Tracking Origins

Andreas Zeller

Today's Topics

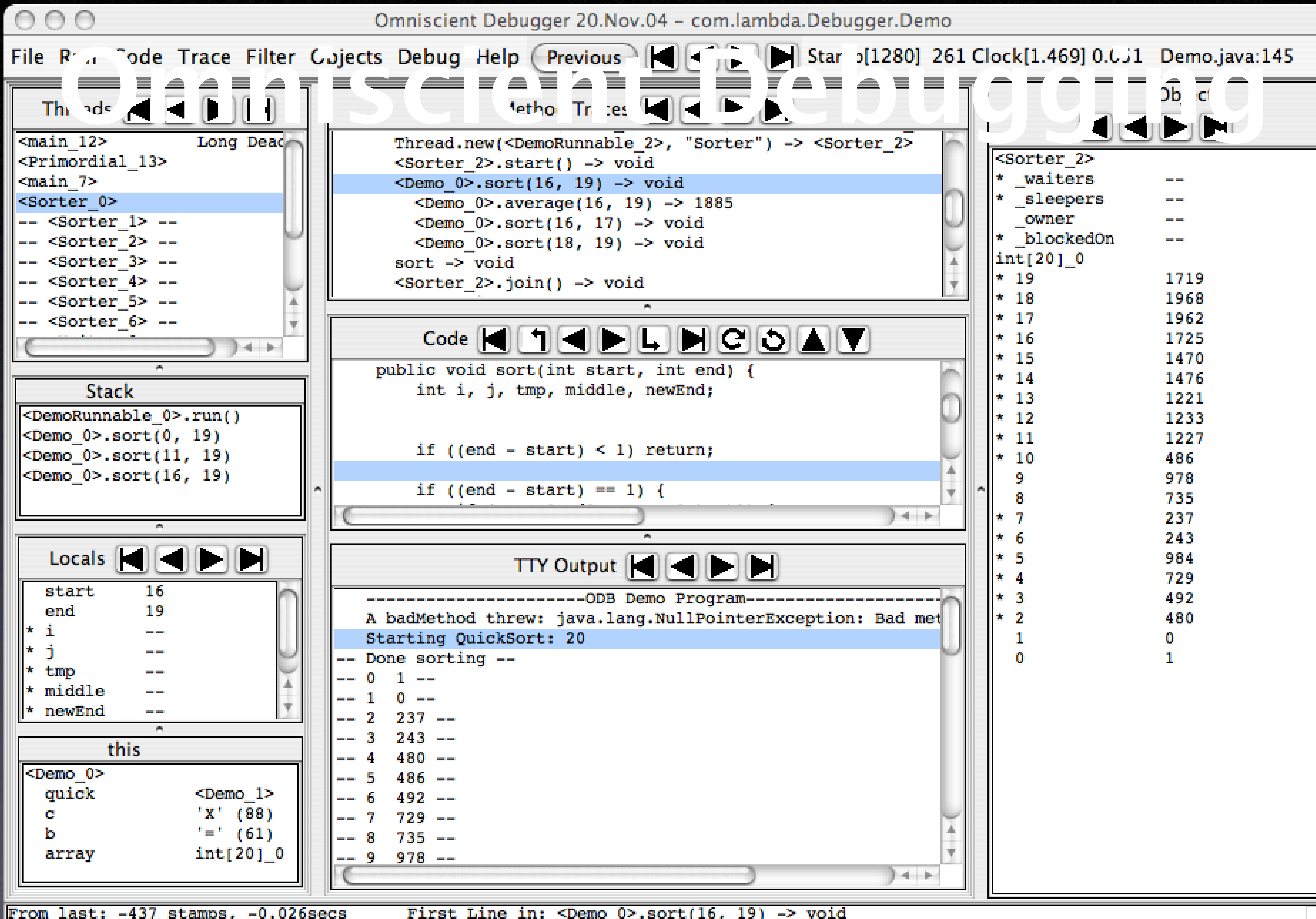
- Exploring History
- Dynamic Slicing
- Leveraging Origins

Exploring the Past

A typical debugging session looks like this:

1. Set a breakpoint
2. Start program, reaching breakpoint
3. Step, Step, Step, …
4. Oops! I’ve gone too far!

Omniscient Debugging



How does it work?

- ODB records a *trace* of the entire execution history
- Slows down programs by a factor of 10
- Records about 100 MB/s
- Now available in commercial tools

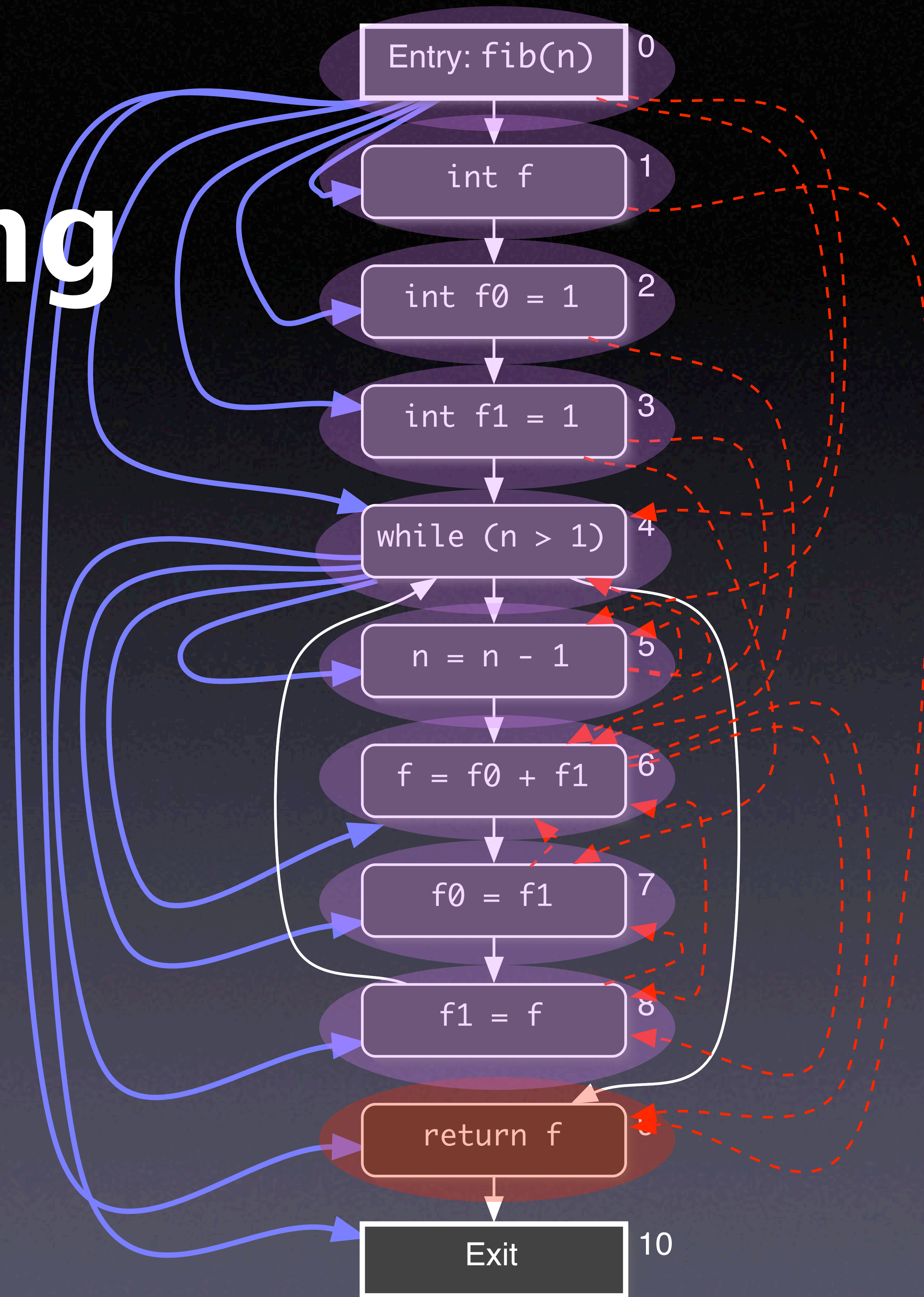
Dynamic Slicing

- Static slices apply to *all* program runs:
  - General + reusable, but imprecise
- A *dynamic slice* applies to *a single run:*
  - Specific and precise

Static Slicing

- Given a statement B, the backward slice contains all statements that could influence the read variables or execution of B
- Formally:

$$S^B(B) = \{A | A \rightarrow^* B\}$$



```
1 n = read();
2 a = read();
3 x = 1;
4 b = a + x;
5 a = a + 1;
6 i = 1;
7 s = 0;
8 while (i <= n) {
9     if (b > 0)
10        if (a > 1)
11            x = 2;
12    s = s + x;
13    i = i + 1;
14 }
15 write(s);
```

Static slice for (s, 15)

```
1 n = read(); // n = 2
2 a = read(); // a = 0
3 x = 1;
4 b = a + x;
5 a = a + 1;
6 i = 1;
7 s = 0;
8 while (i <= n) {
9     if (b > 0)
10        if (a > 1)
11            x = 2;
12    s = s + x;
13    i = i + 1;
14 }
15 write(s);
```

Dynamic slice for (s, 15)

```
n = read();
a = read();
x = 1;
b = a + x;
a = a + 1;
i = 1;
s = 0;
while (i <= n) {
    if (b > 0)
        if (a > 1)
            x = 2;
    s = s + x;
    i = i + 1;
}
write(s);
```

1. Obtain a *trace* of the execution
2. Get the variables that are read and written
3. Assign an empty slice to each written variable
4. Compute the slices from start to end:

$$DynSlice(w) = \bigcup_i (DynSlice(r_i) \cup \{line(r_i)\})$$

Trace	Write	Read	Dynamic Slice
1 n = read();	n		
2 a = read();	a		$DynSlice(w) = \bigcup_i (DynSlice(r_i) \cup \{line(r_i)\})$
3 x = 1;	x		
4 b = a + x;	b	a, x	2, 3
5 a = a + 1;	a	a	2
6 i = 1;	i		
7 s = 0;	s		
8 while (i <= n) {	p8	i, n	6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 8, 6, 1
12 s = s + x;	s	s, x, p8	7, 3, 8, 6, 1
13 i = i + 1;	i	i, p8	8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 13, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 13, 8, 6, 1
12 s = s + x;	s	s, x, p8	12, 7, 3, 6, 8, 1, 13
13 i = i + 1;	i	i, p8	13, 8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
15 write(s);	o15	s	12, 7, 3, 6, 8, 1, 13

Trace	Write	Read	Dynamic Slice
1 n = read();	n		
2 a = read();	a		
3 x = 1;	x		
4 b = a + x;	b	a, x	2, 3
5 a = a + 1;	a	a	2
6 i = 1;	i		
7 s = 0;	s		
8 while (i <= n) {	p8	i, n	6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 8, 6, 1
12 s = s + x;	s	s, x, p8	7, 3, 8, 6, 1
13 i = i + 1;	i	i, p8	8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 13, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 13, 8, 6, 1
12 s = s + x;	s	s, x, p8	12, 7, 3, 6, 8, 1, 8, 13
13 i = i + 1;	i	i, p8	13, 8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
15 write(s);	o15	s	12, 7, 3, 6, 8, 1, 13

Trace	Write	Read	Dynamic Slice
1 n = read();	n		
2 a = read();	a		
3 x = 1;	x		
4 b = a + x;	b	a, x	2, 3
5 a = a + 1;	a	a	2
6 i = 1;	i		
7 s = 0;	s		
8 while (i <= n) {	p8	i, n	6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 8, 6, 1
12 s = s + x;	s	s, x, p8	7, 3, 8, 6, 1
13 i = i + 1;	i	i, p8	8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
9 if (b > 0)	p9	b, p8	4, 2, 3, 13, 8, 6, 1
10 if (a > 1)	p10	a, p9	5, 2, 9, 4, 2, 3, 13, 8, 6, 1
12 s = s + x;	s	s, x, p8	12, 7, 3, 6, 8, 1, 8, 13
13 i = i + 1;	i	i, p8	13, 8, 6, 1
8 while (i <= n) {	p8	i, n	13, 8, 6, 1
15 write(s);	o15	s	12, 7, 3, 6, 8, 1, 13

```
1 n = read();
2 a = read();
3 x = 1;
4 b = a + x;
5 a = a + 1;
6 i = 1;
7 s = 0;
8 while (i <= n) {
9     if (b > 0)
10        if (a > 1)
11            x = 2;
12    s = s + x;
13    i = i + 1;
14 }
15 write(s);
```

Static slice for (s, 15)

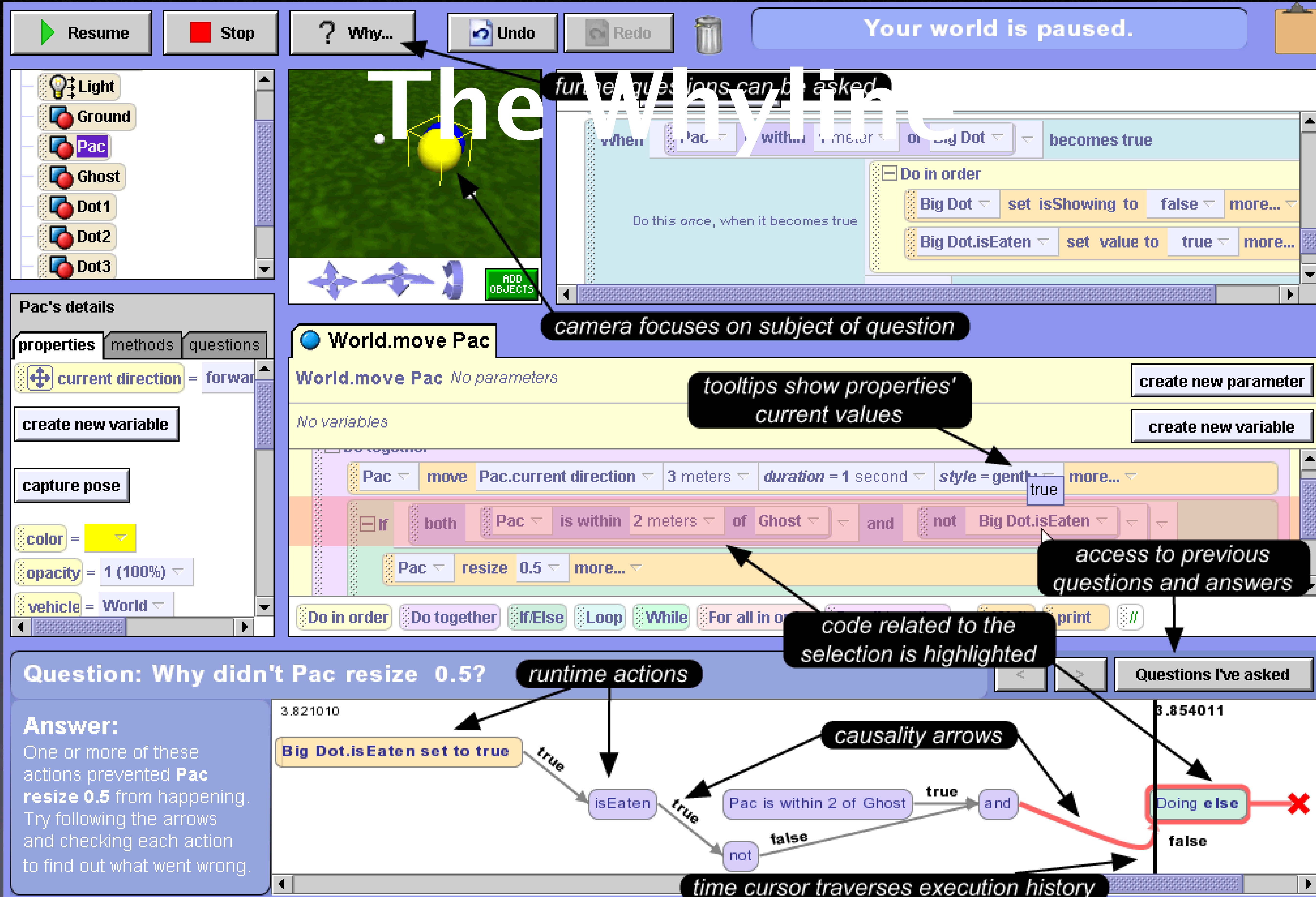
```
1 n = read(); // n = 2
2 a = read(); // a = 0
3 x = 1;
4 b = a + x;
5 a = a + 1;
6 i = 1;
7 s = 0;
8 while (i <= n) {
9     if (b > 0)
10        if (a > 1)
11            x = 2;
12    s = s + x;
13    i = i + 1;
14 }
15 write(s);
```

Dynamic slice for (s, 15)

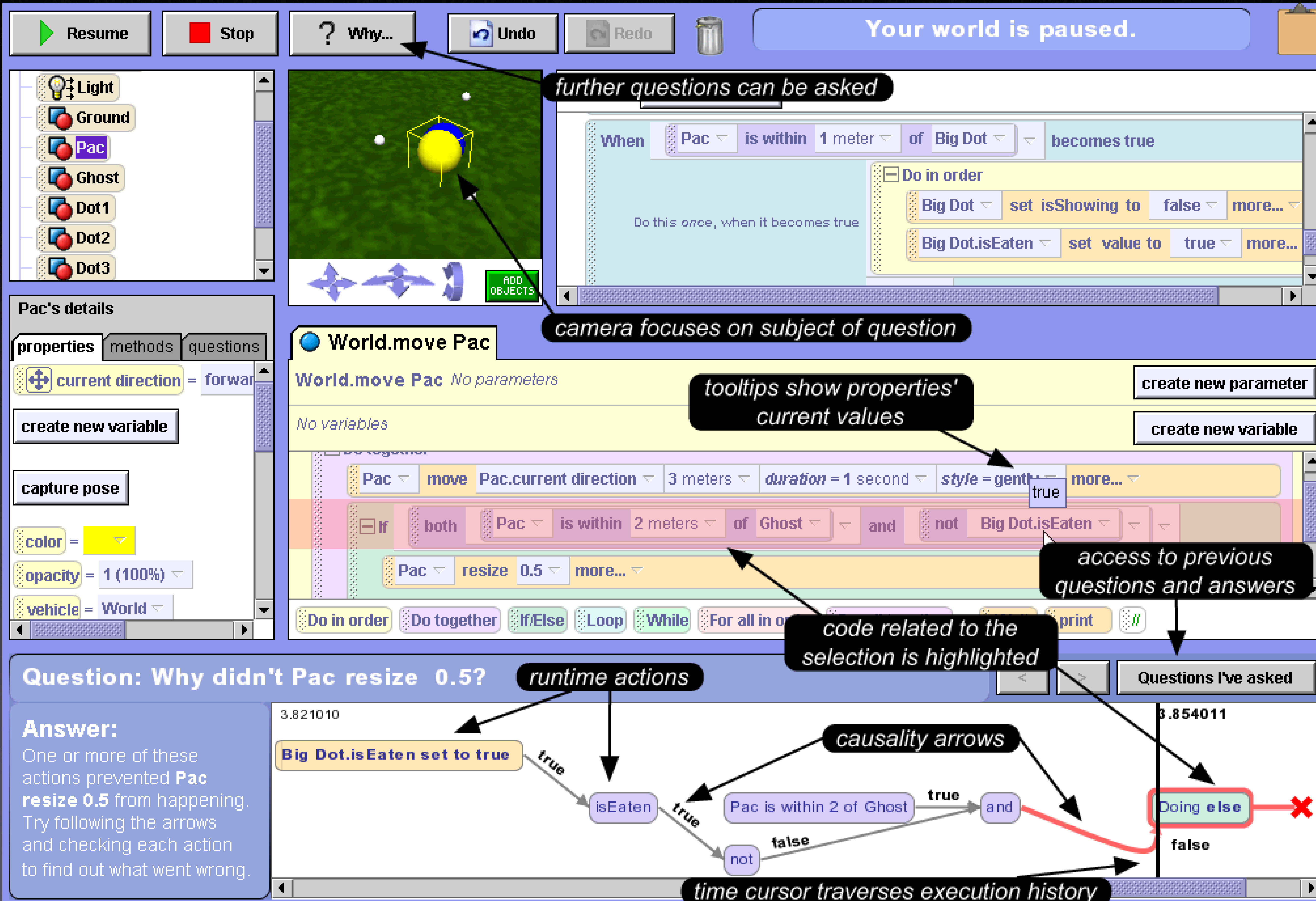
Discussion

- Dynamic slices are much more precise than static slices (applied to the one run, that is)
- From some variable, a backward slice encompasses on average
  - 30% of the *entire* program (static slice)
  - 5% of the *executed* program (dynamic slice)
- Overhead as in omniscient debugging

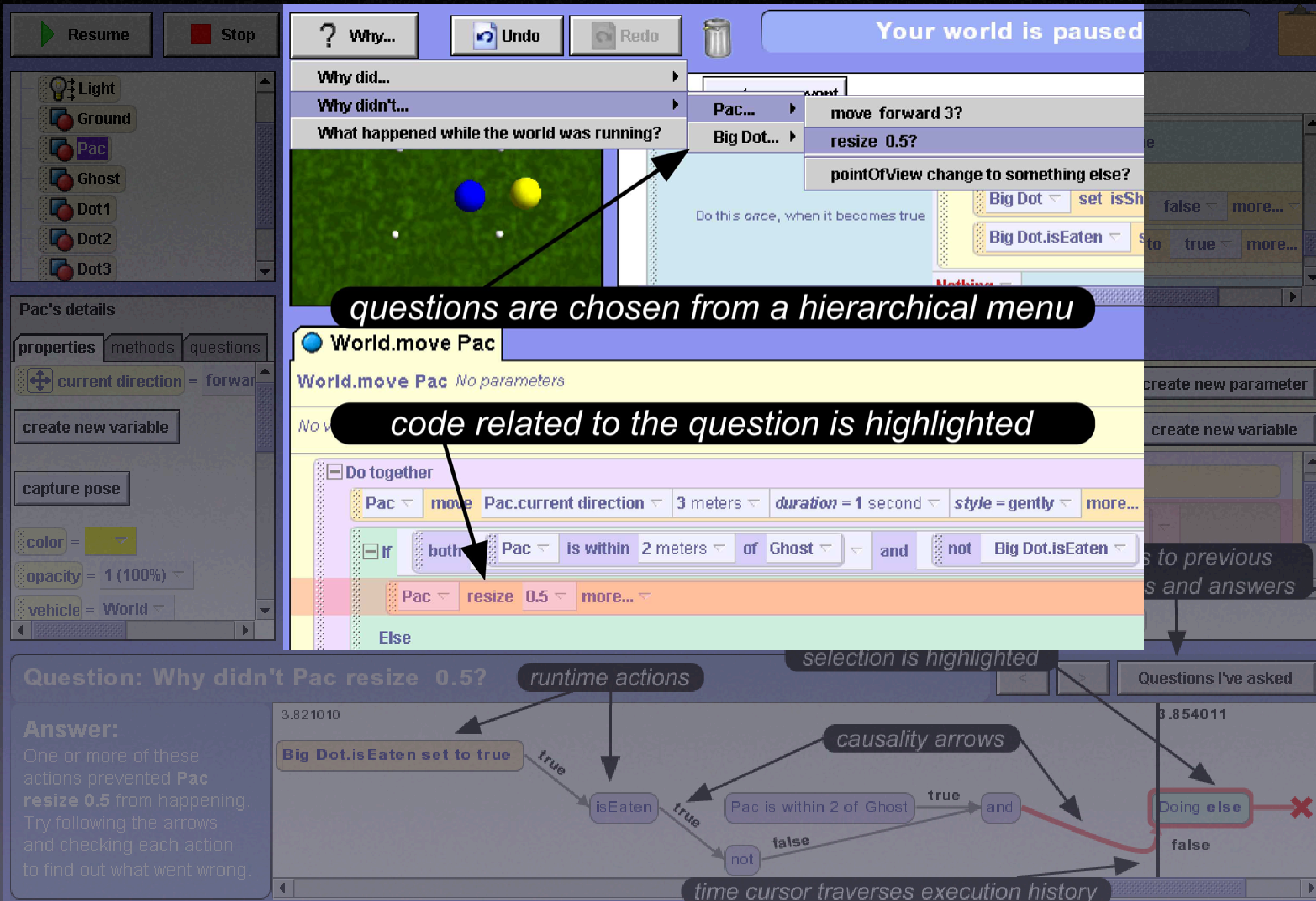
The Whyline



Resume
Stop
Why...
Undo
Redo
Your world is paused.
further questions can be asked
camera focuses on subject of question
World.move Pac
tooltips show properties' current values
access to previous questions and answers
code related to the selection is highlighted
Question: Why didn't Pac resize 0.5?
runtime actions
Questions I've asked
Answer:
One or more of these actions prevented Pac resize 0.5 from happening. Try following the arrows and checking each action to find out what went wrong.
Big Dot.isEaten set to true
causality arrows
isEaten
Pac is within 2 of Ghost
and
not
Doing else
time cursor traverses execution history



Resume
Stop
Why...
Undo
Redo
Your world is paused.
further questions can be asked
camera focuses on subject of question
tooltips show properties' current values
access to previous questions and answers
code related to the selection is highlighted
runtime actions
causality arrows
time cursor traverses execution history
Pac's details
properties
methods
questions
create new variable
capture pose
World.move Pac
No parameters
No variables
create new parameter
When Pac is within 1 meter of Big Dot becomes true
Do this once, when it becomes true
Do in order
Big Dot set isShowing to false more...
Big Dot.isEaten set value to true more...
Pac move Pac.current direction 3 meters duration = 1 second style = gentl more...
If both Pac is within 2 meters of Ghost and not Big Dot.isEaten
Pac resize 0.5 more...
Question: Why didn't Pac resize 0.5?
Questions I've asked
Answer:
One or more of these actions prevented Pac resize 0.5 from happening. Try following the arrows and checking each action to find out what went wrong.
3.821010
3.854011
Big Dot.isEaten set to true
isEaten
not
Pac is within 2 of Ghost
and
Doing else
true
false


Resume
Stop
Why...
Undo
Redo
Your world is paused
Why did...
Why didn't...
What happened while the world was running?
Pac...
Big Dot...
move forward 3?
resize 0.5?
pointOfView change to something else?
questions are chosen from a hierarchical menu
code related to the question is highlighted
World.move Pac
World.move Pac No parameters
Do together
Pac move Pac.current direction 3 meters duration = 1 second style = gently more...
If both Pac is within 2 meters of Ghost and not Big Dot.isEaten
Pac resize 0.5 more...
Else
Pac's details
properties methods questions
create new variable
capture pose
create new parameter
Question: Why didn't Pac resize 0.5?
runtime actions
selection is highlighted
Questions I've asked
Answer:
One or more of these actions prevented Pac resize 0.5 from happening. Try following the arrows and checking each action to find out what went wrong.
3.821010
3.854011
Big Dot.isEaten set to true
isEaten
not
Pac is within 2 of Ghost
and
Doing else
causality arrows
time cursor traverses execution history

“Why did” questions

- Take the dynamic slice of the variable
- Follow at most two dependencies
- If programmer wants, follow dependencies transitively

“Why did s = 2 in Line 15?”

```
1 n = read(); // n = 2
2 a = read(); // a = 0
3 x = 1;
4 b = a + x;
5 a = a + 1;
6 i = 1;
7 s = 0;
8 while (i <= n) {
9     if (b > 0)
10        if (a > 1)
11            x = 2;
12    s = s + x;
13    i = i + 1;
14 }
15 write(s);
```

“Because s = 1 and i = 2”

“Why didn’t” questions

- Follow back control dependencies to closest controlling statement(s)
- Do a “why did” question on each
- Again, follow at most two dependencies

“Why didn’t x = 2 in Line 11?”

```
 1 n = read(); // n = 2
 2 a = read(); // a = 0
 3 x = 1;
 4 b = a + x;
 5 a = a + 1;
 6 i = 1;
 7 s = 0;
 8 while (i <= n) {
 9     if (b > 0)
10         if (a > 1)
11             x = 2;
12     s = s + x;
13     i = i + 1;
14 }
15 write(s);
```

“Because a = 1 and b = 1”

Discussion

The WHYLINE combines

- omniscient debugging
- static slicing
- dynamic slicing

in an attractive package, showcasing the state of the art in interactive debugging

Tracking Infections

1. Start with the infected value as seen in the failure
2. Follow back the dependencies
3. Observe and judge origins – are they sane?
4. If some origin is infected, repeat at Step 2
5. All origins are sane? Here's the infection site!

Concepts

- Omniscient debugging allows for simple exploration of the entire execution history
- Dynamic slicing tells the origin of a value
- To track down an infection, follow dependencies and observe origins, repeating the process for infected origins