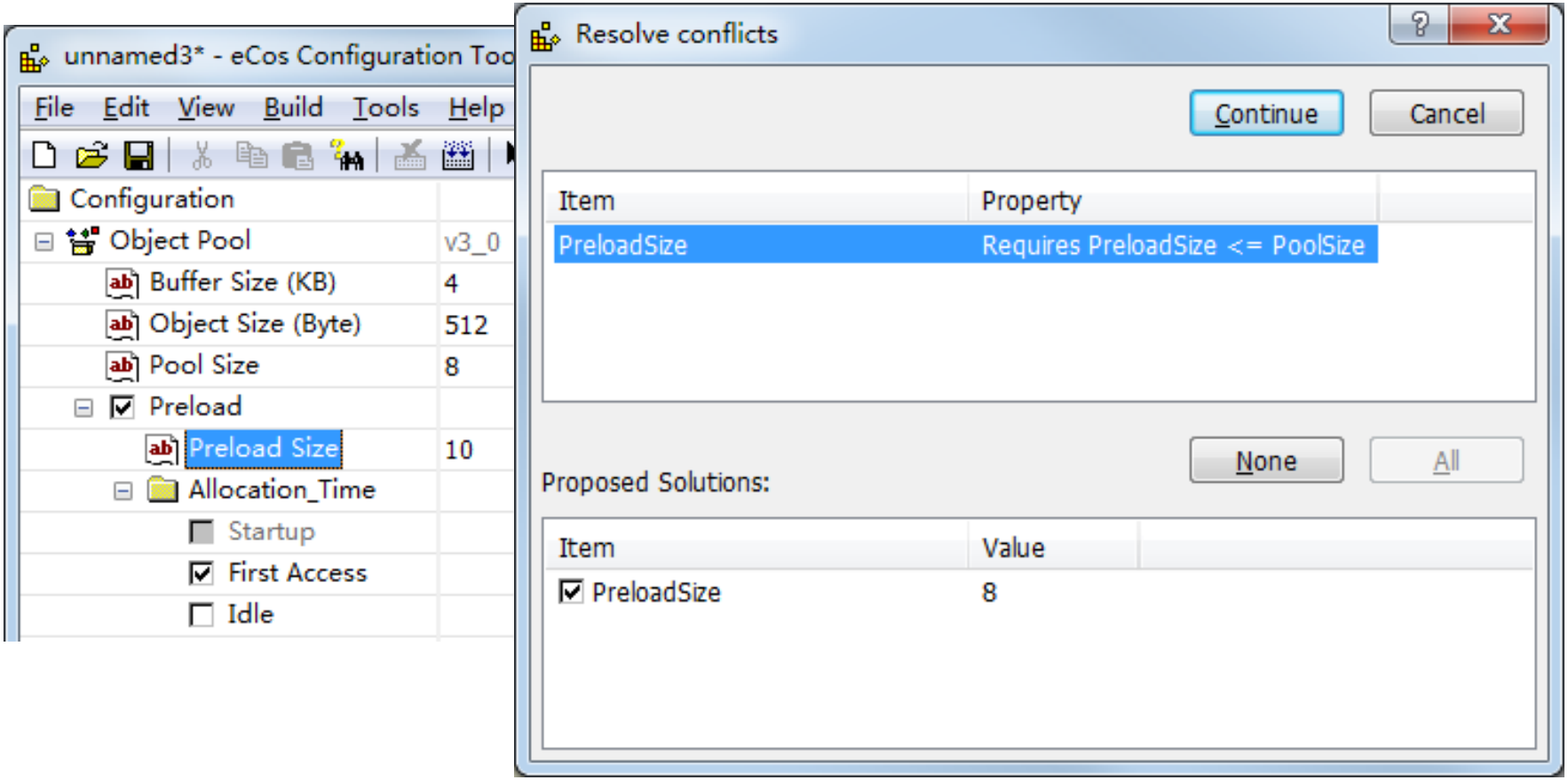


Error resolution and option activation both need to resolve violation of constraint.

Survey

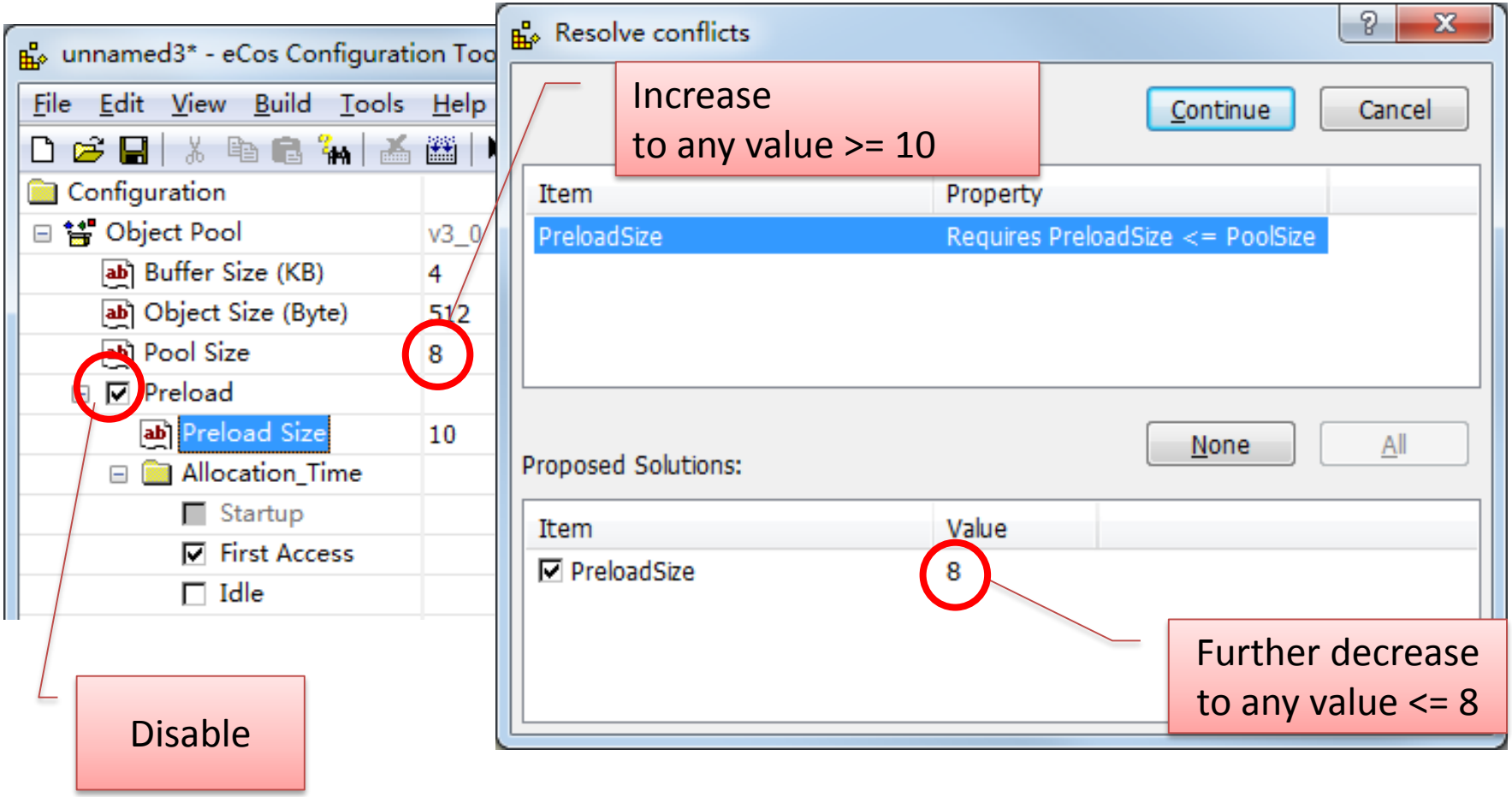
- 97 Linux users and 9 eCos users
- Resolving a violation is hard
  - 20% Linux users need "a few dozen minutes" to activate an option in average
  - 56% eCos users consider activation to be a problem

eCos Configurator



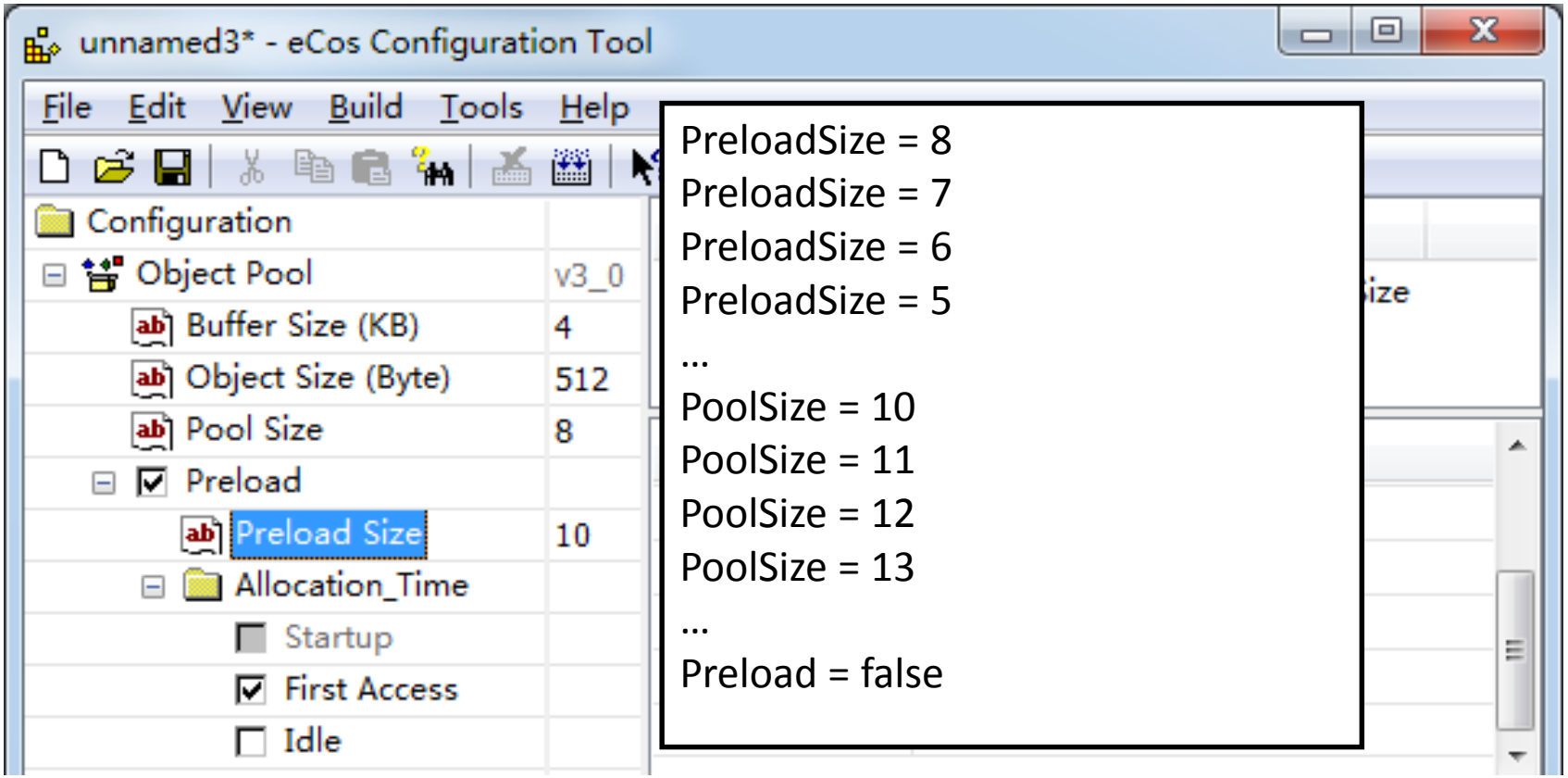
Essentially, fixes work for both resolving errors and activating options

Fix Incompleteness



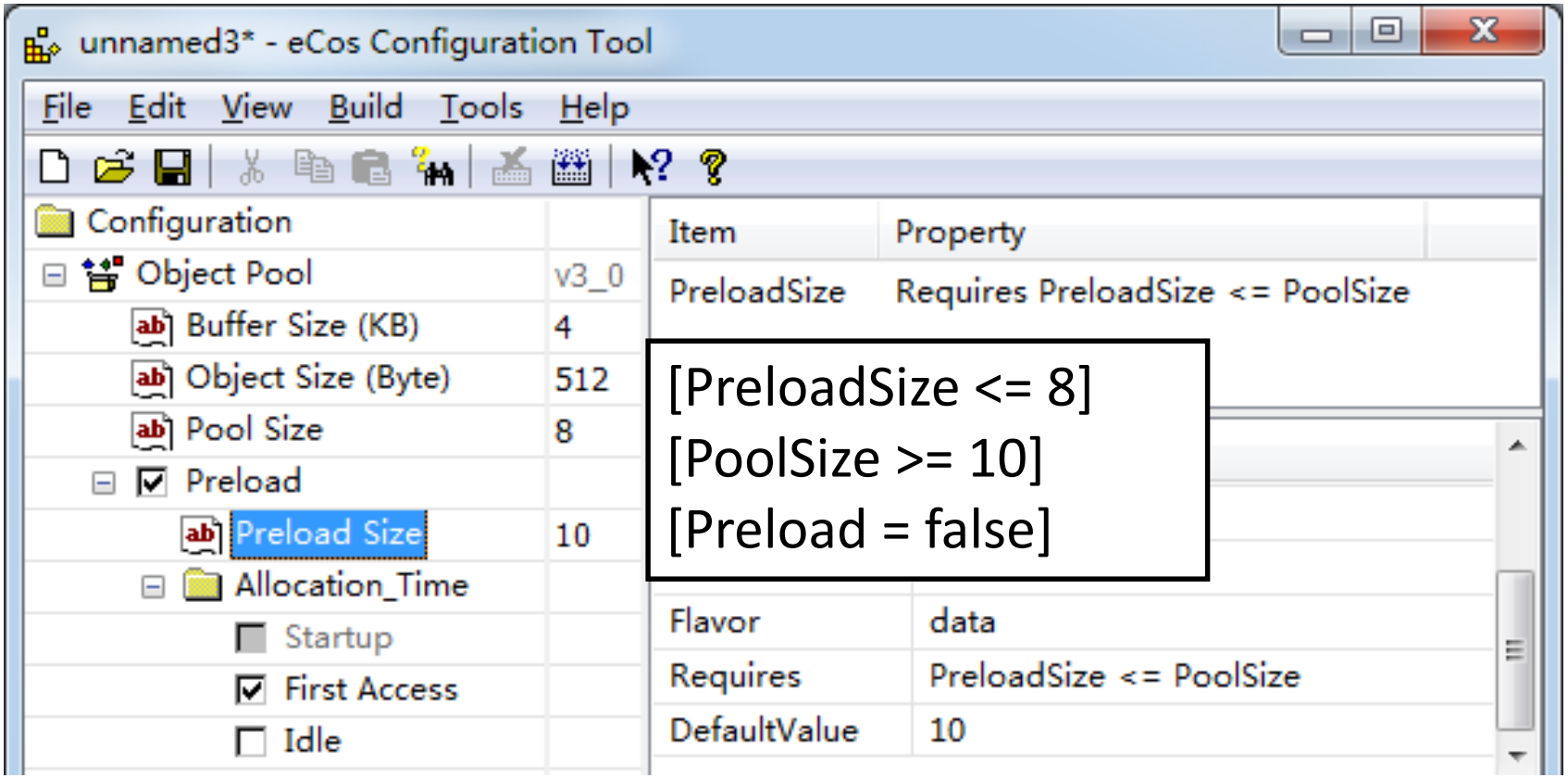
78% eCos users have ecountered situations where the proposed fix is not useful

How to complete fixes



PreloadSize = 8
PreloadSize = 7
PreloadSize = 6
PreloadSize = 5
…
PoolSize = 10
PoolSize = 11
PoolSize = 12
PoolSize = 13
…
Preload = false

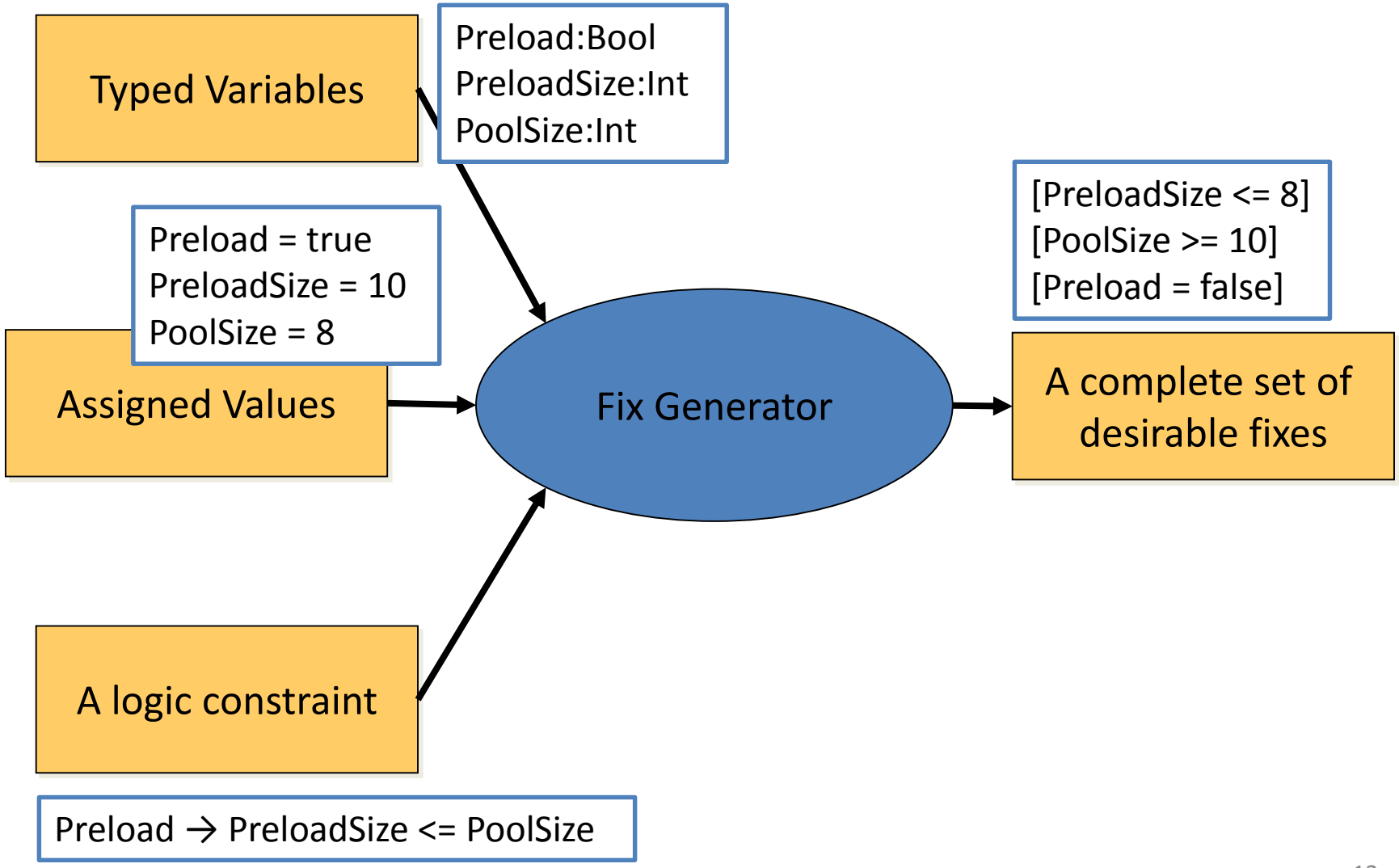
Our Solution – Range Fixes



Our Contributions

- Defining the range fix generation problem
  - Three desirable properties of range fixes
- Proposing a range fix generation algorithm
- Exploring the constraint interaction problem
  - Summarizing and adapting three strategies used in existing work
  - Comparing the strategies empirically

Fix Generation Problem – a General Definition



Desired Properties of Fixes

Correctness	Minimality of variables	Maximality of ranges
Any change represented by a range fix will satisfy the constraint	There is no way to change a subset of variables to satisfy the constraint	A range fix represents the maximal ranges over the variables
A desirable one: [PreloadSize <=8]		
Undesirable ones		
[PreloadSize <= 9]	[PreloadSize <=8, Preload = false]	[PreloadSize <=7]

Algorithm Outline

- Step 1: find the variables to change
  - Basic idea: translating to an SMT problem
    - ① treat configurations also as soft constraints
      1. [soft] Preload = true
      2. [soft] PreloadSize = 10
      3. [soft] PoolSize = 8
      4. [hard] Preload → PreloadSize <= PoolSize
    - ② ask an SMT solver for unsatisfiable cores
      - (1, 2, 3)
    - ③ pick one variable from each core
      - {Preload}, {PreloadSize}, {PoolSize}

Algorithm Outline

- Step 2: find the range of the variables
  - Basic idea: simplify the constraint
    - Example: {PreloadSize}

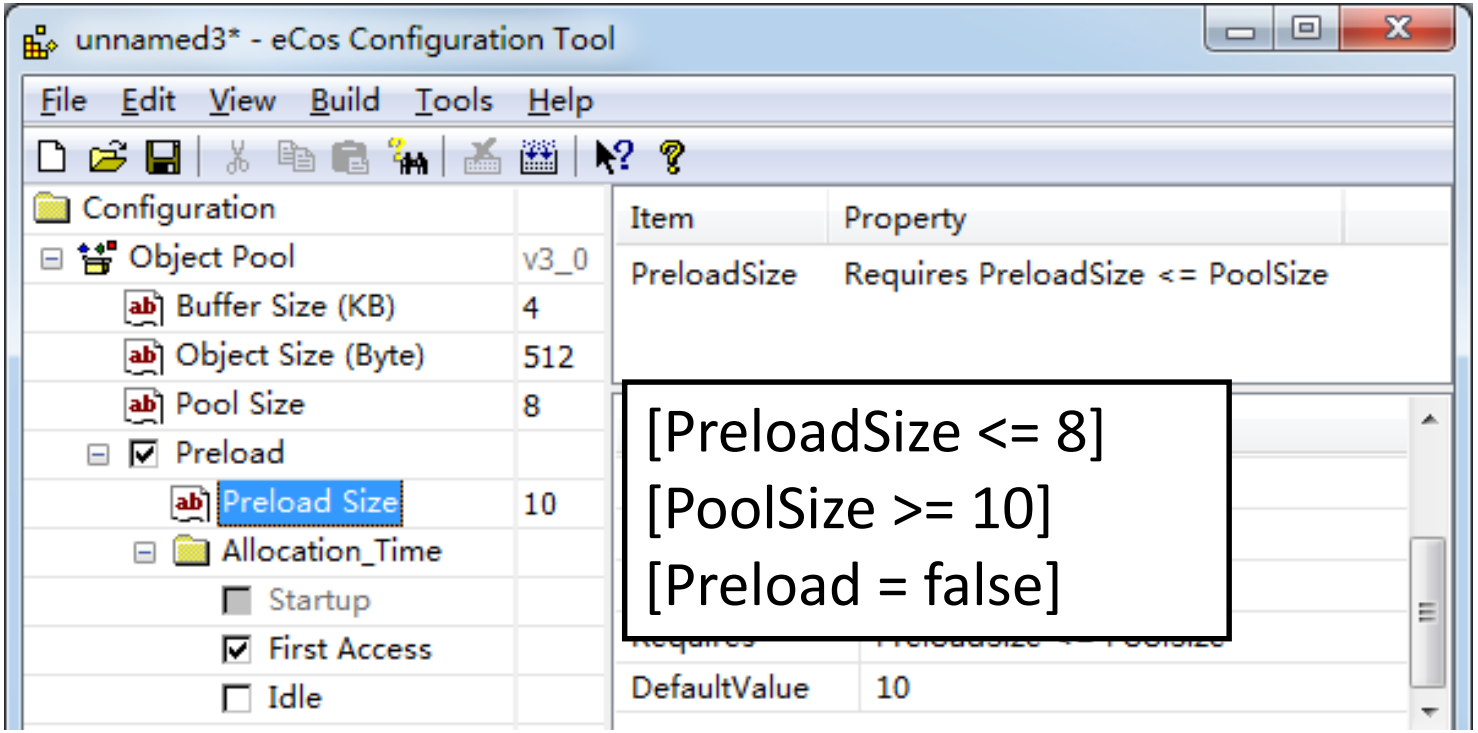
① replace unchangeable variables with their current values

- true → PreloadSize <= 8

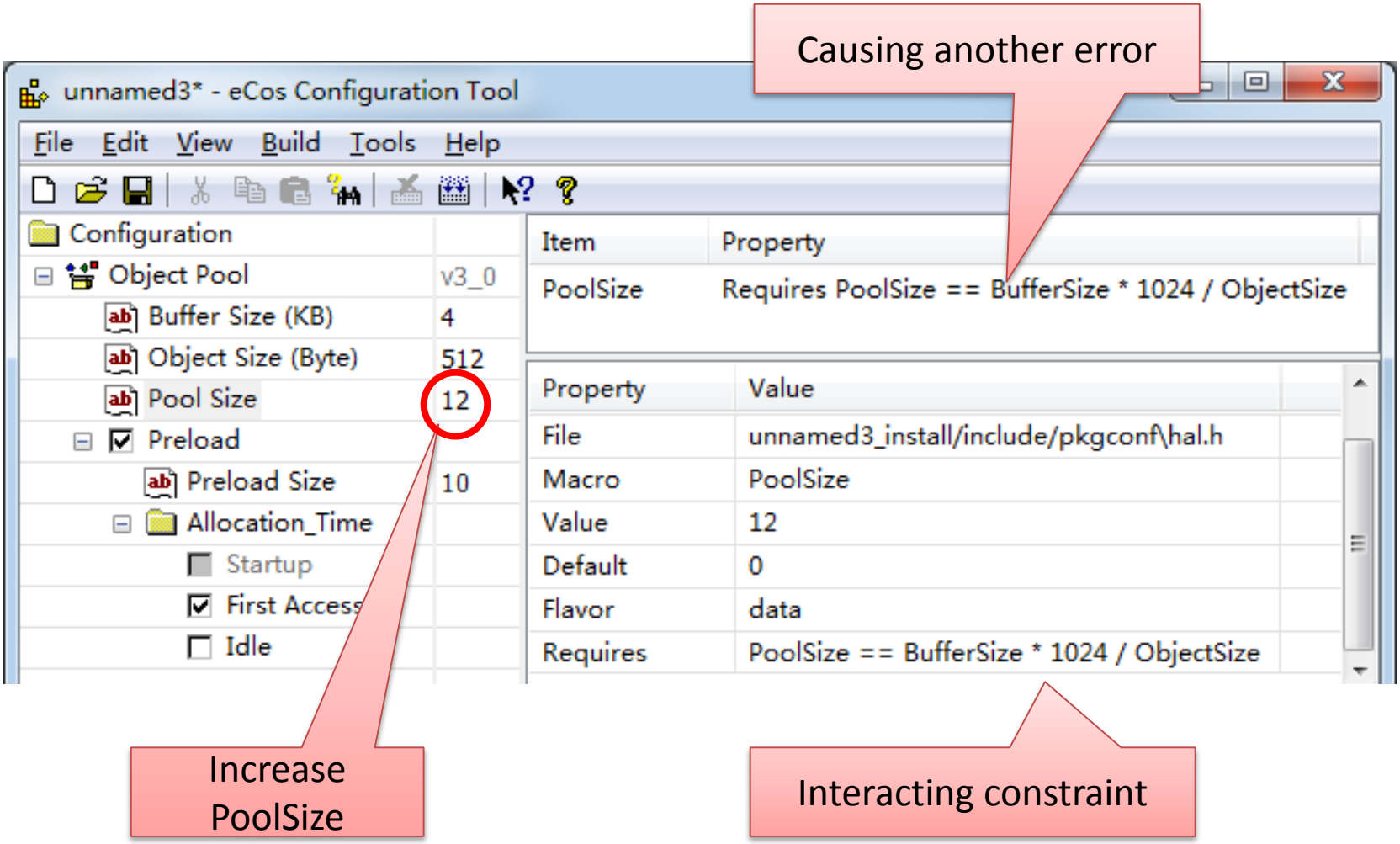
② simplify the constraint and convert to CNF

- [PreloadSize <= 8]

Constraint Interaction

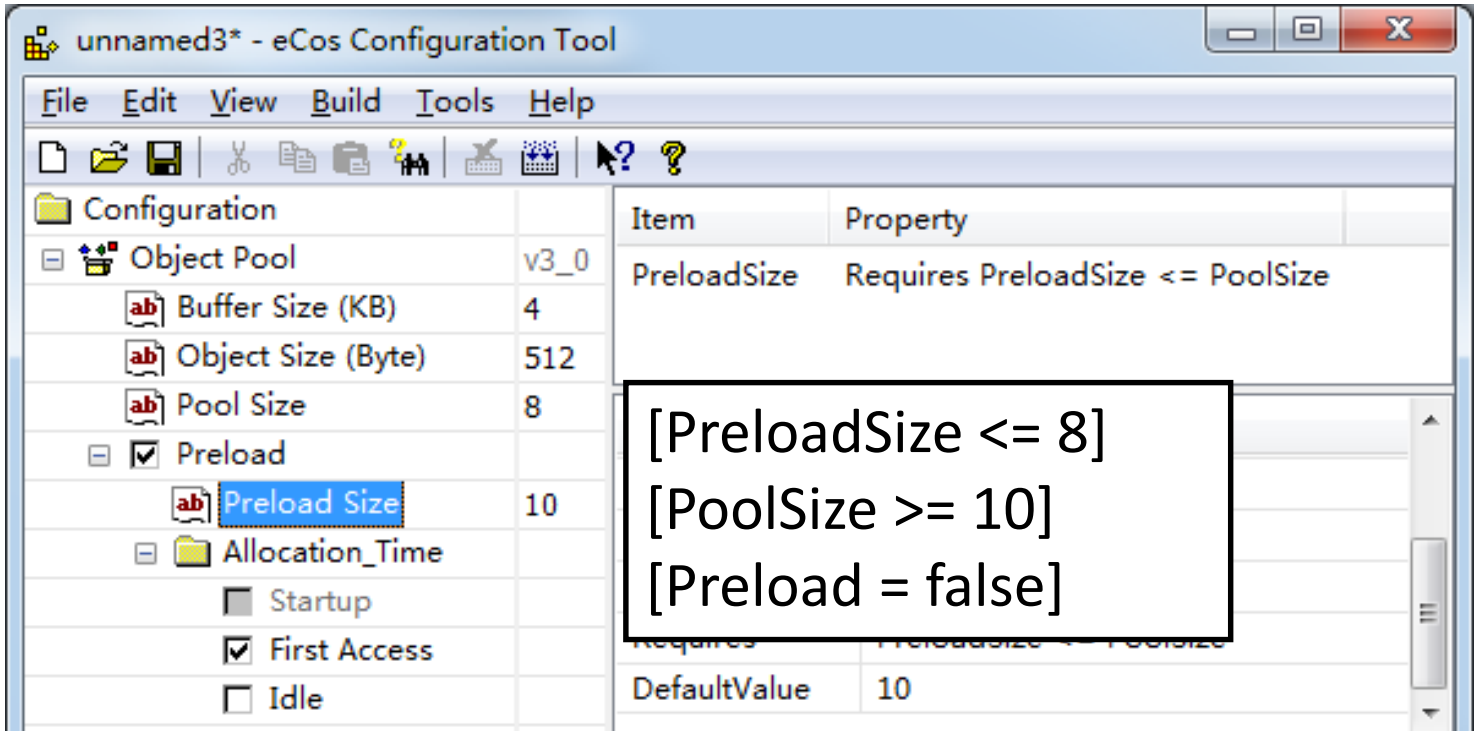


Constraint Interaction



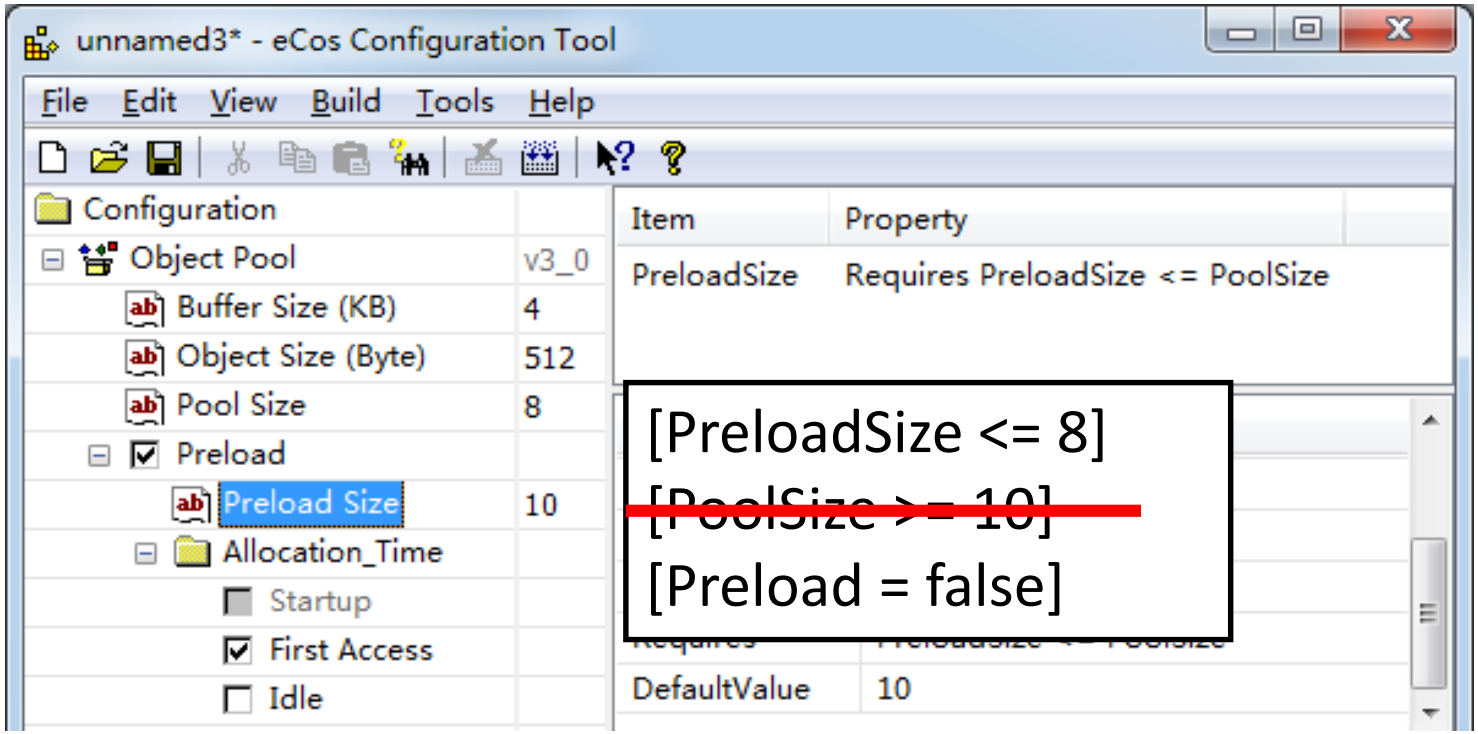
Ignorance

Ignore the interaction



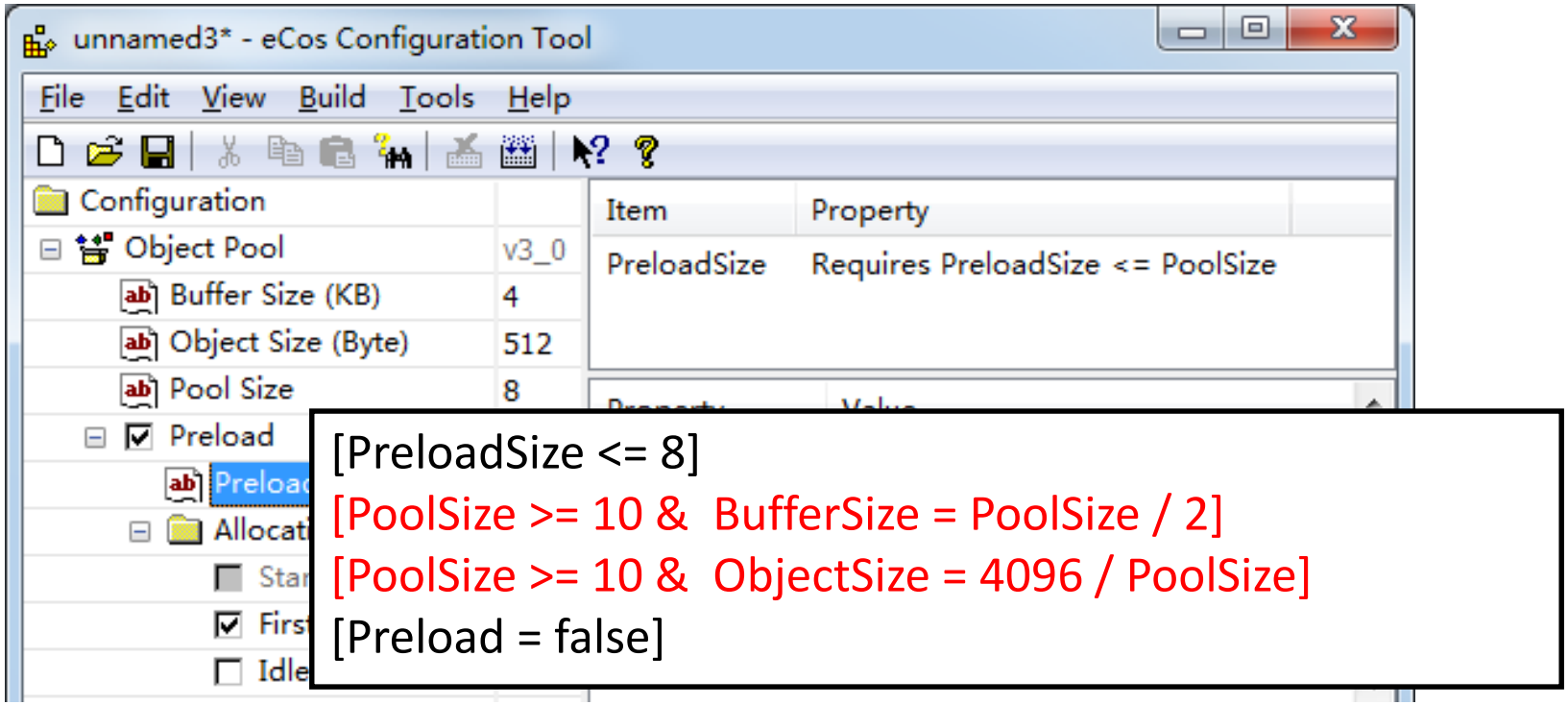
Elimination

Eliminate all changes that will violate other constraints



Propagation

Propagate the change along other constraints



Translating to the basic case

- Assignments: Preload = true, PreloadSize = 10, PoolSize = 8, BufferSize = 4, ObjectSize = 512
- Constraints:
  - **Preload → PreloadSize <= PoolSize**
  - PoolSize == BufferSize * 1024 / ObjectSize
- Ignorance:
  - Preload → PreloadSize <= PoolSize
- Elimination:
  - Preload -> PreloadSize <= PoolSize /\ PoolSize == 4 * 1024 / 512
- Propagation:
  - Preload → PreloadSize <= PoolSize /\ PoolSize == BufferSize * 1024 / ObjectSize

Comparison of Strategies

	Ignorance	Elimination	Propagtion
Execution time	Shortest	Short	Possbily long
Complexity of fix lists	Simple	Simplest	Possibly complex
Introduction of new errors	Possible	Never	Never
Fix completeness	Complete (for one constraint)	Incomplete	Complete (for all constraints)

Experiments

- Source
  - Version histories from 5 open source projects
- Steps
  - Compare each pair of consecutive versions
  - Replay the user changes in different orders
  - Generate fixes for the violations and compare with user changes

Execution Time

	Ignorance	Elimination	Propagtion
Execution time	Average: 17ms Maximum: 20ms	Average: 20ms Maximum: 30ms	Average: 50ms Maximum: 250ms
Complexity of fix lists	Simple	Simplest	Possibly complex
Introduction of new errors	Possible	Never	Never
Fix completeness	Complete (for one constraint)	Incomplete	Complete (for all constraints)

Our algorithm is sufficiently fast for each strategy

Complexity of fix lists

	Ignorance	Elimination	Propagtion
Execution time	Average: 17ms Maximum: 20ms	Average: 20ms Maximum: 30ms	Average: 50ms Maximum: 250ms
Complexity of fix lists (Number of variables in a list)	Max: 4 Median: 2 Average: 2.2	Max: 4 Median: 2 Average: 1.64	Max: 58 Median: 2 Average: 8.0
Introduction of new errors	Possible	Never	Never
Fix completeness	Complete (for one constraint)	Incomplete	Complete (for all constraints)

In propagation, 83% of the fix lists contain less than 10 variables

Introduction of new errors

	Ignorance	Elimination	Propagtion
Execution time	Average: 17ms Maximum: 20ms	Average: 20ms Maximum: 30ms	Average: 50ms Maximum: 250ms
Complexity of fix lists (Number of variables in a list)	Max: 4 Median: 2 Average: 2.2	Max: 4 Median: 2 Average: 1.64	Max: 58 Median: 2 Average: 8.0
Introduction of new errors	44% of all violations	Never	Never
Fix completeness	Complete (for one constraint)	Incomplete	Complete (for all constraints)

Fix completeness

	Ignorance	Elimination	Propagtion
Execution time	Average: 17ms Maximum: 20ms	Average: 20ms Maximum: 30ms	Average: 50ms Maximum: 250ms
Complexity of fix lists (Number of variables in a list)	Max: 4 Median: 2 Average: 2.2	Max: 4 Median: 2 Average: 1.64	Max: 58 Median: 2 Average: 8.0
Introduction of new errors	44% of all violations	Never	Never
Fix completeness (coverage of user changes)	100%	57%	100%

eCos configurator: 73%

Problem: Large Fixes

	Ignorance	Elimination	Propagtion
Execution time	Average: 17ms Maximum: 20ms	Average: 20ms Maximum: 30ms	Average: 50ms Maximum: 250ms
Complexity of fix lists (Number of variables in a list)	Max: 4 Median: 2 Average: 2.2	Max: 4 Median: 2 Average: 1.64	Max: 58 Median: 2 Average: 8.0
Introduction of new errors	Possible	Never	Never
Fix completeness	Complete (for one constraint)	Incomplete	Complete (for all constraints)

In propagation, 83% of the fix lists contain less than 10 variables

How to guide the users to identify their desirable fixes?

Our Solution

- Use the idea of priority
  - The priority of a variable represents the likelihood of its current value being desirable to the user.

- Two Basic ideas:
  - Generate fixes that only change variables with lower priorities
  - Dynamically adjust the priority of variables through implicit translation of user feedback

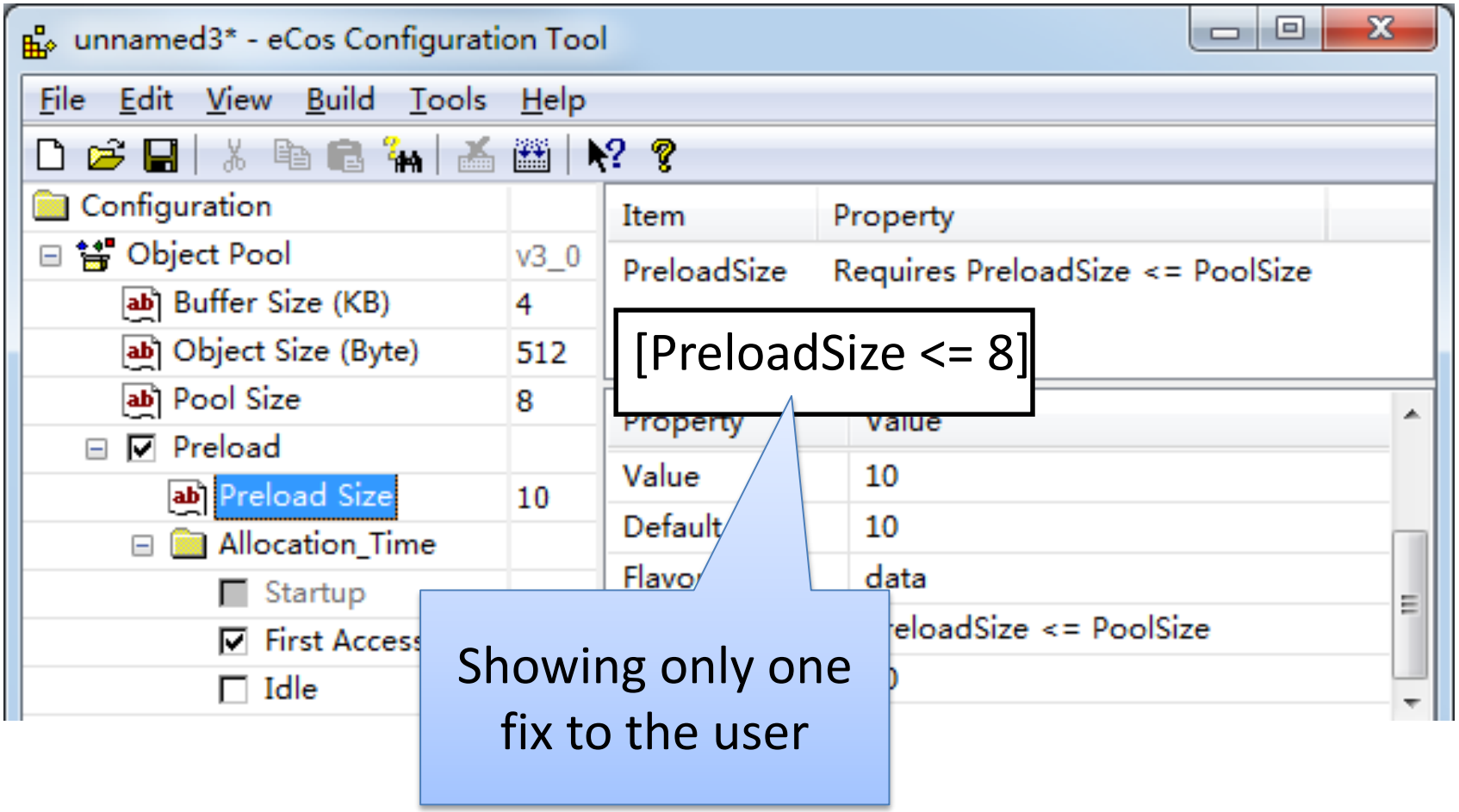
Our Contribution

- A priority-based approach to locating a desirable fix through user feedbacks
- An algorithm to implement the approach using any fix generation algorithm
- An empirical evaluation that shows the overall reduction of choices exposed to the user

Our Contribution

- A priority-based approach to locating a desirable fix through user feedbacks
- An algorithm to implement the approach using any fix generation algorithm
- An empirical evaluation that shows the overall reduction of choices exposed to the user

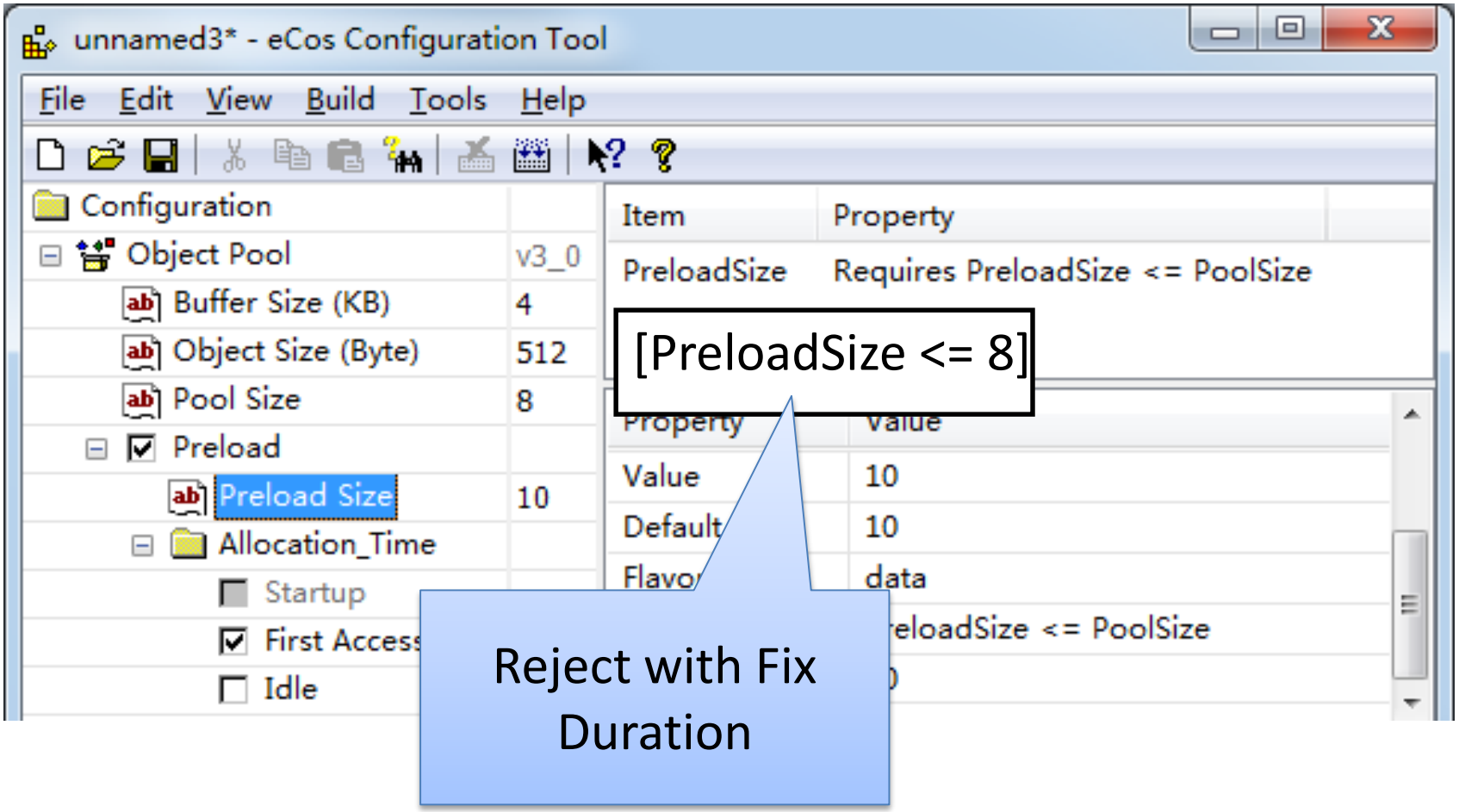
Our Approach



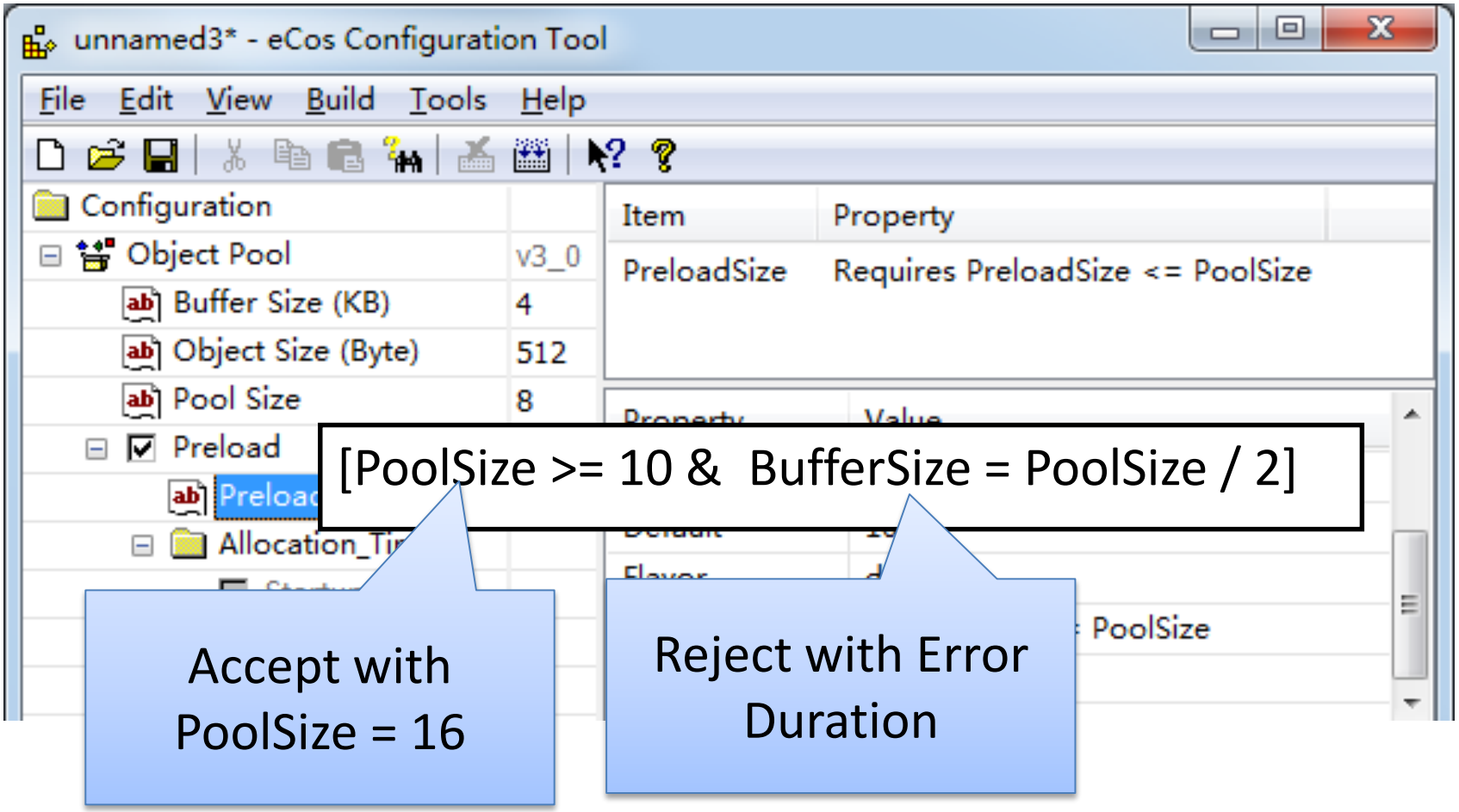
Provide feedback for each variable

- Accept the change (and pick a value)
- Reject the change
  - *Fix* duration
    - Current range is incorrect, future fixes can propose changes for this variable
  - *Error* duration
    - Current value is correct when fixing this error
  - *Permanent* duration
    - Current value is correct in the whole configuration process

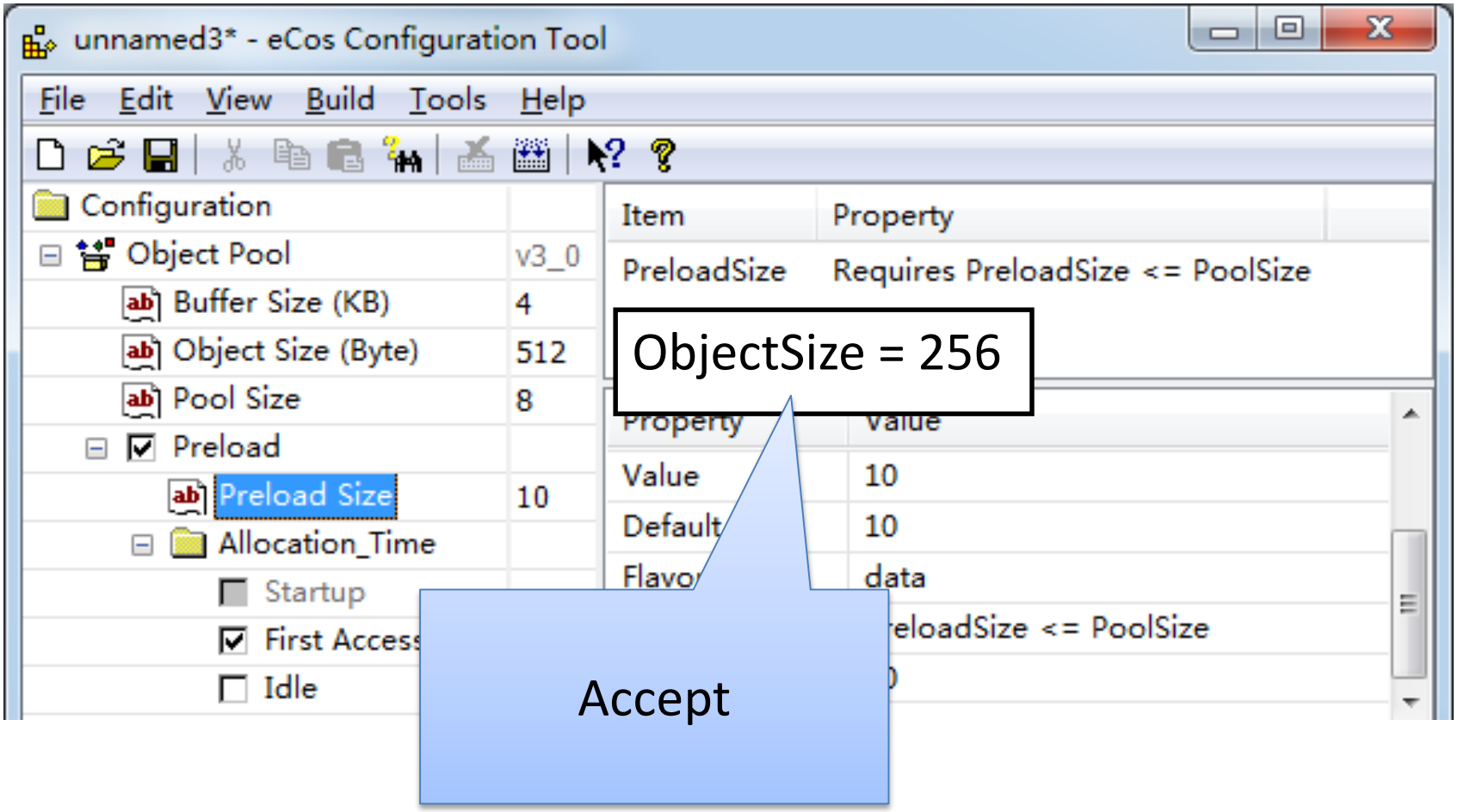
Our Approach



Our Approach



Our Approach



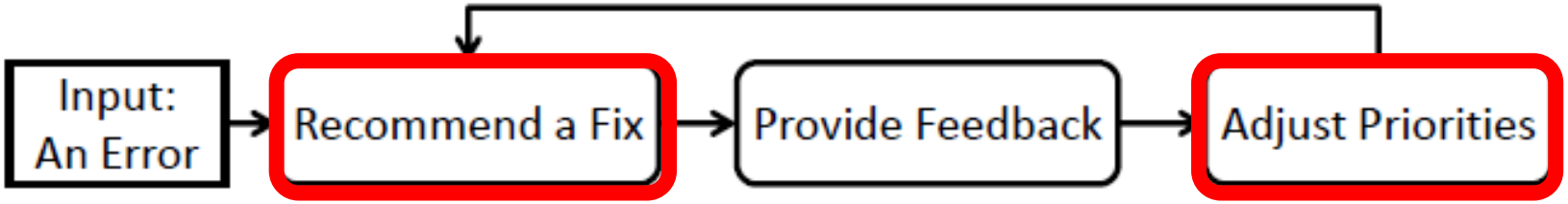
The user feedbacks are stored so that later fixes will be smarter.

Our Contribution

- A priority-based approach to locating a desirable fix through user feedbacks
- An algorithm to implement the approach using any fix generation algorithm
- An empirical evaluation that shows the overall reduction of choices exposed to the user

Algorithm Overview

Each variable is assigned a priority, initially zero.



Recommend a fix

- Use a threshold to confine the fix generation scope
  - Variables are changeable only when priority <= threshold.
  - Constraint [variable = current_value] is added for variables whose priority > threshold

Recommend a fix

- Initial threshold for an error = 1
- Invoke the fix generator
  - Randomly pick one fix from the generated fix list
  - Threshold += 1 if no fix is generated

Adjust Priorities

- New value is assigned
  - priority = 0
- Reject with *Fix* duration
  - priority +=1
- *Reject with Error* duration
  - priority binds to <threshold> +1
  - will be updated when threshold increases
- *Reject with Permanent* duration
  - priority = <max>

Handling No fixes

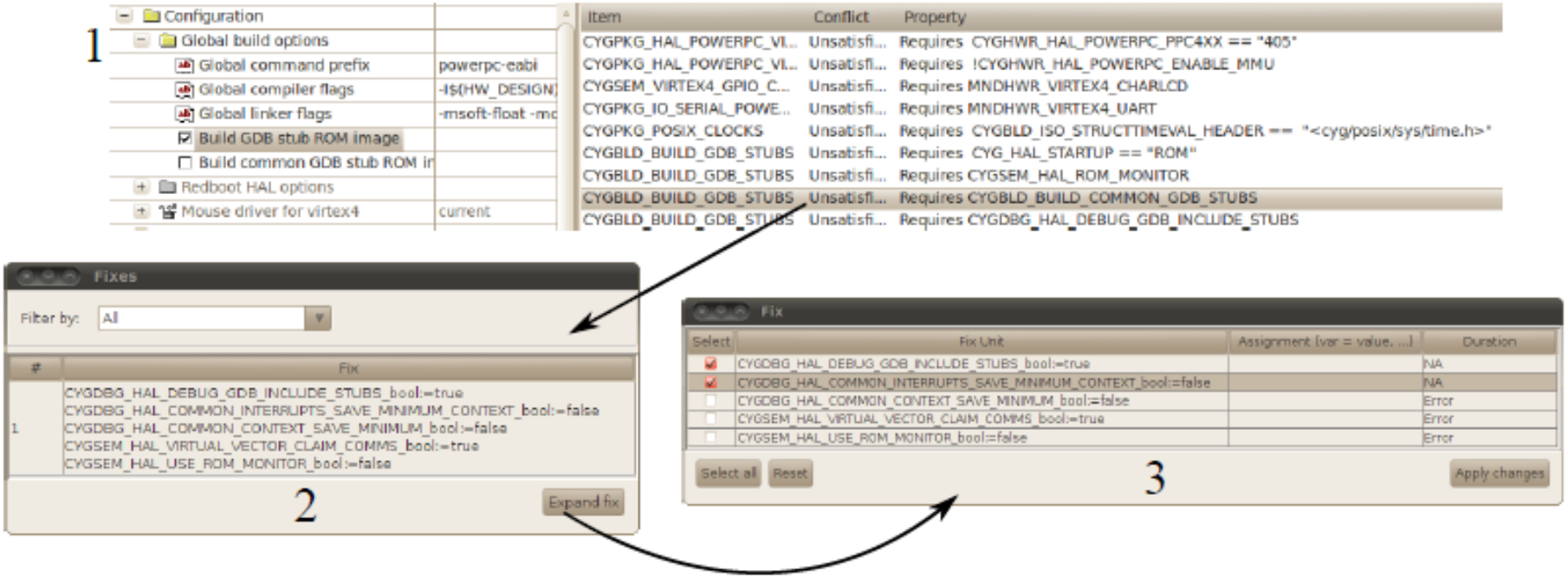
- Provide users with the variables with *error* and *permanent* durations

- Users should change the durations

Our Contribution

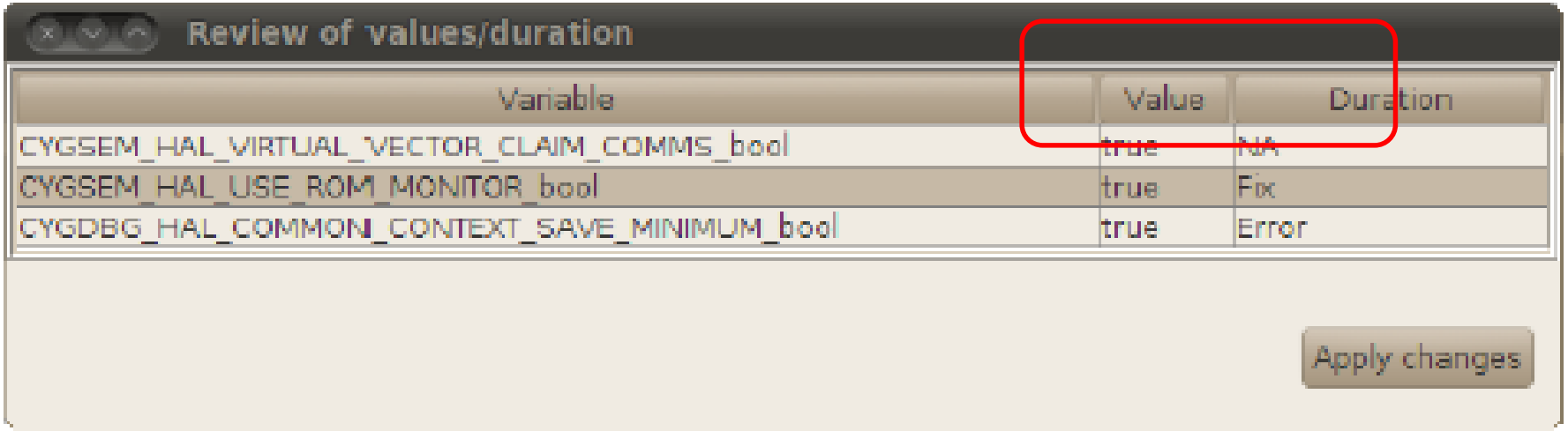
- A priority-based approach to locating a desirable fix through user feedbacks
- An algorithm to implement the approach using any fix generation algorithm
- An empirical evaluation that shows the overall reduction of choices exposed to the user

Supporting Tool: Smart Fixer



(a) SmartFixer: Interactive process GUI for fix resolution

Smart Fixer: providing feedbacks



Evaluation

- Sources
  - Version history from 2 open source projects that cause large fix lists
  - Simulate the user change from the default configurations to the final configurations

Project	Architecture	Variables	Constraints	Errors (initial conf.)
ReconOS	virtex4	933	330	56
	xilinx	765	272	48

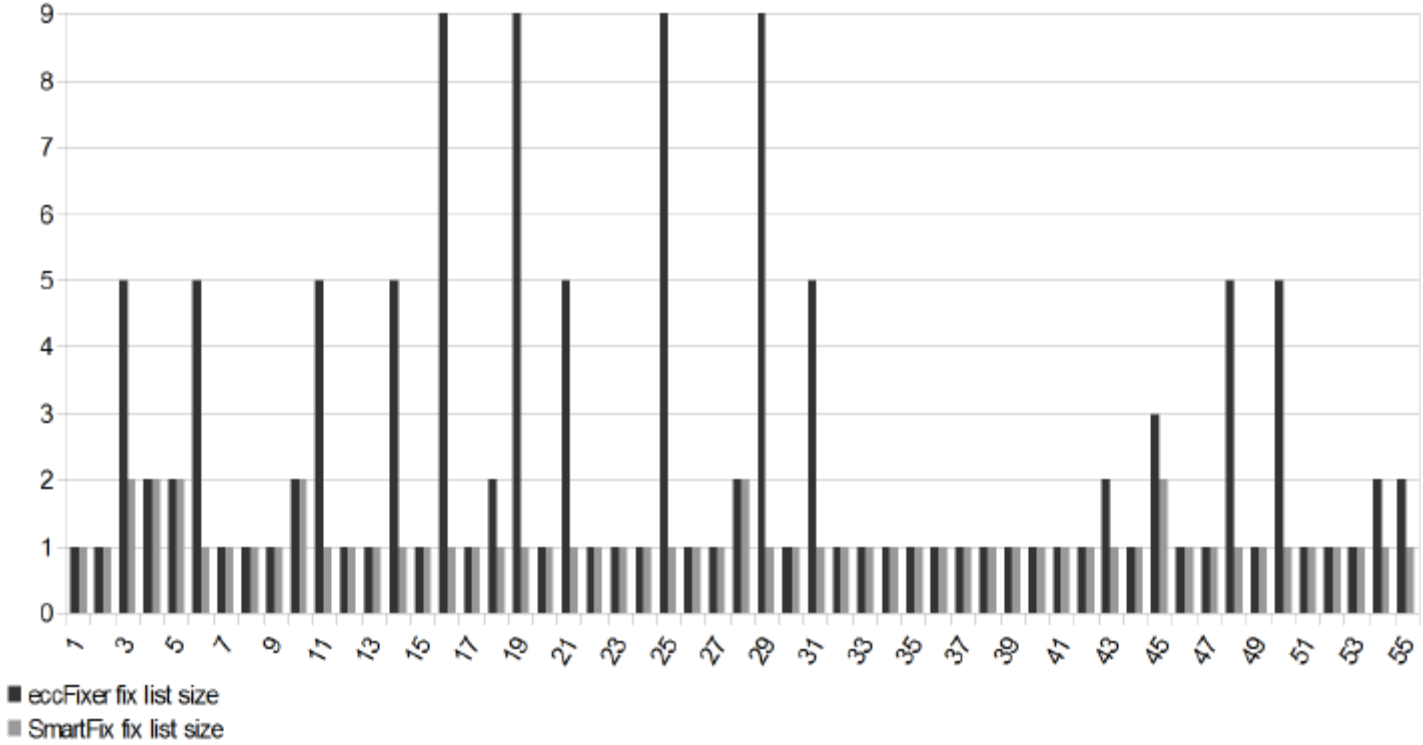
Evaluations

- Steps:
  - Generate a fix for each error, simulate the user feedback

Situation #	Current Value	Fix Changes	Final Value	Operation
1	a = 1	a < 1	a = 2	Reject Fix duration
2	a = 1	a >1	a = 2	Accept Assign new value
2s	a = 2	a > 2	a = 2	Reject Error duration

  - Count the number of fixes and variables

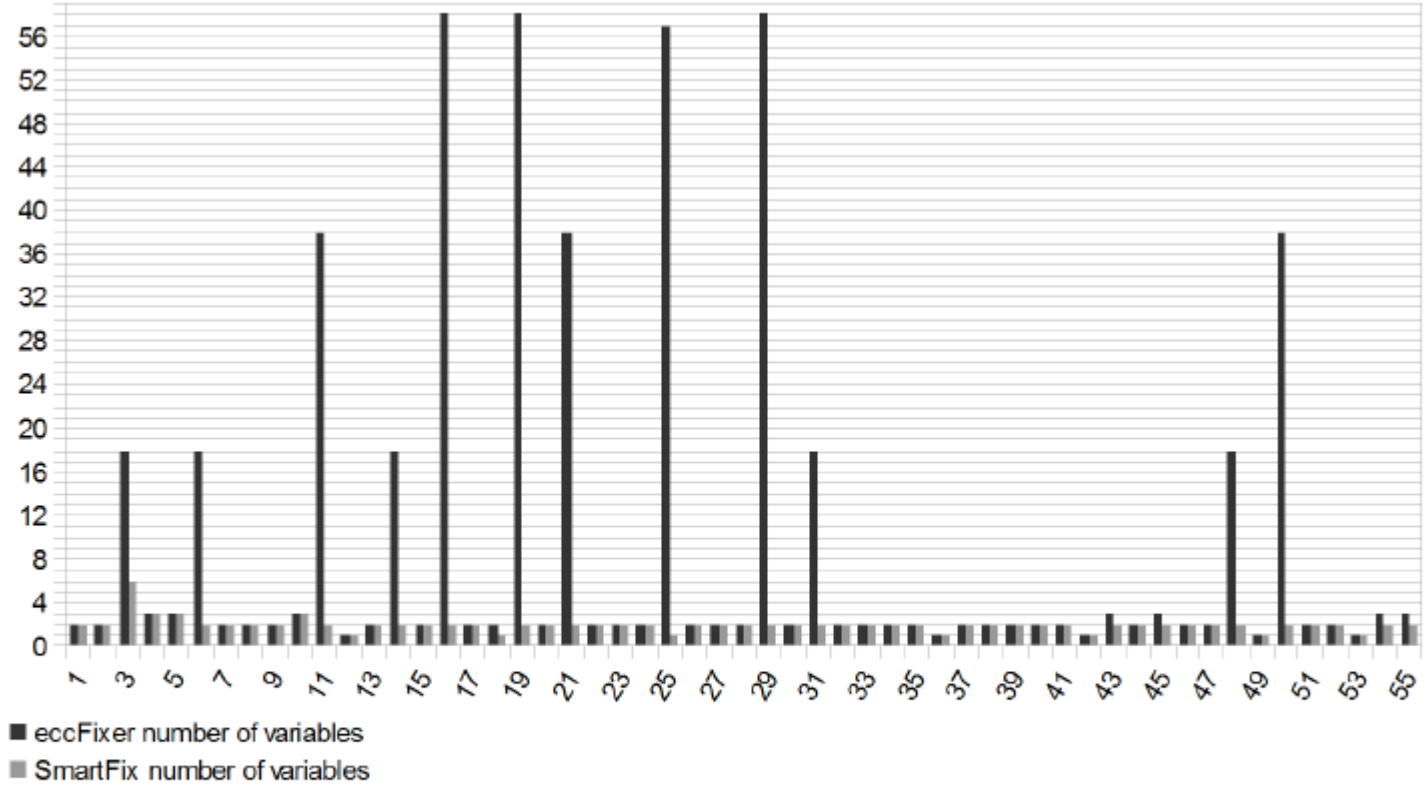
Evaluation Results – virtex4 (1/2)



(a) Fix list size

The number of fixes is decreased in 31% of the errors.
In average, there is a reduction of 22%, with a maximum reduction of 89% in the number of fixes

Evaluation Results – virtex4 (2/2)

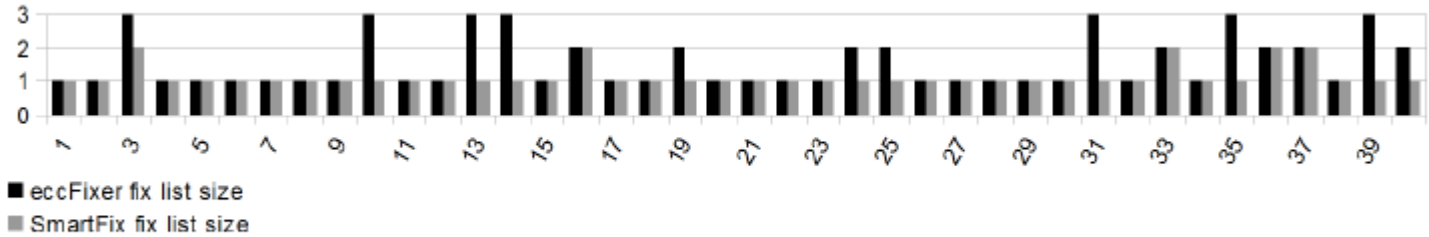


(b) Number of variables

Fig. 6: Experimental results for virtex4

The number of variables is decreased by 23% in average, with a maximum reduction of 98%

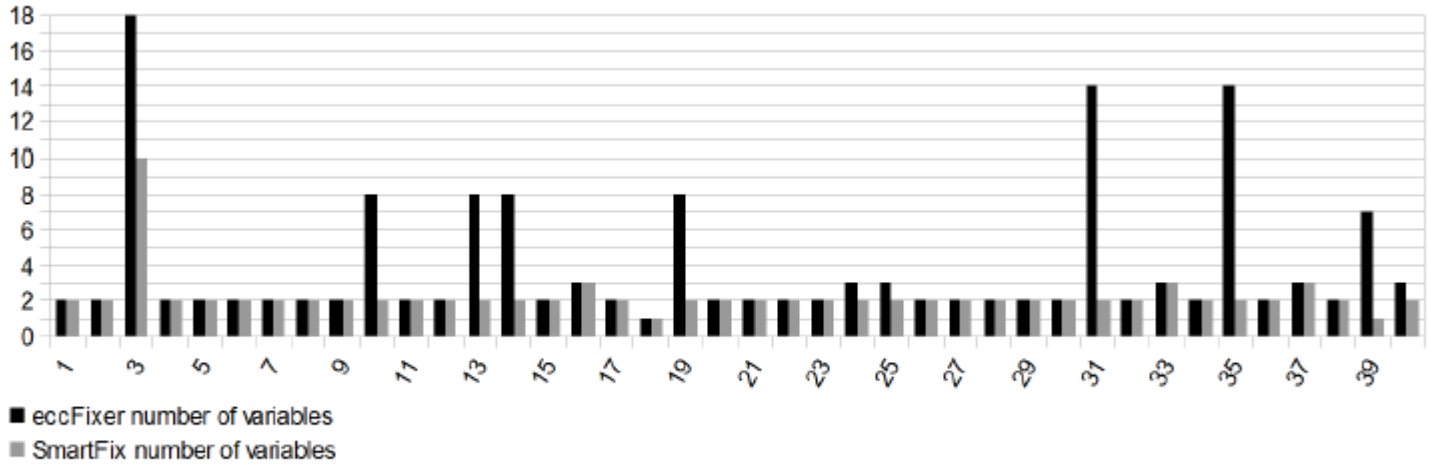
Evaluation Results – xilinx (1/2)



(a) Fix list size

The number of fixes is decreased in 28% of the errors.
In average, there is a reduction of 16%, with a maximum reduction of 2/3 in the number of fixes

Evaluation Results - xilinx (2/2)



(b) Number of variables

The number of variables is decreased by 18% in average, with a maximum reduction of 86%

Summary

- Error Resolution is difficult in configuring large systems
- Range fixes can be generated efficiently
- Large fix list could be controlled by priorities

Thank you for your attention!

- **References**:

- Yingfei Xiong, Arnaud Hubaux, Steven She, Krzysztof Czarnecki. Generating Range Fixes for Software Configuration. In ICSE'12: Proceedings of 34th International Conference on Software Engineering, pages 89-99, June 2012.
- Bo Wang, Leonardo Passos, Yingfei Xiong, Krzysztof Czarnecki, Haiyan Zhao, Wei Zhang. SmartFixer: Fixing Software Configurations based on Self-adaptive Priorities. In SPLC'13: Proceedings of 17th International Software Product Line Conference, August 2013.
- Arnaud Hubaux, Yingfei Xiong, Krzysztof Czarnecki. A User Survey of Configuration Challenges in Linux and eCos. VaMoS'12: Sixth International Workshop on Variability Modelling of Software-intensive Systems, January 2012.
- Leonardo Passos, Marko Novakovic, Yingfei Xiong, Thorsten Berger, Krzysztof Czarnecki, Andrzej Wasowski. A Study of Non-Boolean Constraints in Variability Models of an Embedded Operating System. FOSD'11: 3rd International Workshop on Feature-Oriented Software Development, June 2011.