

# Specification Slicing for VDM-SL

Tomohiro Oda

Software Research Associates, Inc.

Han-Myung Chang

Nanzan University

Vienna  
Talk



# Agenda

- background 1: explicit operation definitions
- background 2: program slicing
- slice extraction for VDM-SL
- slicing in ViennaTalk
- demo
- summary and future work

# background I: explicit operation definitions

```
1 register : Name ==> Id
2 register(name) ==
3   (dcl i:Id := NextId;
4    NextId := NextId + 1;
5    NameBook(i) := name;
6    return i)
7 post
8   RESULT not in set dom NameBook~
9   and NameBook = NameBook~ munion {RESULT | -> name}
```

The diagram illustrates the structure of an explicit operation definition. It shows the following components:

- signature**: Points to the type declaration `register : Name ==> Id`.
- operation name and params**: Points to the parameter declaration `register(name)`.
- statements**: Points to the sequence of statements: `(dcl i:Id := NextId;`, `NextId := NextId + 1;`, `NameBook(i) := name;`, and `return i`).
- post condition**: Points to the postcondition section starting with `post`, followed by the result constraint `RESULT not in set dom NameBook~` and the assignment `and NameBook = NameBook~ munion {RESULT | -> name}`.

## background 2: (backward static) program slicing

backward static slice = subset of the original source that produce the same result with regard to the value of the particular variable (slicing criterion).

```
(dcl i : Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; return i)
```

## background 2: (backward static) program slicing



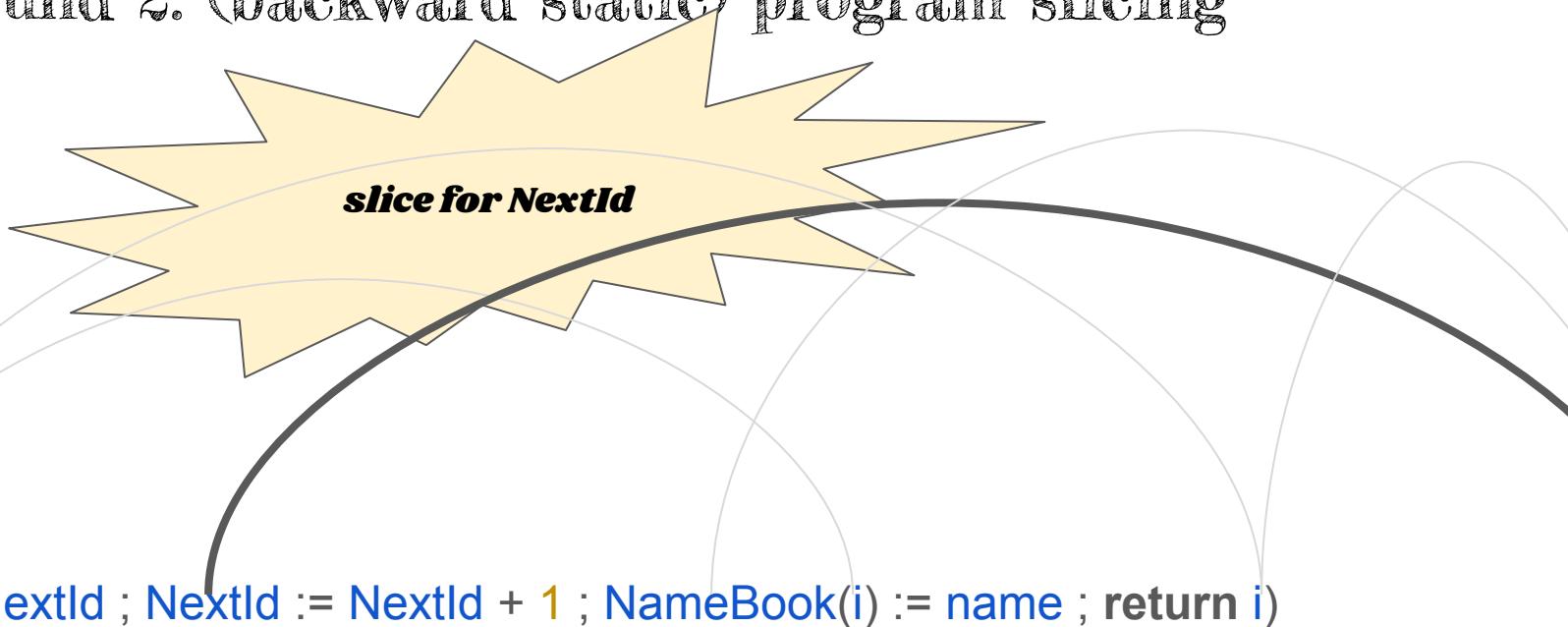
```
(dcl i : Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; return i)
```

## background 2: (backward static) program slicing



```
(dcl i: Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; return i)
```

## background 2: (backward static) program slicing



## background 2: (backward static) program slicing



```
(dcl i: Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; return i)
```

## background 2: (backward static) program slicing

(**dcl** i : Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; **return** i)

(**dcl** i : Id := NextId ; NextId := NextId + 1 ; **NameBook(i) := name** ; return i)  
slice for **NameBook**

(dcl i : Id := NextId ; **NextId := NextId + 1** ; NameBook(i) := name ; return i)  
slice for **NextId**

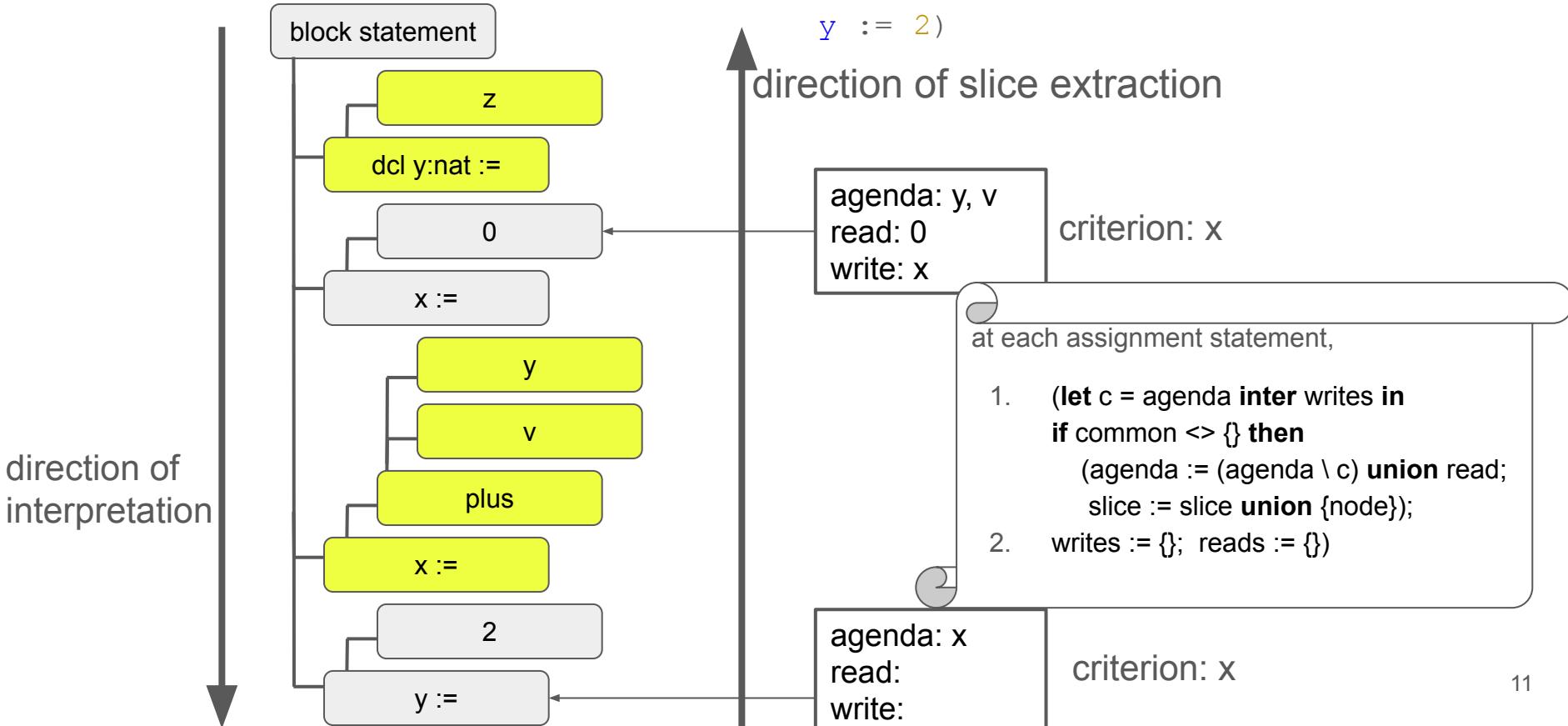
(**dcl** i : Id := NextId ; NextId := NextId + 1 ; NameBook(i) := name ; **return** i)  
slice for **RESULT**

## background 2: (backward static) program slicing

to extract a slice, you trace

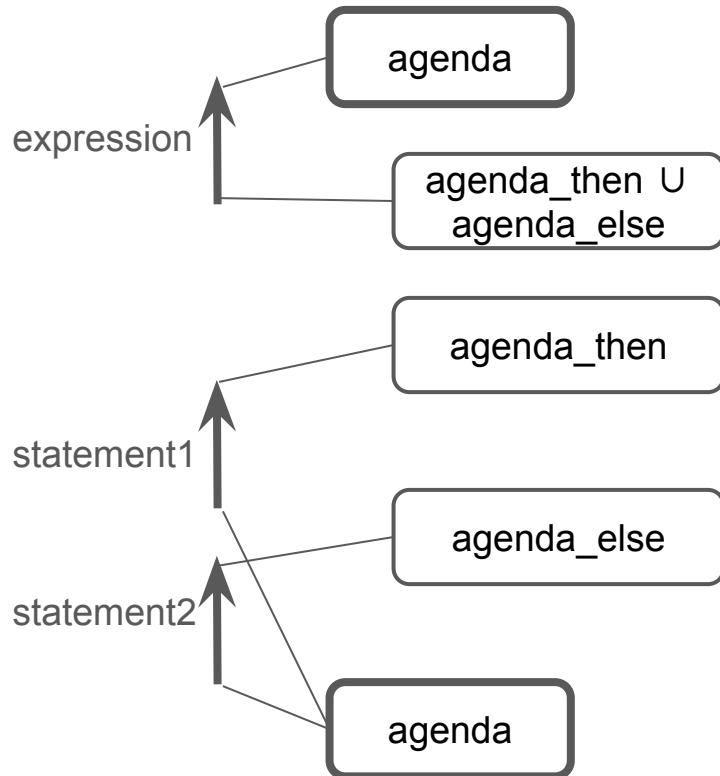
- data dependency : write-read relationship between AST nodes
  - assignment, return, apply expression, ...
- control dependency : conditions of execution
  - if, cases, for, while, ...

# slice extraction for VDM-SL



# conditionals and loops

if



then

statement1

else

statement2

agenda

agenda\_then  
U  
agenda\_else

agenda\_then

agenda\_else

agenda

while

expression

do

statement

agenda

until agenda converges

agenda

# debugging using slicing

```
1 register : Name * [Email] ==> Id
2 register(name, email) ==
3     (dcl i:Id := NextId;
4      NextId := NextId + 1;
5      NameBook(i) := name;
6      if
7          email <> nil
8      then
9          (i := NextId;
10         NextId := NextId + 1;
11         EmailBook(i) := email);
12         return i)
13 post
14     NameBook = NameBook~ munion {RESULT |-> name}
15     and (email = nil and EmailBook = EmailBook~
16           or email <> nil and EmailBook = EmailBook~ munion {RESULT |-> email})
```

slice for the failed assertion

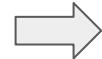
point of error

# Advantage of VDM-SL in slice extraction

- the value semantics of VDM-SL makes slicing easier

Example:

```
(dcl xs:seq of nat := [1,2,3], ys: seq of nat;  
  ys := xs;  
  ys(1) := 0;  
  return xs(1))
```



**RESULT = 1**  
but in the most PLs  
**RESULT = 0**

- no aliasing  $\Rightarrow$  a state variable can be updated only by assignments.
- no hidden states in lower layers or 3rd-party binary modules
- not applicable to VDM++ and VDM-RT because of objects can be aliased.

demo

# Summary and Future Work

- Applied slicing technique to explicit definitions in VDM-SL.
- ViennaTalk provides specification slicing in Browser and Debugger.

## Future Work

- More applications
  - to filter testcases/traces
  - version control
- More specifications
  - implicit operation definitions
  - implicit function definitions

ViennaTalk repository: <https://github.com/tomoooda/ViennaTalk>



A part of this research was supported by JSPS KAKENHI Grant Number JP 23K11058 and Nanzan University Pache Research Subsidy I-A-2 for the 2024 academic year.