Semantic Slicing
Exploiting Relational Verification

Stephan Gocht, Daniel Lentzsch

stephan.gocht@student.kit.edu
uwejg@student.kit.edu

Thursday 28th July, 2016

Overview

Slicing after Weiser

"Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior." Weiser 1981

Usually Static Backward Slicing:

- same behavior:
  same value for of a variable
  at specific location
- minimal form:
  remove unnecessary
  statements

Listing 1: Minimalistic

```
1  int a = 3;
2  int b = 2;
3  a = a + 1;
```

Slicing after Weiser

"Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior." Weiser 1981

Usually Static Backward Slicing:

- same behavior:
  same value for of a variable
  at specific location
- minimal form:
  remove unnecessary
  statements

Listing 1: Minimalistic

```
1  int a = 3;
2  int b = 2;
3  a = a + 1;
```

Applications of Slicing

- code comprehension
  - refactoring
  - debugging
  - code reuse
- information flow control

Syntactic vs. Semantic Slicing Algorithms

syntactic	semantic
▶ fast ▶ applicable to large real world programs ▶ not precise	▶ increased precision ▶ as hard as program verification

Listing 2: Example - Not Sliced

```
1  const int NO_ERROR = 0;
2  const int NULL_POINTER = -1;
3  const int OUT_OF_BOUND = -2;
4
5  int countOccurrence(int x, int* a, int N) {
6    int result = 0;
7    int err = NO_ERROR;
8    if (a == NULL)
9      err = NULL_POINTER;
10   else
11     for (int i = 0; i < N; i++) {
12       if (0 <= i && i < N) {
13         if (a[i] == x)
14           result++;
15       } else {
16         err = OUT_OF_BOUND;
17       }
18
19       if (err)
20         break;
21     }
22   return err?err:result;
23 }
```

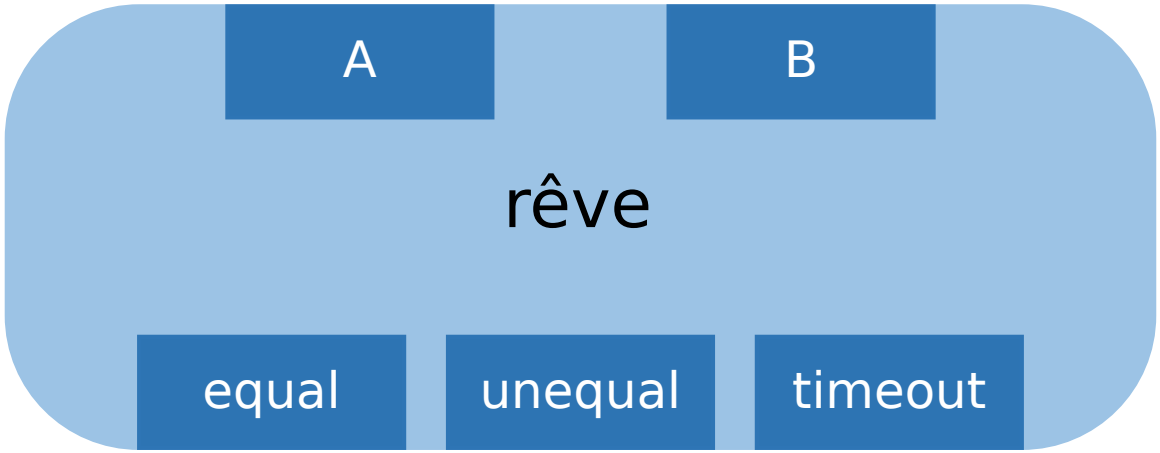
Listing 3: Example - Sliced for Result

```
1  const int NO_ERROR = 0;
2  const int NULL_POINTER = -1;
3  const int OUT_OF_BOUND = -2;
4
5  int countOccurrence(int x, int* a, int N) {
6    int result = 0;
7    int err = NO_ERROR;
8    if (a == NULL)
9      err = NULL_POINTER;
10   else
11     for (int i = 0; i < N; i++) {
12       if (0 <= i && i < N) {
13         if (a[i] == x)
14           result++;
15       } else {
16         err = OUT_OF_BOUND;
17       }
18
19       if (err)
20         break;
21     }
22   return err?err:result;
23 }
```

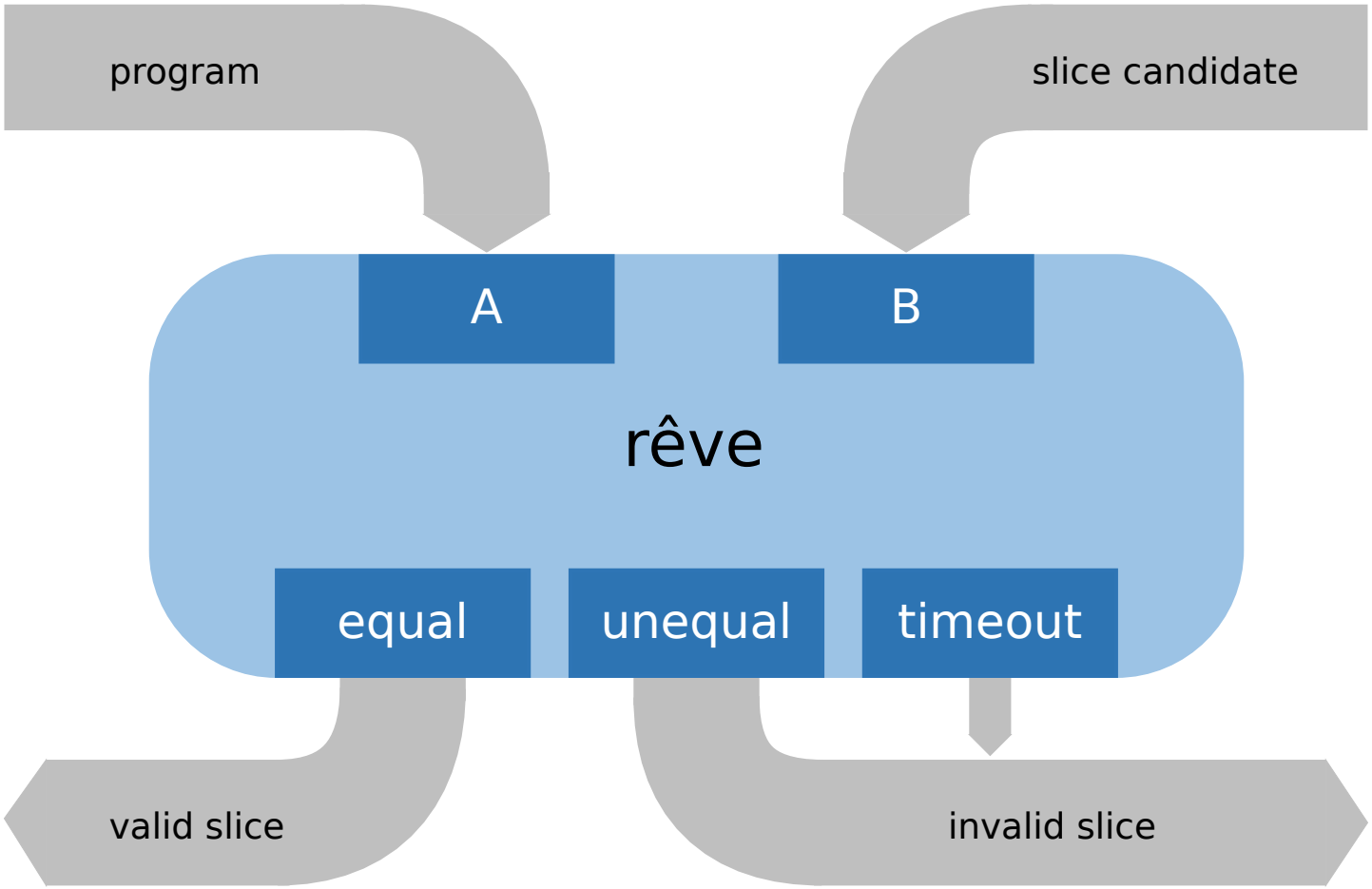
Listing 4: Example - Sliced for Error

```
1  const int NO_ERROR = 0;
2  const int NULL_POINTER = -1;
3  const int OUT_OF_BOUND = -2;
4
5  int countOccurrence(int x, int* a, int N) {
6    int result = 0;
7    int err = NO_ERROR;
8    if (a == NULL)
9      err = NULL_POINTER;
10   else
11     for (int i = 0; i < N; i++) {
12       if (0 <= i && i < N) {
13         if (a[i] == x)
14           result++;
15       } else {
16         err = OUT_OF_BOUND;
17       }
18
19       if (err)
20         break;
21     }
22   return err?err:result;
23 }
```

How To: Exploit Relational Verification for Slicing



How To: Exploit Relational Verification for Slicing



How To: Finding Slice Candidates

- Brute Force
- Impact Analysis for Assignments
- Counter Example Guided Slicing

Brute Force

- test all possible slice candidates
- does not scale!
- precisest results we can get

Listing 5: Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Brute Force

- test all possible slice candidates
- does not scale!
- precisest results we can get

- calls to rêve: 555
- runtime: 85s

Listing 6: Slice of Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Impact Analysis for Assignments

- information flow driven (find assignments without effect)
- always has a valid slice

Listing 5: Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Impact Analysis for Assignments

- information flow driven (find assignments without effect)
- always has a valid slice

Listing 7: Running Example - IAA 1

```
1  int foo(int heigh, int low, int N) {
2    for (int i = HAVOC; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Impact Analysis for Assignments

- information flow driven (find assignments without effect)
- always has a valid slice

Listing 8: Running Example - IAA 2

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if ( i < N - 1)
4        low = HAVOC;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Impact Analysis for Assignments

- information flow driven (find assignments without effect)
- always has a valid slice

Listing 9: Running Example - IAA 3

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = HAVOC;
7    }
8    return low;
9  }
```

Impact Analysis for Assignments

- information flow driven (find assignments without effect)
- always has a valid slice

- calls to rêve: 5
- runtime: < 1s

Listing 6: Slice of Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Counter Example Guided Slicing

- compute simultaneous dynamic slice for counter examples
- fragile in case of timeouts
- depends on quality of counter examples

Listing 5: Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Counter Example Guided Slicing

- compute simultaneous dynamic slice for counter examples
- fragile in case of timeouts
- depends on quality of counter examples

Listing 10: Running Example - CGS 1

```
1  int foo(int heigh=0, int low=0, int N=0) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Counter Example Guided Slicing

- compute simultaneous dynamic slice for counter examples
- fragile in case of timeouts
- depends on quality of counter examples

Listing 11: Running Example - CGS 2

```
1  int foo(int heigh=0, int low=0, int N=1) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Counter Example Guided Slicing

- compute simultaneous dynamic slice for counter examples
- fragile in case of timeouts
- depends on quality of counter examples

Result:

- calls to rêve: 2
- runtime: < 1s

Listing 6: Slice of Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Conclusion

- minimal slicing is undecidable
- existing semantic approaches exploit...
  - symbolic execution
    Jaffar et al. 2012
  - verification conditions
    Barros et al. 2012
  - abstract interpretation
    Halder and Cortesi 2013; Mastroeni and Nikolić 2010

$\rightarrow$ object to handle loops is provided externally

Conclusion

advantage of our approach:

- slice and slice candidate are very similar
- coupling predicates are trivial while programs behave the same
- preliminary result is promising

Conclusion

Listing 5: Running Example

```
1  int foo(int heigh, int low, int N) {
2    for (int i = 0; i < N; i++) {
3      if (i < N - 1)
4        low = heigh;
5      else
6        low = 3;
7    }
8    return low;
9  }
```

Coupling predicates with slice:

$$[(N_1 - N_2 + i_2 - i_1 = 0) \wedge (N_1 - i_1 \geq 1)]$$
$$\vee[(N_1 - N_2 + i_2 - i_1 = 0) \wedge (low_1 = low_2)]$$

Conclusion

- future work
  - real world programs
  - compare to existing approaches
- challenges:
  - increased size
  - heaps

Questions?

References I

Barros, José Bernardo, et al. 2012. “Assertion-based slicing and slice graphs”. *Formal Aspects of Computing* 24 (2): 217–248.

Halder, Raju, and Agostino Cortesi. 2013. “Abstract program slicing on dependence condition graphs”. *Science of Computer Programming* 78 (9): 1240–1263.

Jaffar, Joxan, et al. 2012. “Path-Sensitive Backward Slicing”. In *Proceedings of the 19th International Conference on Static Analysis*, 231–247. Springer.

Mastroeni, Isabella, and Đurica Nikolić. 2010. “Abstract program slicing: From theory towards an implementation”. In *Formal Methods and Software Engineering. 12th International Conference on Formal Engineering Methods, ICFEM 2010. Proceedings*, 452–467. Springer.

References II

Weiser, Mark. 1981. “Program Slicing”. In *Proceedings of the 5th International Conference on Software Engineering*, 439–449. IEEE Press.