Evaluating lexical approximation of program dependence

Seongmin Lee, David Binkley, Nicolas Gold, Syed Islam, Jens Krinke, Shin Yoo

Journal of Systems and Software

KAIST

LOYOLA
UNIVERSITY MARYLAND

UCL

Naturalness of source code

Naturalness of source code

- Java

```
127
128   private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNa
129
130   private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finaliz
131
```

```
200       try {
201         ((FinalizableReference) reference).finalizeReferent();
202       } catch (Throwable t) {
203         logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
204       }
```

- Python

```
456         except Exception:
457             if not from_error_handler:
458                 raise
459             self.logger.exception('Request finalizing failed with an ' 'error while handling
460         return response
```

Code lines handing the logging function contains the word ‘log’

Naturalness of source code

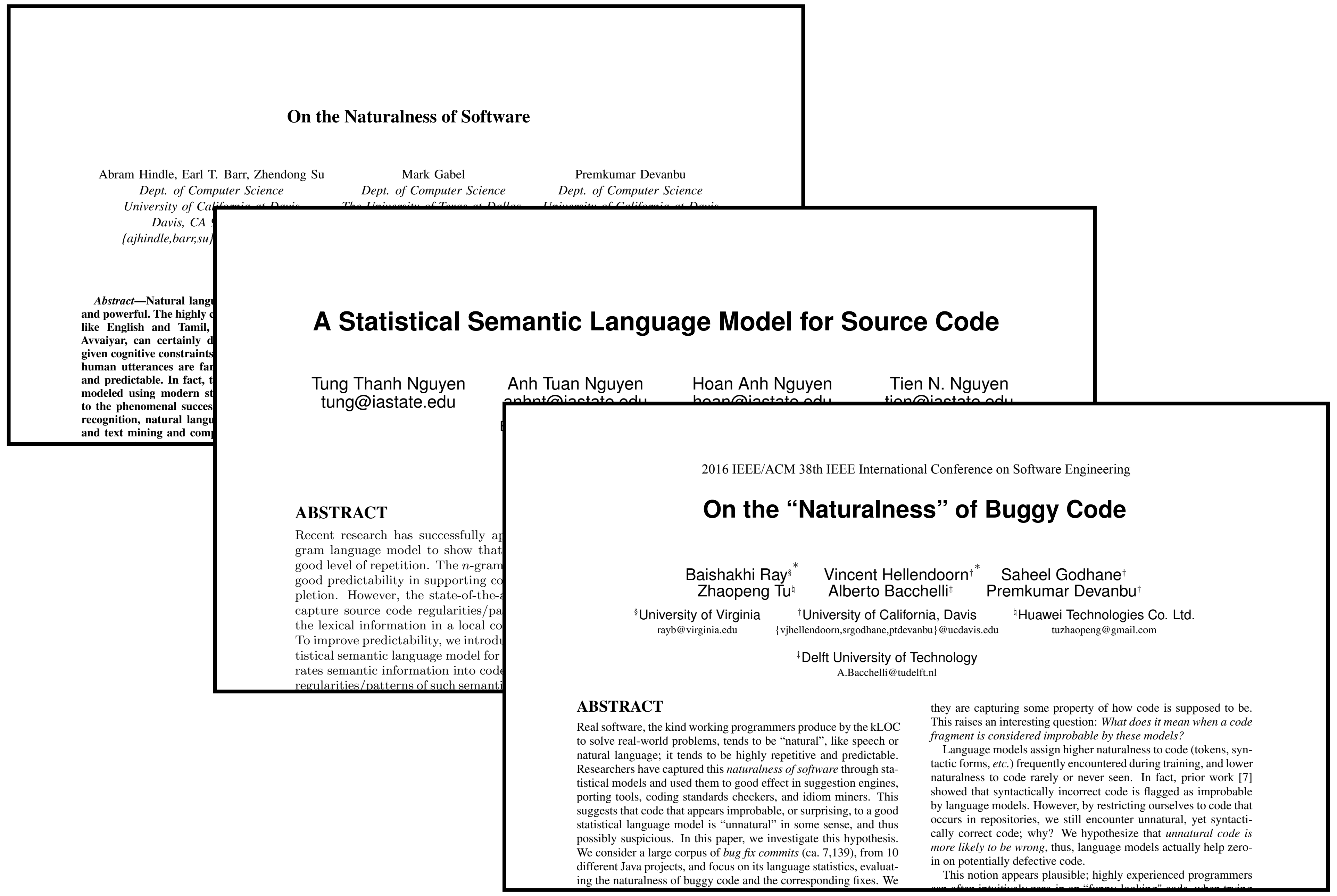
- Java

```java
private static final Logger logger = Logger.getLogger(FinalizableReferenceQueue.class.getNar

private static final String FINALIZER_CLASS_NAME = "com.google.common.base.internal.Finalize
```

```java
    try {
      ((FinalizableReference) reference).finalizeReferent();
    } catch (Throwable t) {
      logger.log(Level.SEVERE, "Error cleaning up after reference.", t);
    }
```

- Python

```python
        except Exception:
            if not from_error_handler:
                raise
            self.logger.exception('Request finalizing failed with an ' 'error while handling
        return response
```



Code lines handing the logging function contains the word 'log'

Like a natural language, a source code is also repetitive and predictable.

Naturalness of source code

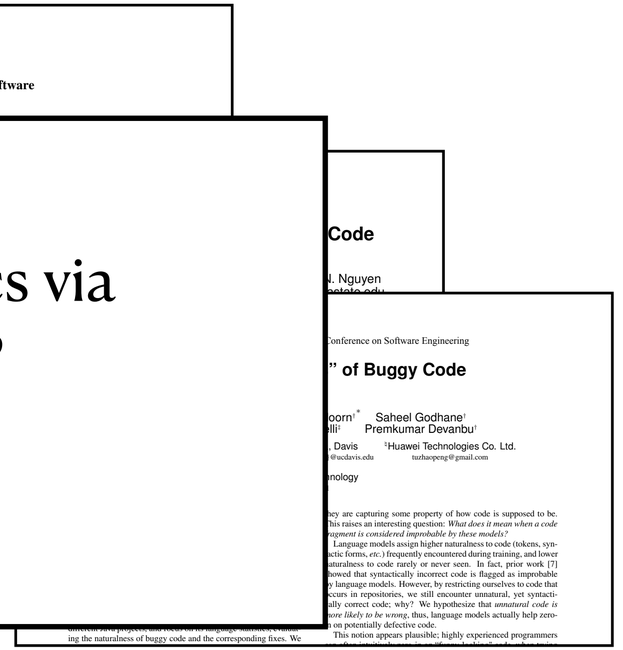
- Java

```
127
128   private static final Lo
129
130   private static final St
131
```

```
200       try {
201         ((FinalizableRefe
202       } catch (Throwable
203         logger.log(Level.
204       }
```

- Python

```
456           except Exception:
457               if not from_e
458                   raise
459               self.logger.e
460           return response
```



Can we approximate the program semantics via lexical information of the source code?

➡ Program dependency analysis

Code lines handing the logging function contains the word ‘log’

Like a natural language, a source code is also repetitive and predictable.

Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.

Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“ORBS: %d\n”, i);
}
```

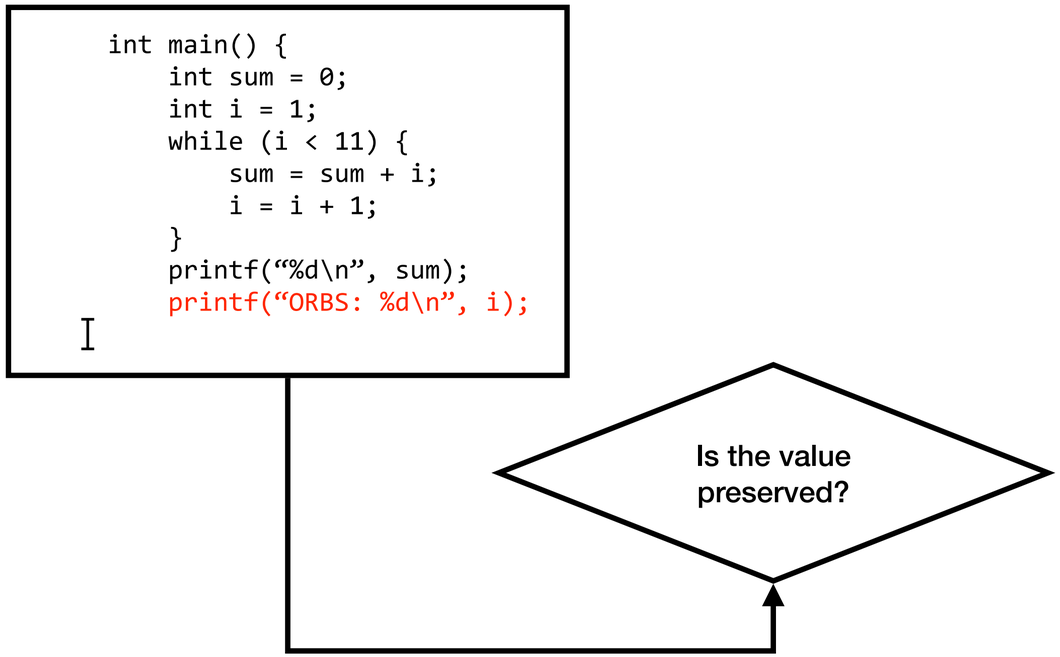
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.

```
int main() {
    int sum = 0;
    int i = 1;
    while (i < 11) {
        sum = sum + i;
        i = i + 1;
    }
    printf(“%d\n”, sum);
    printf(“ORBS: %d\n”, i);
}
```

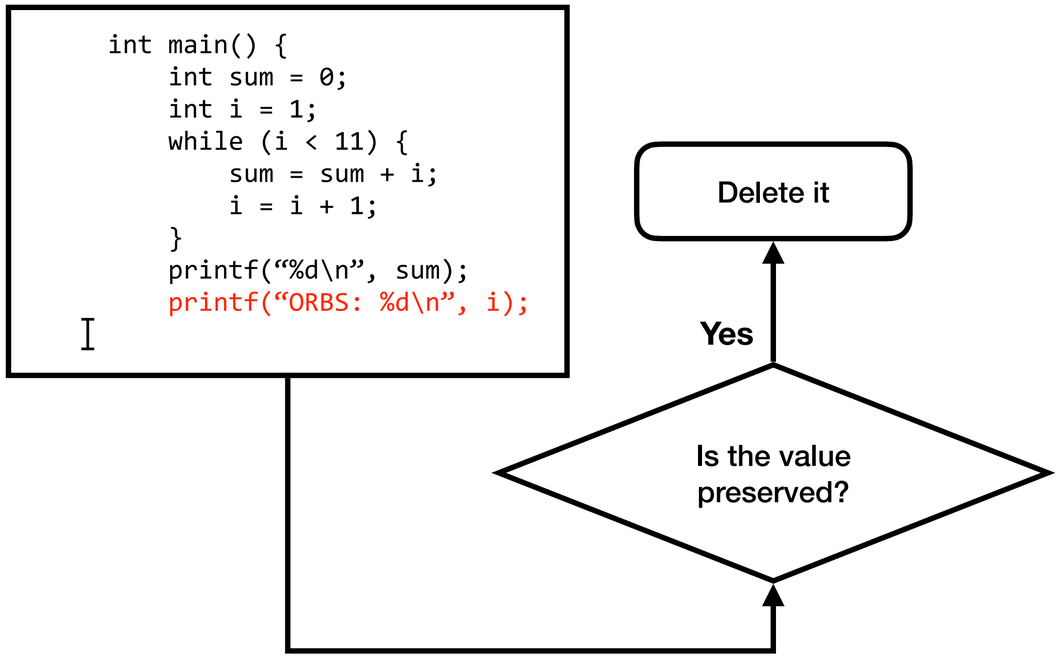
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



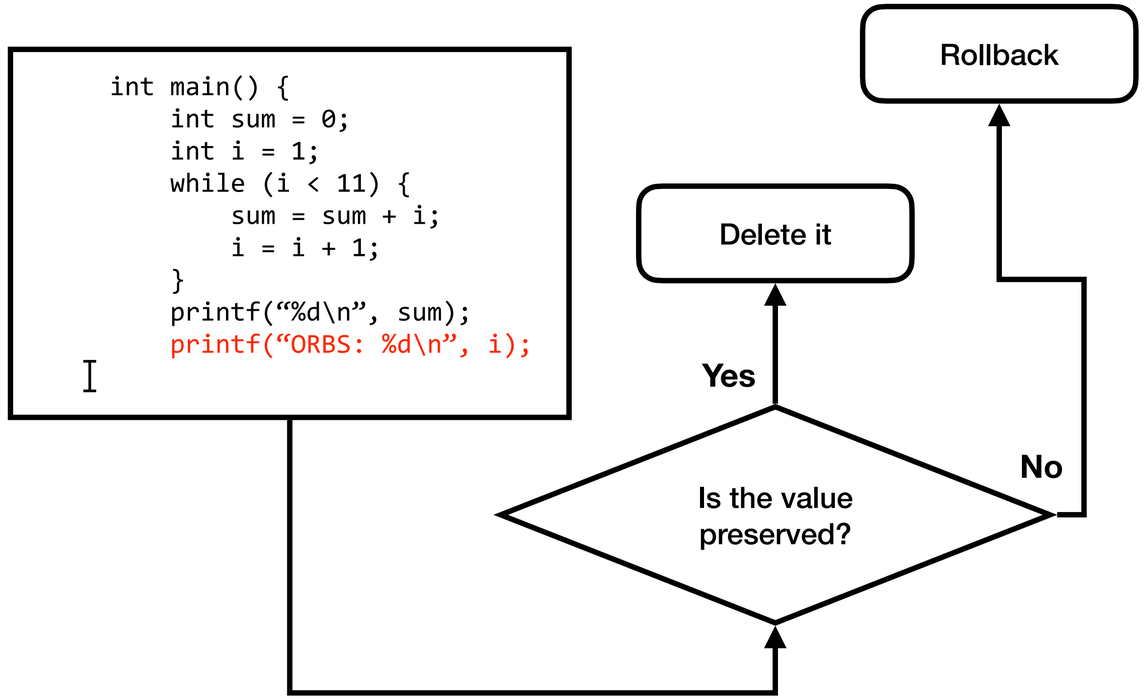
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



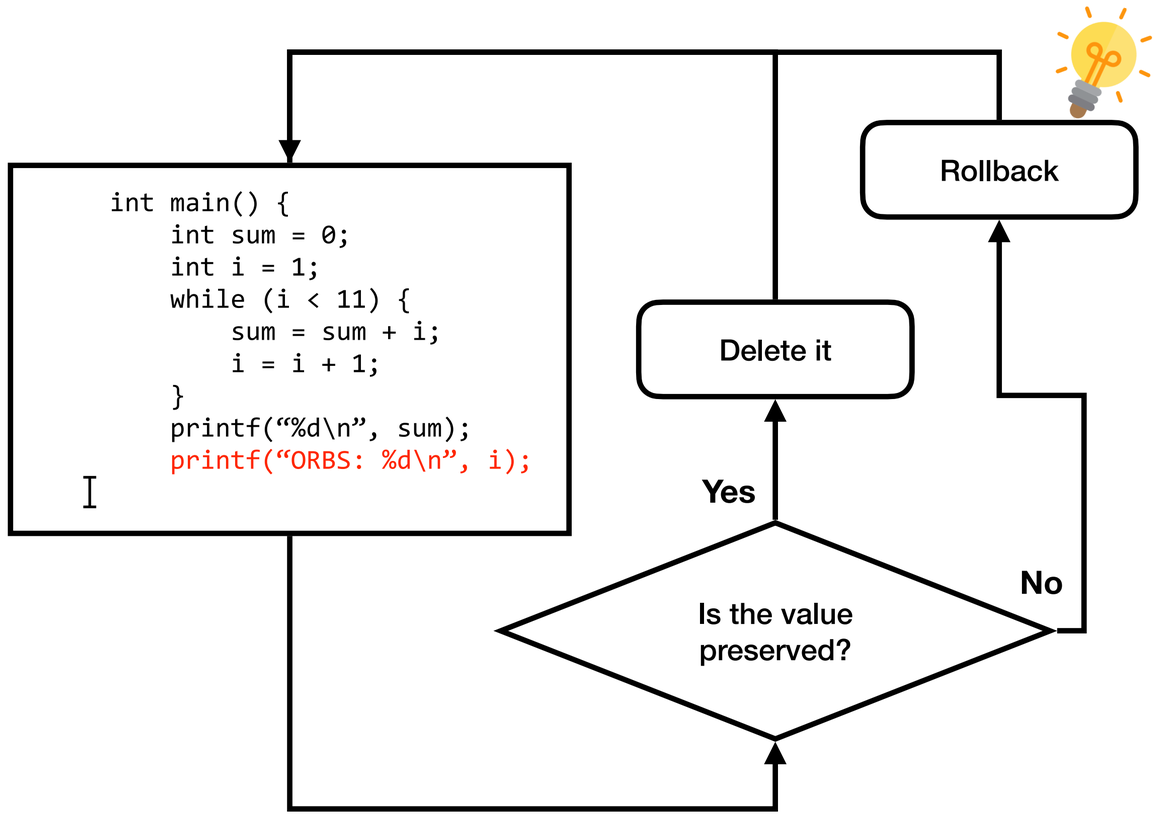
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



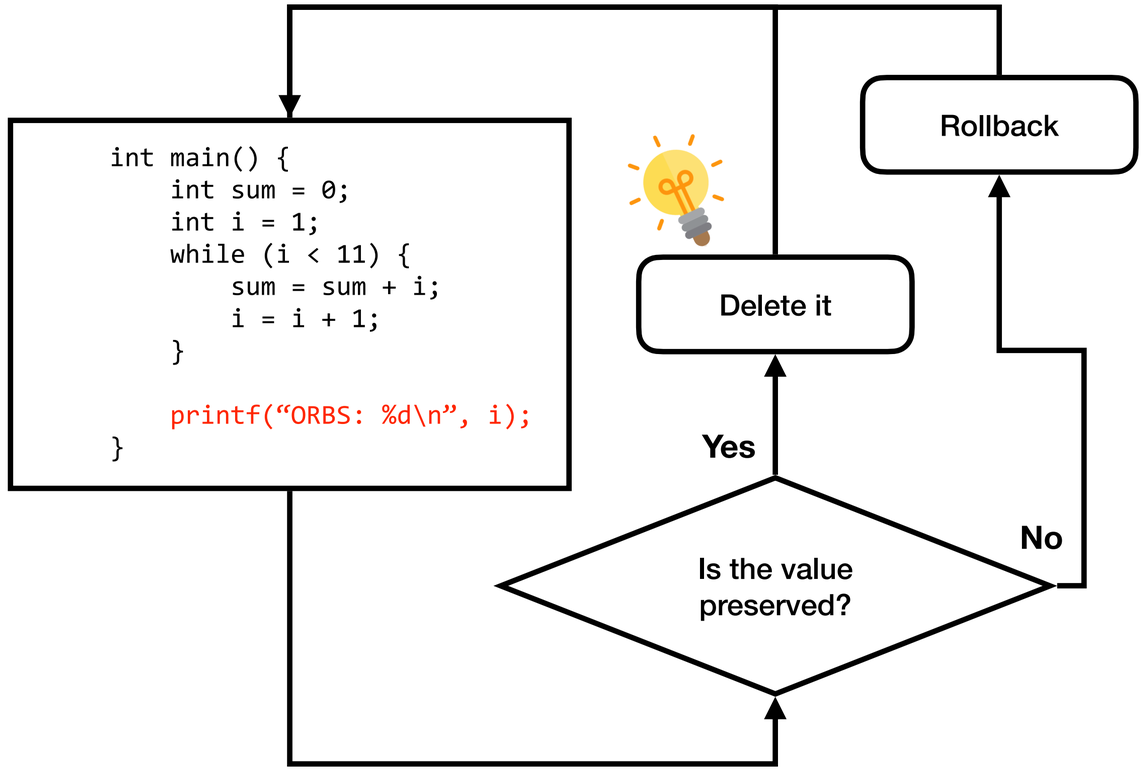
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



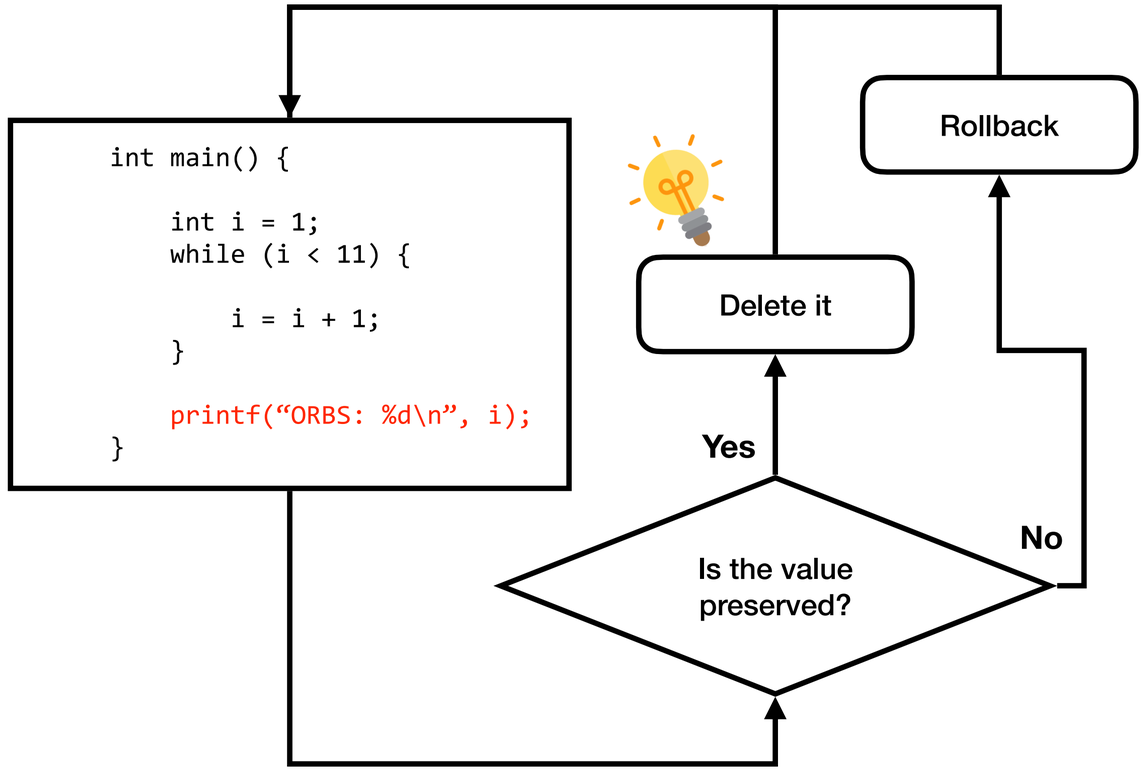
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



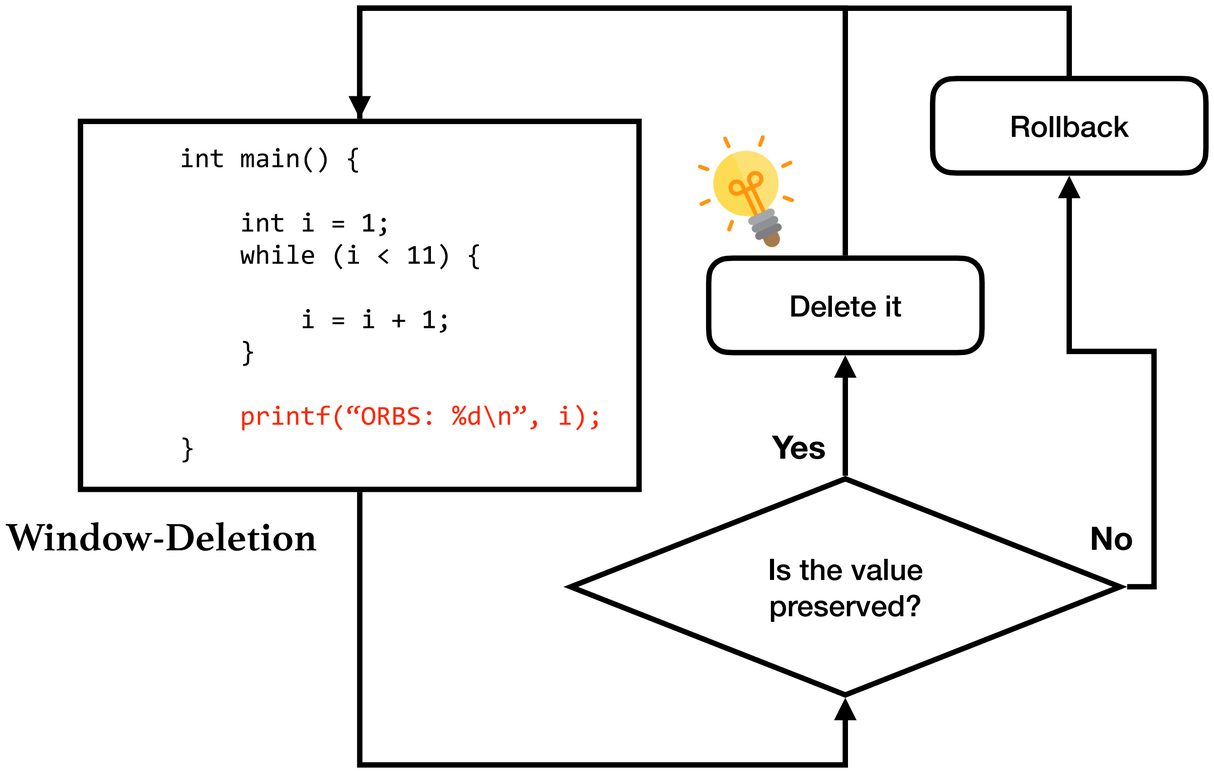
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



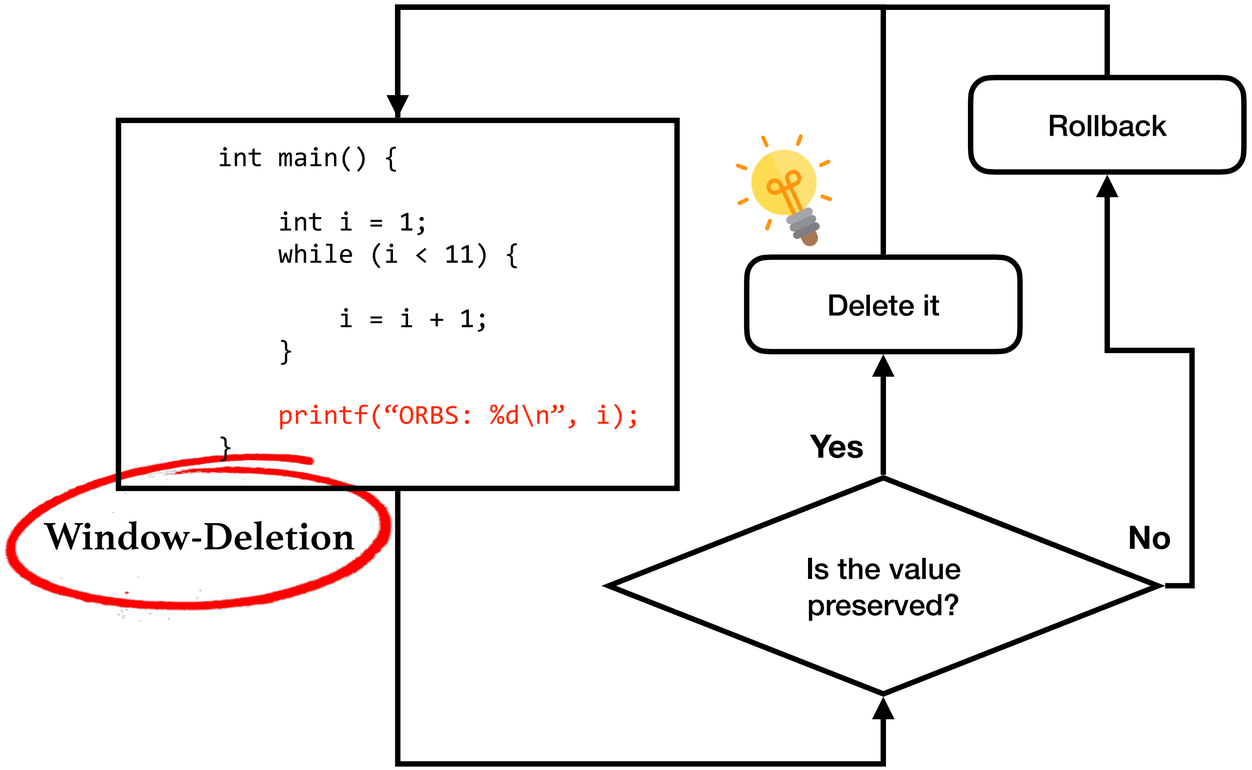
Observation-Based Slicing (ORBS)

- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.



Observation-Based Slicing (ORBS)

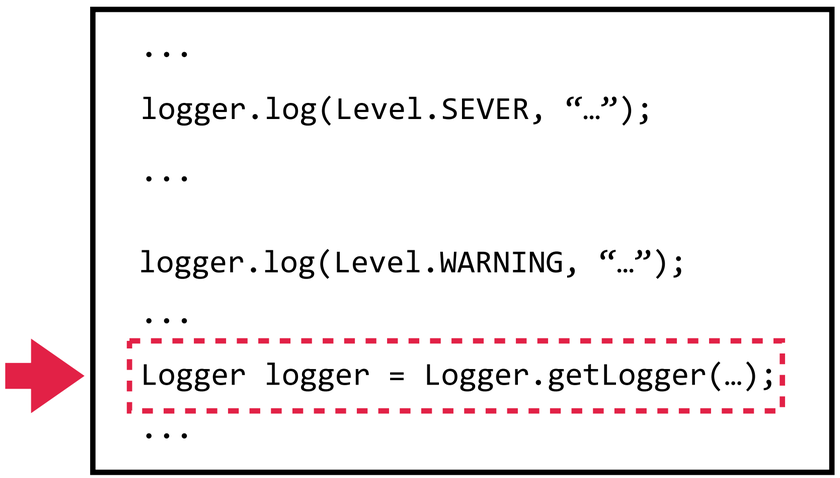
- Purely dynamic program slicing technique
- Use code-level modification & runtime information
- Thus, it can work on
  - multi-lingual programs, or
  - programs with third party libraries.
- Scalability issue 
  - Takes around 7,200 seconds to delete 220 lines.
    ⇒ **0.03 del/s = 32.7 s/del**
    (* ‘escape’ package in Guava)



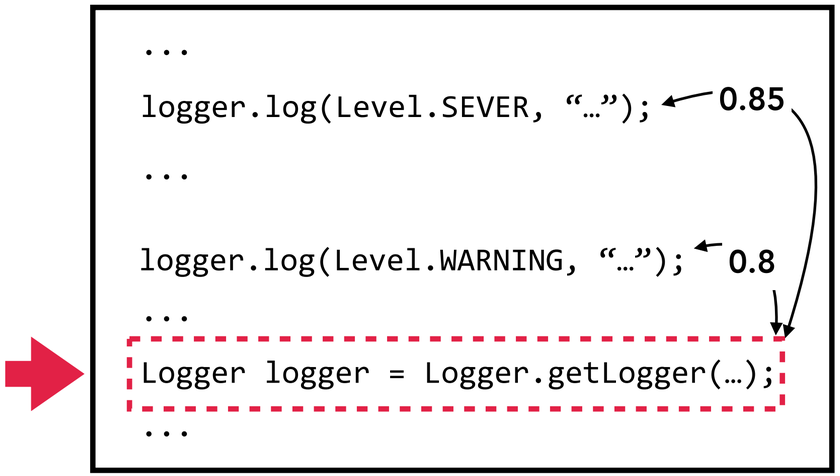
Lexical deletion operator

```
...
logger.log(Level.SEVER, “…”);
...

logger.log(Level.WARNING, “…”);
...
Logger logger = Logger.getLogger(…);
...
```

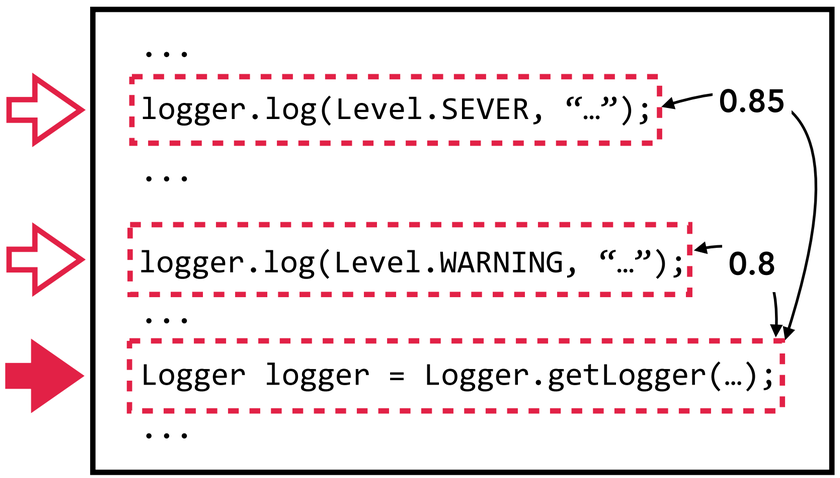
Lexical deletion operator



Lexical deletion operator

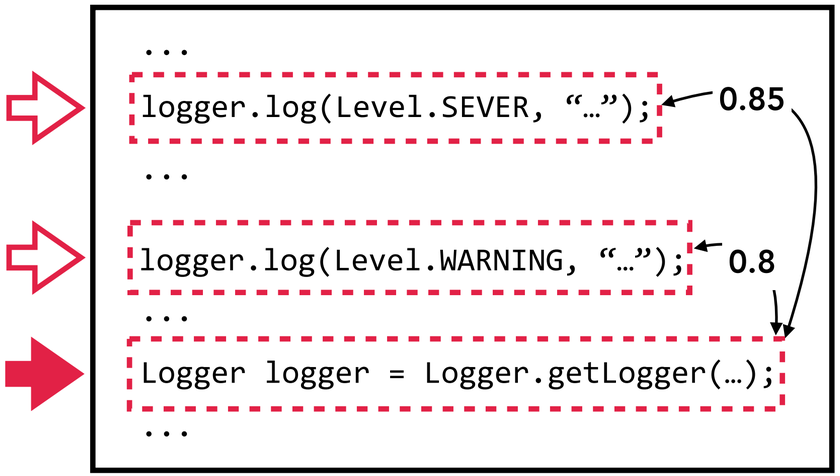


Lexical deletion operator

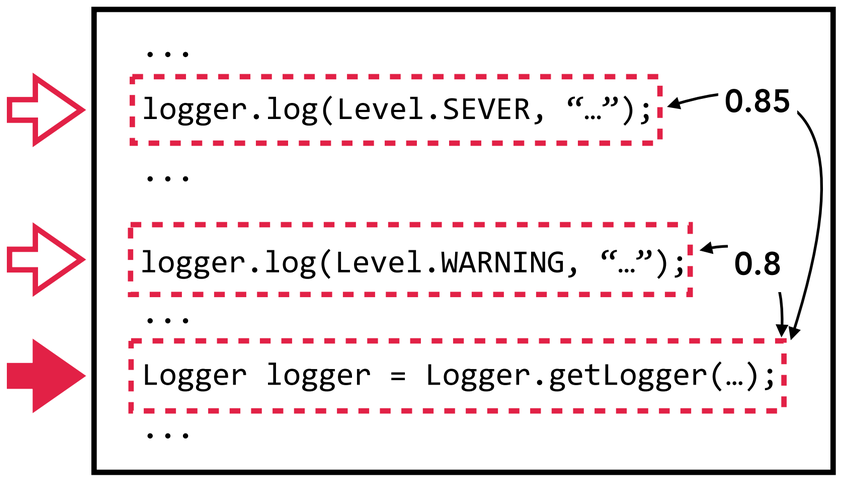


Lexical deletion operator

Shares the functionality



Lexical deletion operator



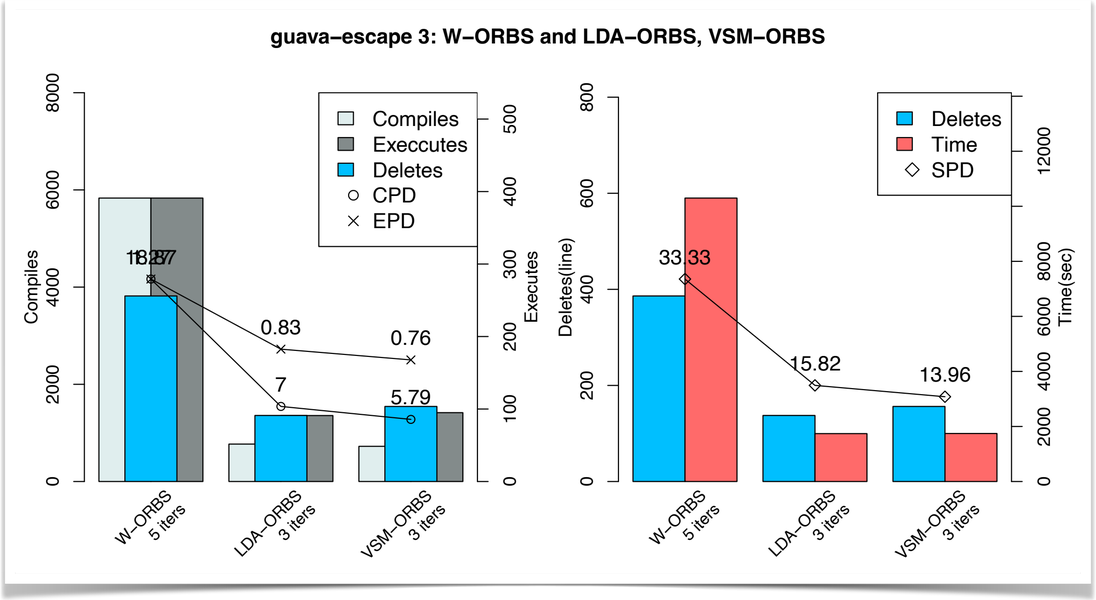
- Two language model to calculate the similarity
  - Vector Space Model (VSM)
  - Latent Dirichlet Allocation (LDA)

- Advantage of the lexical deletion operators:
  - Can delete an ***arbitrary number*** of similar lines in a single deletion
  - Can delete ***non-consecutive lines***
  - Still, language agnostic

Lexical deletion operator

ORBS vs. LS-ORBS

- Benchmarks: 18 slicing criteria from Java and C programs
  - Java: apache commons csv, cli, and guava library
  - C: Siemens suite



LS-ORBS achieves / uses

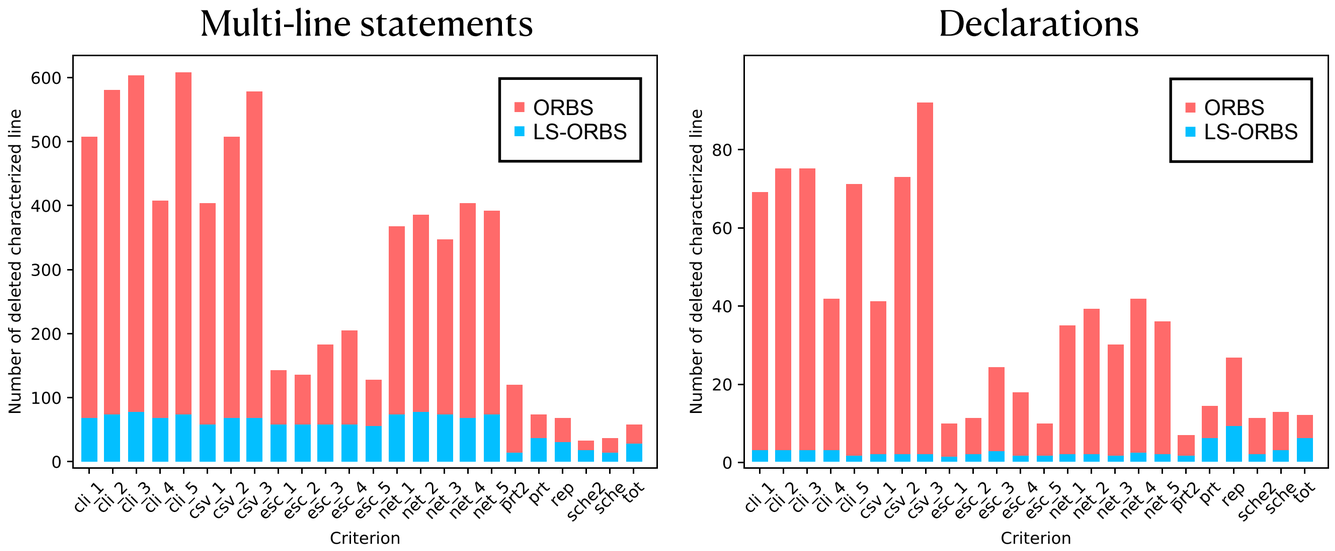
👍 **45%** # of compilations,

👍 **70%** # of executions,

👎 **38%** # of deleted lines,

👍 **64%** time taken per deleted line compared to ORBS.

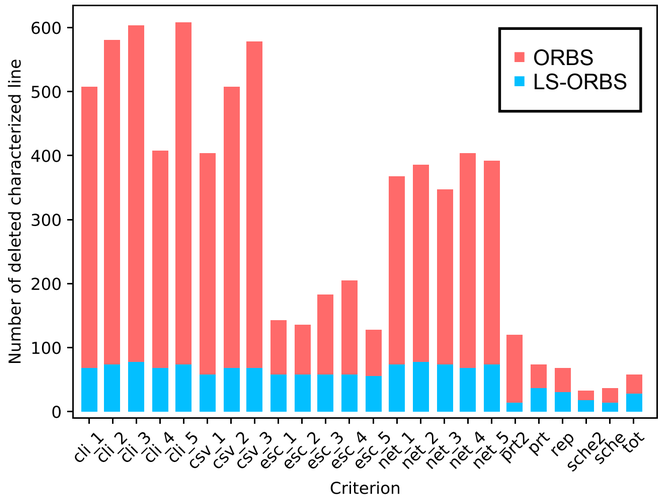
When are lexical deletion operators effective / ineffective?



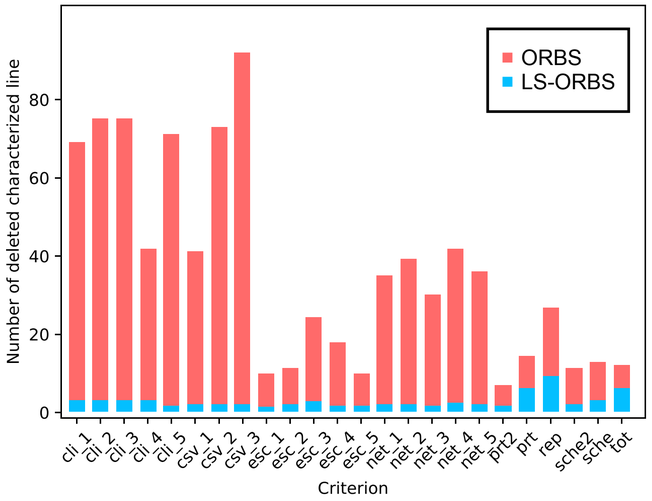
Syntactic structures in source code is challenging to the lexical deletion operators

When are lexical deletion operators effective / ineffective?

Multi-line statements

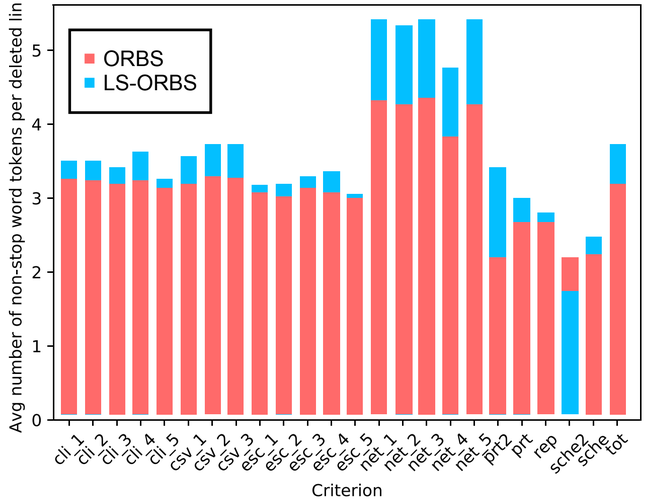


Declarations



Syntactic structures in source code is challenging to the lexical deletion operators

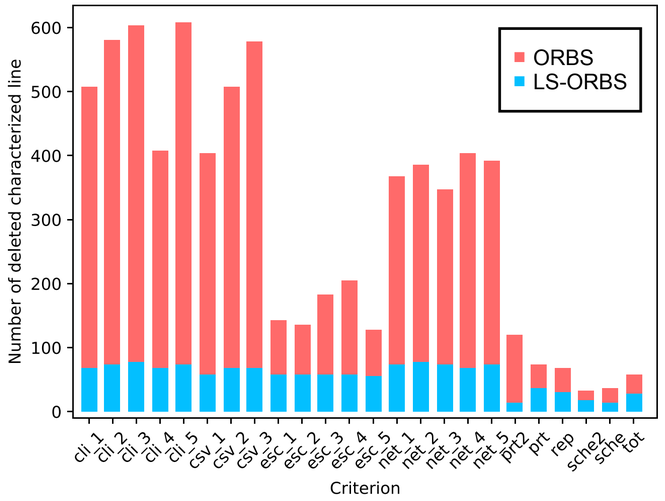
Non-stop words in deleted lines



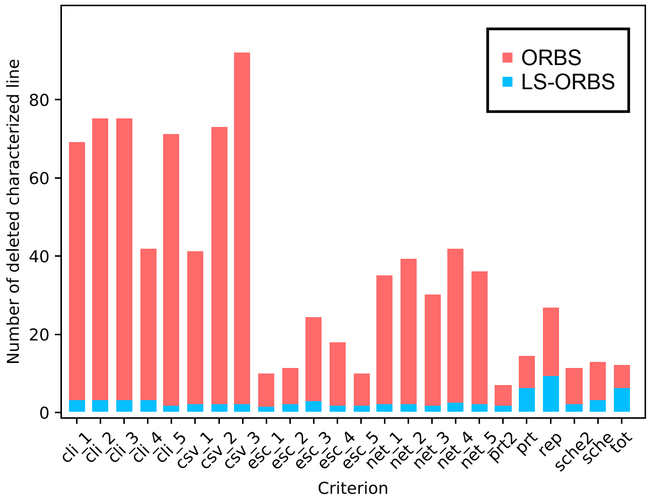
Lexical deletion operators are effective in the statements with non-stop words.

When are lexical deletion operators effective / ineffective?

Multi-line statements

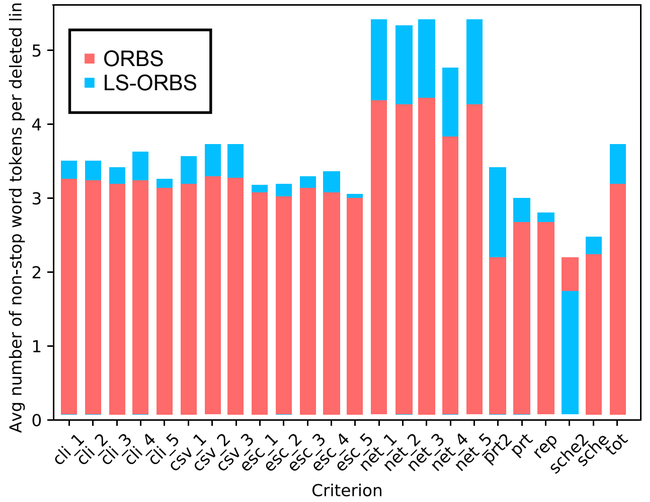


Declarations



Syntactic structures in source code is challenging to the lexical deletion operators

Non-stop words in deleted lines



Lexical deletion operators are effective in the statements with non-stop words.

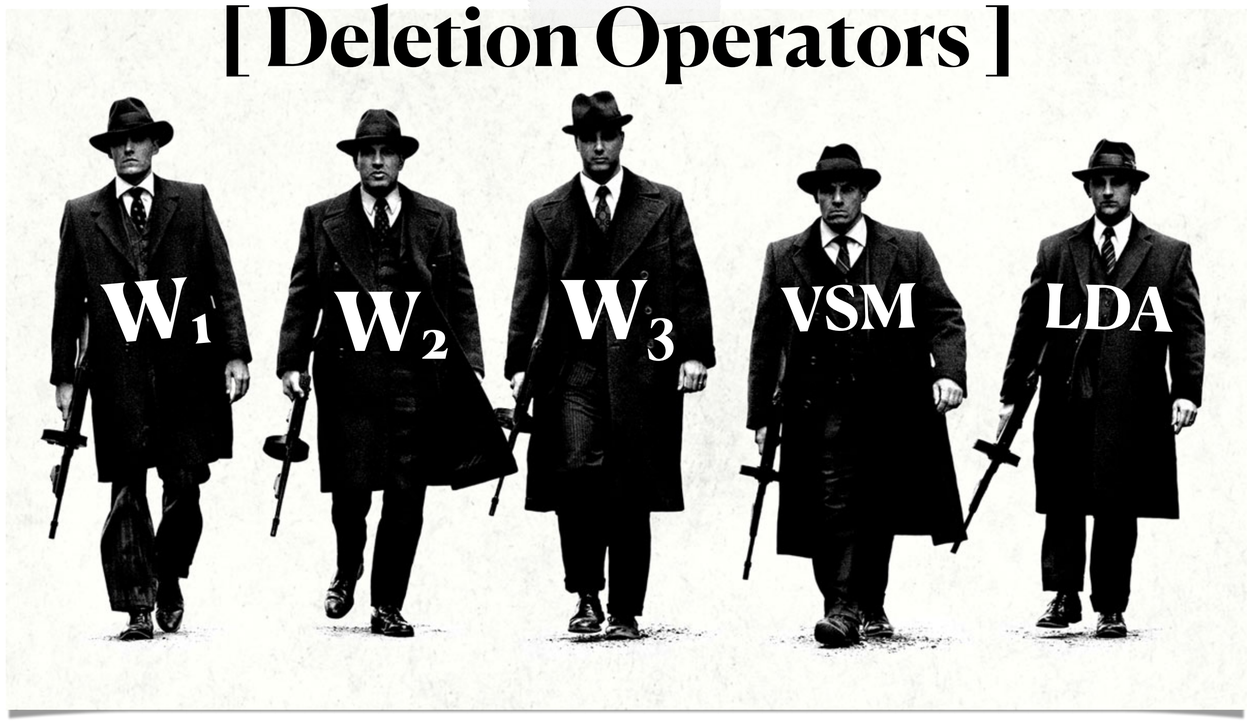
There is a complementary relation between window deletion and lexical deletion.

MOBS: Multi-operator ORBS

```
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```

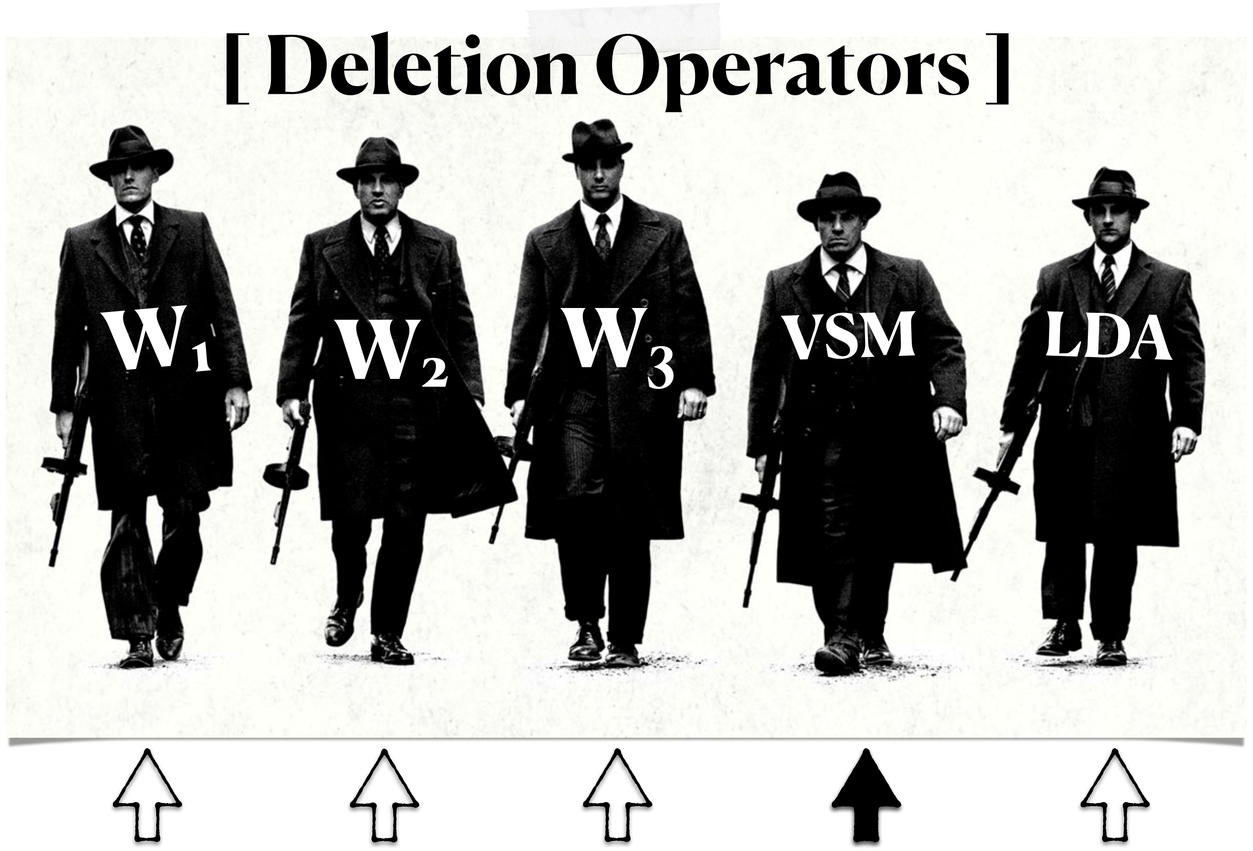
MOBS: Multi-operator ORBS

```
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



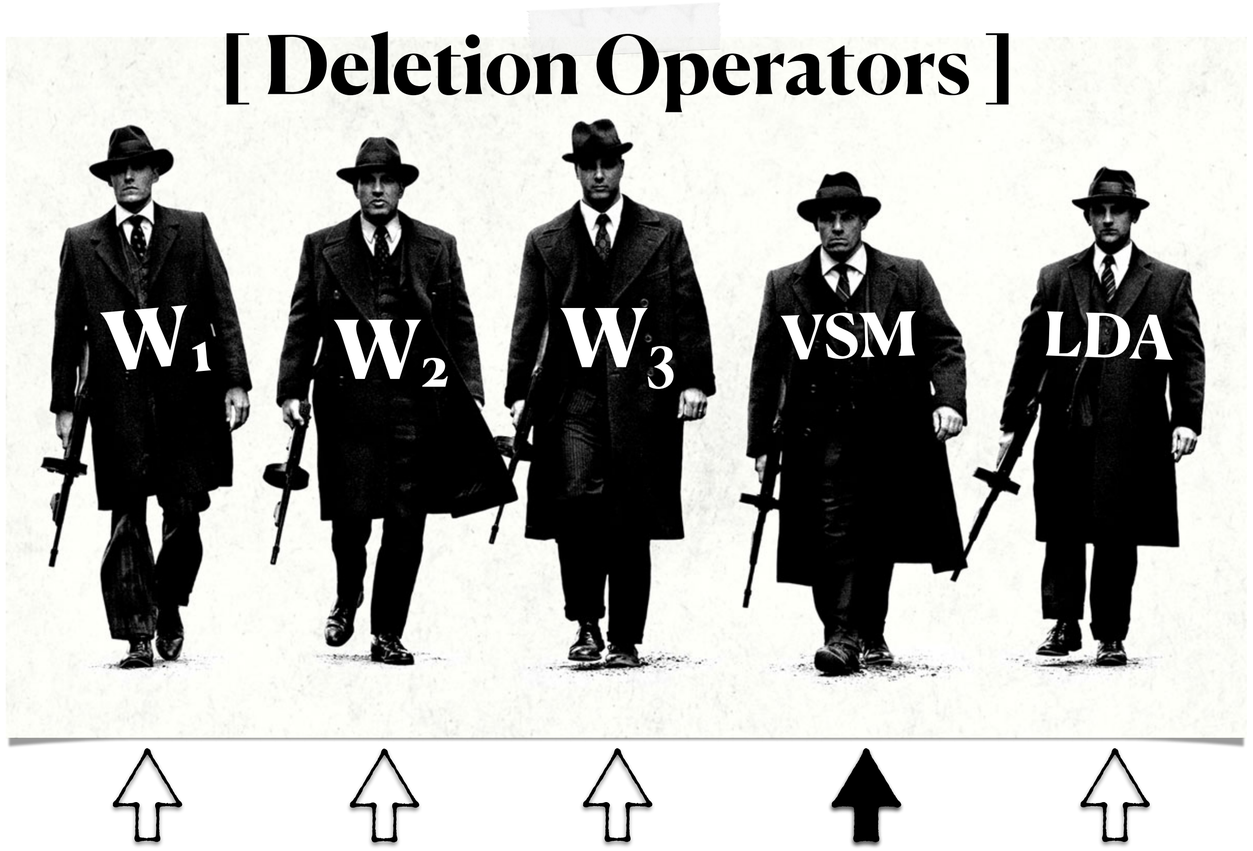
MOBS: Multi-operator ORBS

```
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = np.append(
                obs_row, 0 if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            )
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



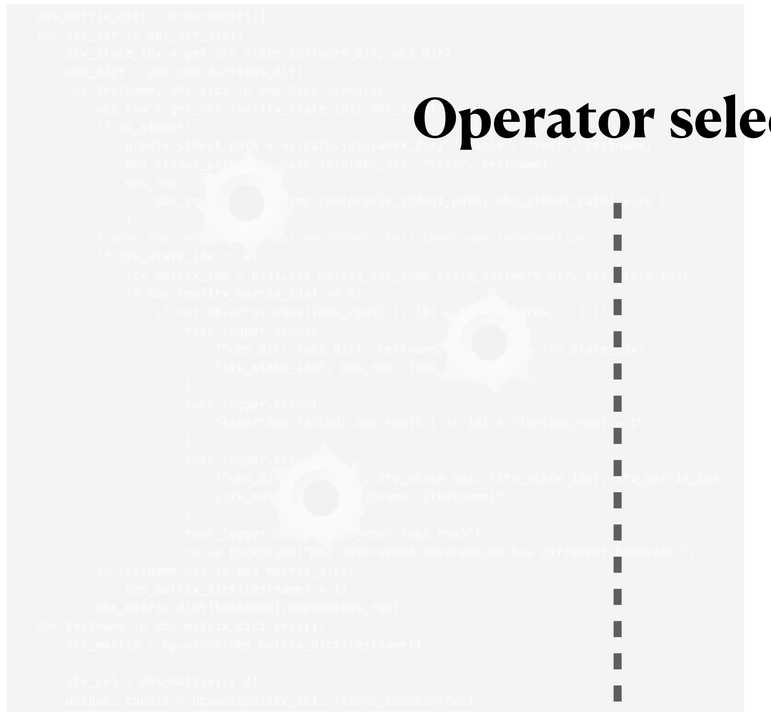
MOBS: Multi-operator ORBS

```
obs_matrix_dict = OrderedDict()
for obs_dir in obs_dir_list:
    itv_state_idx = get_itv_state_idx(work_dir, obs_dir)
    cmp_dict = get_cmp_dict(obs_dir)
    for testname, obs_dict in cmp_dict.items():
        obs_row = get_obs_row(itv_state_idx, obs_dict)
        if is_stdout:
            oracle_stdout_path = os.path.join(work_dir, "oracle", "test", testname)
            obs_stdout_path = os.path.join(obs_dir, "test", testname)
            obs_row = [
                obs_row[0] if filecmp.cmp(oracle_stdout_path, obs_stdout_path) else 1
            ]
        # When the intervention has no effect, tell there was intervention.
        if itv_state_idx != 0:
            itv_matrix_idx = util.get_matrix_idx_from_state_idx(work_dir, itv_state_idx)
            if obs_row[itv_matrix_idx] == 0:
                if not np.array_equal(obs_row[1:], [0] * (len(obs_row) - 1)):
                    root_logger.debug(
                        f"obs_dir: {obs_dir}, testname: {testname}, itv_state_idx:
                        {itv_state_idx}, obs_row: {obs_row}"
                    )
                    root_logger.error(
                        "Assertion failed: obs_row[1:] != [0] * (len(obs_row) - 1"
                    )
                    root_logger.error(
                        f"obs_dir: {obs_dir}, itv_state_idx: {itv_state_idx}, itv_matrix_idx:
                        {itv_matrix_idx}, testname: {testname}"
                    )
                    root_logger.error(f"obs_row: {obs_row}")
                    raise Exception("Not intervened observation has different behavior.")
        if testname not in obs_matrix_dict:
            obs_matrix_dict[testname] = []
        obs_matrix_dict[testname].append(obs_row)
for testname in obs_matrix_dict.keys():
    obs_matrix = np.array(obs_matrix_dict[testname])

    itv_col = obs_matrix[:, 0]
    unique, counts = np.unique(itv_col, return_counts=True)
```



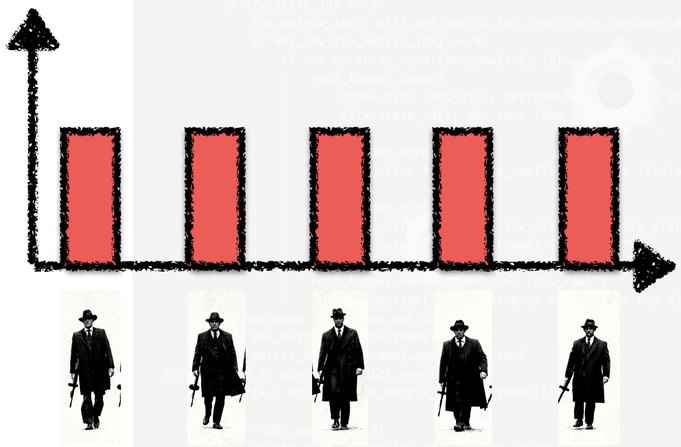
MOBS: Multi-operator ORBS

Operator selection using probability distribution



MOBS: Multi-operator ORBS

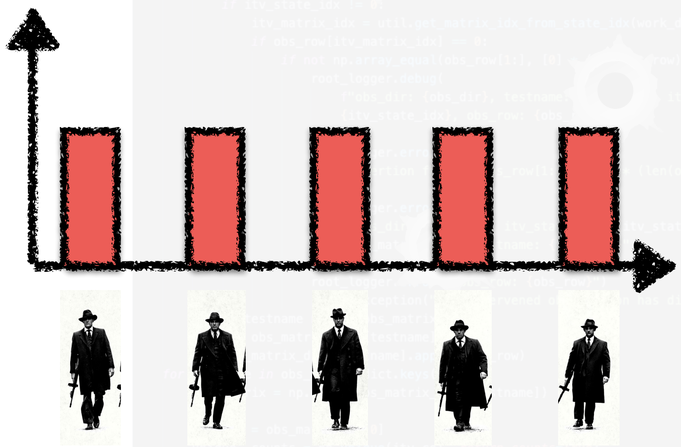
Operator selection using probability distribution



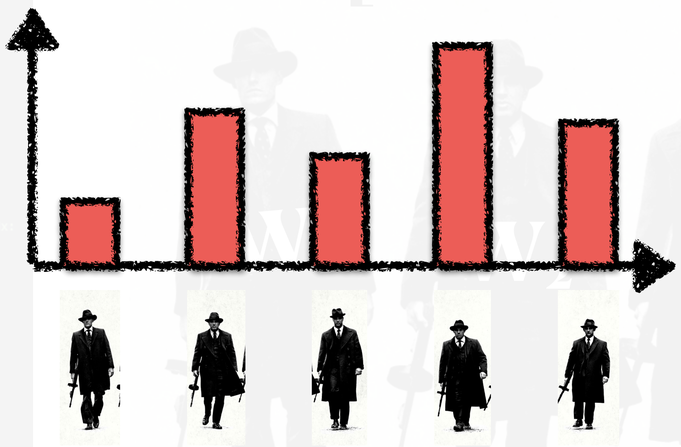
Uniform distribution

MOBS: Multi-operator ORBS

Operator selection using probability distribution



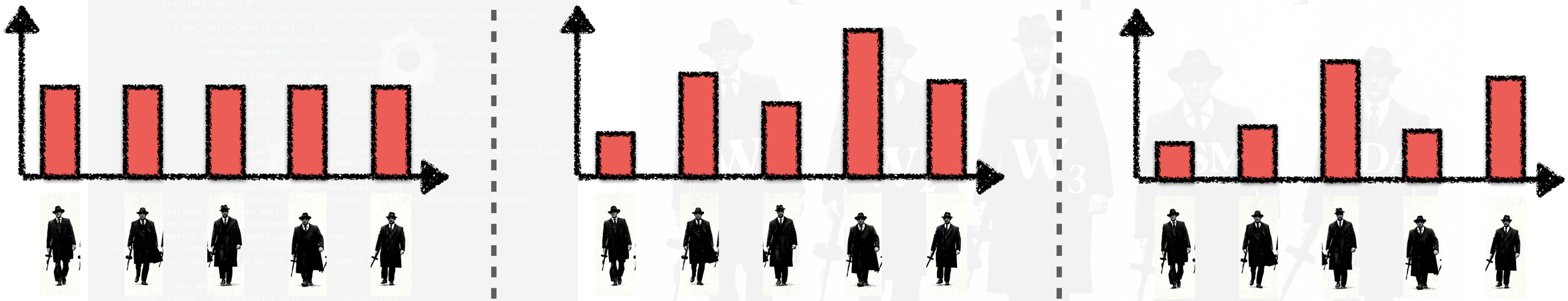
Uniform distribution



Fixed distribution based on applicability / effect of D.O

MOBS: Multi-operator ORBS

Operator selection using probability distribution



Uniform distribution

Fixed distribution based on applicability / effect of D.O

Adaptive distribution

Result

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

Result

Table 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

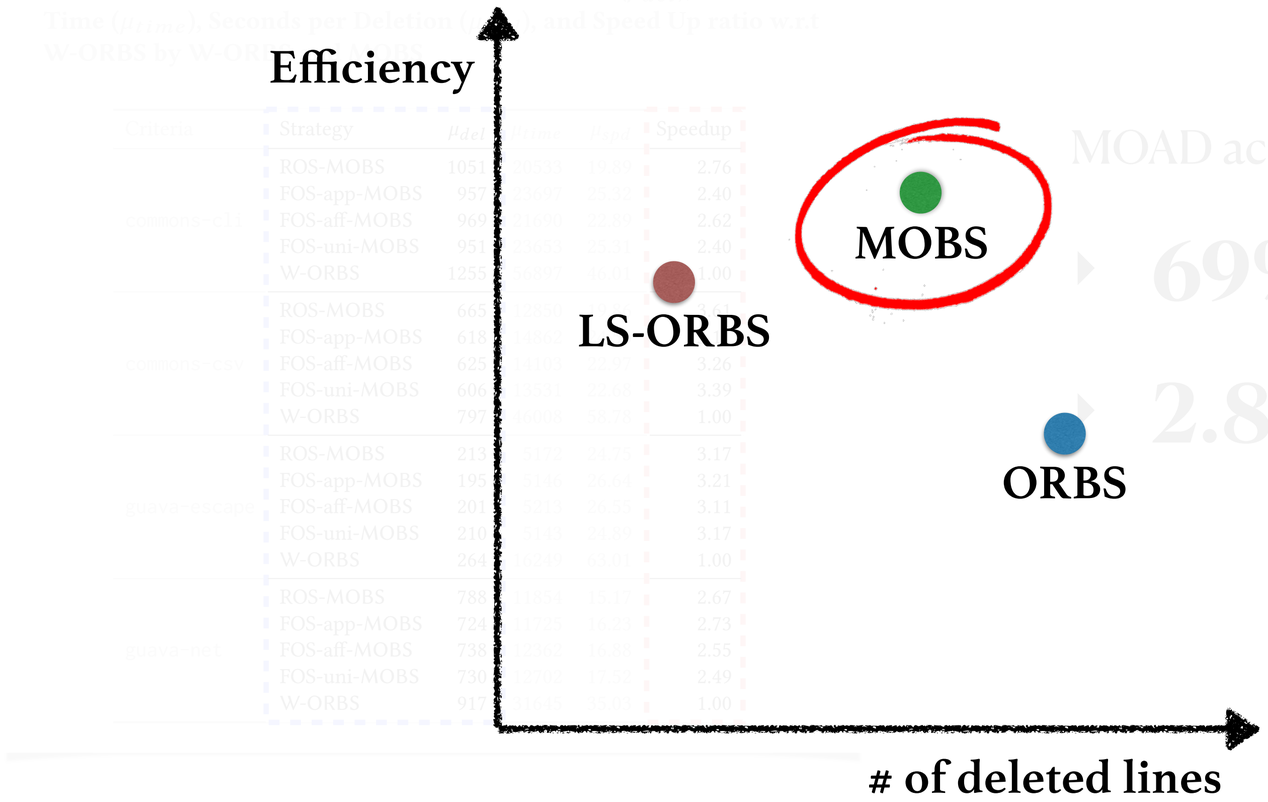
Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

MOAD achieves / uses

- **69%** # of deleted lines,
- **2.8X** faster

  compared to ORBS.

Result

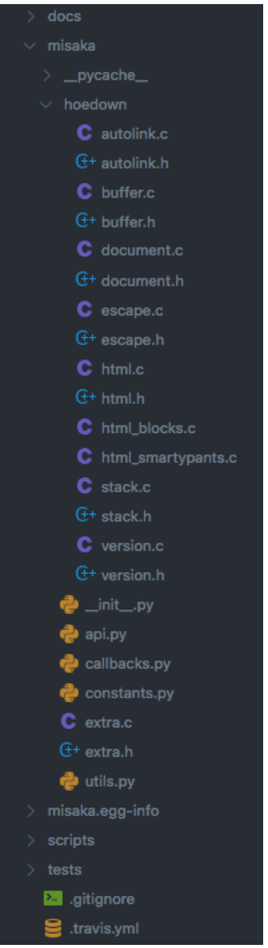


MOAD achieves / uses

- **69%** # of deleted lines,
- **2.8X** faster

compared to ORBS.

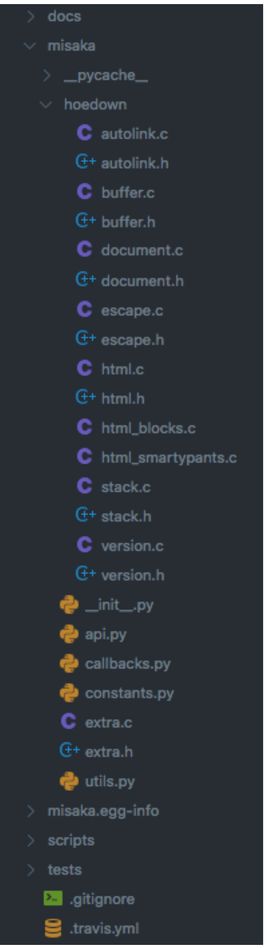
Example. Multi-lingual deletion



- Misaka(http://misaka.61924.nl)
  - A Python binding for Hoedown, a markdown parsing C library.
  - Programming language: C, Python

	NCLOC	FILES	TC
C	4360	10	
Python	473	5	
Total	4833	15	92

Example. Multi-lingual deletion



- Misaka(http://misaka.61924.nl)
  - A Python binding for Hoedown, a markdown parsing C library.
  - Programming language: C, Python

	NCLOC	FILES	TC
C	4360	10	
Python	473	5	
Total	4833	15	92

- VSM Deletion operator

```
┌ callbacks.py      (97)  > elif align_bit == TABLE_ALIGN_LEFT:
| callbacks.py      (98)  >     align = 'left'
└ hoedown/html.c    (393) > case HOEDOWN_TABLE_ALIGN_LEFT:
```

- LDA Deletion operator

```
┌ api.py                  (29)   > lib.hoedown_buffer_puts(ib, text.encode('utf-8'))
| hoedown/document.c      (2490) > hoedown_buffer_free(text);
└ hoedown/html_smartypants.c (195) > hoedown_buffer_putc(ob, text[0]);
```

- Both LDA and VSM Deletion operator

```
┌ callbacks.py      (125) > result = renderer.blockhtml(text)
└ hoedown/html.c    (635) > renderer->blockhtml = NULL;
```

Naturalness of source code

- Java
- Python

Can we approximate the program semantics via lexical information of the source code?

➡ Program dependency analysis

Code lines handing the logging function contains the word 'log'

Like a natural language, a source code is also repetitive and predictable.

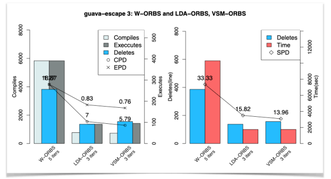
Lexical deletion operator

```
...
logger.log(Level.SEVER, "…");   ← 0.85
...
logger.log(Level.WARNING, "…");  ← 0.8
...
Logger logger = Logger.getLogger(…);
...
```

- Two language model to calculate the similarity
  - Vector Space Model (VSM)
  - Latent Dirichlet Allocation (LDA)
- Advantage of the lexical deletion operators:
  - Can delete an ***arbitrary number*** of similar lines in a single deletion
  - Can delete ***non-consecutive lines***
  - Still, language agnostic

ORBS vs. LS-ORBS

- Benchmarks: 18 slicing criteria from Java and C programs
  - Java: apache commons csv, cli, and guava library
  - C: Siemens suite



LS-ORBS achieves / uses

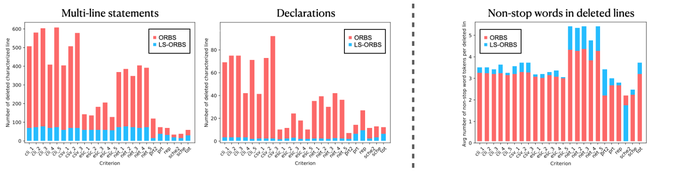
👍 **45%** # of compilations,

👍 **70%** # of executions,

👎 **38%** # of deleted lines,

👍 **64%** time taken per deleted line compared to ORBS.

When are lexical deletion operators effective / ineffective?

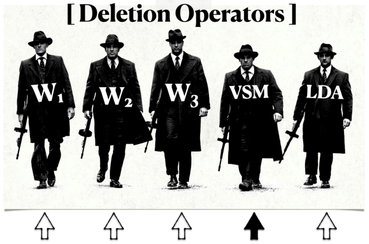


Syntactic structures in source code is challenging to the lexical deletion operators

Lexical deletion operators are effective in the statements with non-stop words.

There is a complementary relation between window deletion and lexical deletion.

MOBS: Multi-operator ORBS



Result

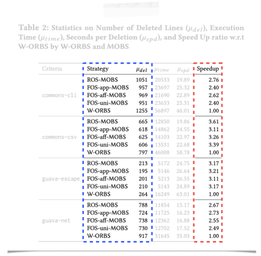


Table 2: Statistics on Number of Deleted Lines ($\mu_{\#del}$), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS and MOBS

Criteria	Strategy	#del	time	spd	Speedup
common-csv-1	BOS-MOBS	1051	50533	19.86	2.76
	POS-app-MOBS	957	47697	25.39	2.40
	POS-aff-MOBS	969	61690	22.89	2.62
	POS-uni-MOBS	951	53655	25.58	2.40
	W-ORBS	1255	54897	46.05	1.00
common-csv-2	BOS-MOBS	665	52850	39.86	3.61
	POS-app-MOBS	618	54662	24.58	3.11
	POS-aff-MOBS	625	54135	22.99	3.26
	POS-uni-MOBS	606	53551	22.68	3.39
	W-ORBS	797	56008	58.73	1.00
guava-escape-1	BOS-MOBS	213	51372	34.70	3.17
	POS-app-MOBS	195	51546	26.68	3.21
	POS-aff-MOBS	201	52215	26.56	3.11
	POS-uni-MOBS	210	51745	24.89	3.17
	W-ORBS	264	164249	63.09	1.00
guava-net	BOS-MOBS	788	51834	35.37	2.67
	POS-app-MOBS	724	51735	34.28	2.73
	POS-aff-MOBS	738	52362	34.86	2.55
	POS-uni-MOBS	730	52702	37.56	2.49
	W-ORBS	917	71645	35.08	1.00

MOAD achieves / uses

- **69%** # of deleted lines,
- **2.8X** faster compared to ORBS.

Thank you.