Supporting Software Product Line Testing by Optimizing Code Configuration Coverage

László Vidács¹, Ferenc Horváth², József Mihalicza³, Béla Vancsics² and Árpád Beszédes²

¹MTA-SZTE Research Group on Artificial Intelligence, Hungary
²Department of Software Engineering, University of Szeged, Hungary
³NNG LLC, Budapest, Hungary
Context

NNG ships navigation solutions on a broad spectrum:

- Automotive line fit solutions for tier 1 clients
  Over 30 car brands carry iGO navigation (Qnx, Android, Linux, WinCE)

- White label core product
  After-market head units, mobile apps (iOS, WinCE, WinMobile, Android)

- Mobile navigation app for B2C end users
Product Line

NNG philosophy: „Navigation for All”

- Achieved by a single code base for core functionalities
- Customizations should integrate well with core features
- SPL: code variability at preprocessor level
  - Platforms (and variants), compilers, rendering engines, 32bit/64bit
  - Windows CE/Mobile/PC, QNX, Linux, Android, iOS
  - Features, customizations
Research goal

- Testing release configurations is not sufficient
  - Get a feature from Config A and turn it on in Config B
- Efficient testing of the configurable core code
- Research goal:
  
  **Select small number of configurations which cover large amount of code**
Preprocessor based Variability

```
#define PLATFORM_WIN32
#define A 2
#define B 10

Coverage
Source code lines with enabled presence conditions

#define PLATFORM_WIN32
#define A 2
#define B 10

Variable
(Configuration)

Preprocessor based Variability

1  #if A == 1
2      #define B 2
3  #endif
4  #if A == 2
5      #define B 6
6  #endif
7  #if !defined(B)
8      byte x;
9  #else
10      B >= 4
11        int x;
12  #endif

Block

Presence condition
```
Search algorithms

- Find N(<10) configurations with highest possible coverage

Approach

- Build each configuration incrementally (greedy approach)
- Create new configurations until N is reached

- **Block-based** approach
  - Try to cover the largest uncovered block

- **Variable-based** approach
  - Select the variable which results the highest overall coverage increase
Block-based algorithm

1. Examine largest uncovered block

   \#if X > 0 && Y == 5

2. Satisfy presence condition

3. Extend candidate configuration

   \#define A 100
   \#define B -43
   \#define X 1
   \#define Y 5

4. Refresh the global coverage
Variable-based algorithm

1. For each free configuration variable
   - Define: A 100
   - Define: B -43
   - Define: X 1

2. Compute coverage for each value interval
   - Undefined
   - [-∞; 0]: 35
   - [1; ∞]: 121

3. Extend candidate configuration & refresh the global coverage

   + B1
   + B2
   + B3
   + B4
   + B5
   + B6
   + BN
**iGO Navigation Measurements**

<table>
<thead>
<tr>
<th>Condition type</th>
<th>Blocks</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtered (T, F, #error)</td>
<td>11,847 (25%)</td>
<td>682,300 (35%)</td>
</tr>
<tr>
<td>Configuration</td>
<td>22,067 (47%)</td>
<td>920,926 (48%)</td>
</tr>
<tr>
<td>Mixed</td>
<td>10,085 (22%)</td>
<td>271,710 (14%)</td>
</tr>
<tr>
<td>Non-configuration</td>
<td>2,811 (6%)</td>
<td>50,064 (3%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>46,810</strong></td>
<td><strong>1,925,000</strong></td>
</tr>
</tbody>
</table>

- **#error directives**
  - Prevent invalid configurations
- Non-configuration variables -> mixed conditions
  - `MODULE1_DETAILED_DIAG, PERSONAL_PATCHES_JOE`
Results: block-based, $N = 10$

- Config coverage: 98.74%
- Search time: 22 min

![Graph showing coverage (kLOC) against configurations]

- Red line: Config coverage
- Blue line: Delta coverage
- Green line: Config LOC

Config LOC remains constant, indicating stable coverage across configurations.
Results: variable-based, $N = 10$

Config coverage: 75.56%

Search time: 27 min

Coverage (kLOC)

Config LOC

Search time

Config coverage

Delta coverage
Results: block-based, N = 50

- Config coverage: 99.74%
- Search time: 2 h

Diagram showing coverage (kLOC) and configurations.
Results & plans

- **BLOCK-BASED**
  - 98.74%
  - N=10, 22 min

- **VARIABLE-BASED**
  - 75.56%
  - N=10, 27 min

Enhanced #error directives

Multiple variables at a time

Hybrid algorithm