CS162 Week 5

Kyle Dewey

Overview

- `atomic` **blocks**
- More examples
- Question / answer session

RIP Refresher

newCons

```
var a in
   a := 0;
   newCons {
      output a // `a` is reactive
   };
   while (a < 10) {
      a := a + 1 // trigger `output`
   }
```

Output:

0

1

...

10

atomic **Blocks**

Problem

- We need to update a variable multiple times during a loop
- The computation is not “done” until the last assignment
- We want to update only when the computation is done

Example

```
var a in
   a := 0;
   newCons {
      output a
   };
while (a < 11) {
   a := a + 3
};
a := a + a // now `a` is ready
```

Output:

0

3

6

9

12

24

Hacky Solution

- Add a flag `isDone`
  - Set to `false` beforehand
  - Set to `true` when a constraint is ready
  - In the constraint, only process if `isDone` is `true`

Better Solution

- Let the language handle it
- Introduce a special `atomic` block
  - Constraints are only updated once we leave the `atomic` block
  - Instead of having multiple updates of the same constraint, only update the constraint once at the end

With atomic

```
var a in
   a := 0;
   newCons {
      output a
   };
   atomic {
      while (a < 11) {
         a := a + 3
      };
      a := a + a // now `a` is ready
   }
```

Output:
0
24

Nesting atomic

```
var a, b in
   newCons {
      output b
   };
   newCons {
      output a
   };
   atomic {
      a := 2;
      atomic {
         b := 4
      };
      a := 3
   }
```

Output:

undef

undef

3

4

More Examples

What's the Output?

```
var b in

b:=4;
atomic {
  newCons {
    output b
  };
  atomic {
    b := b+1
  }
};
b := 10
```

What's the Output?

```
var b in

b:=4;
atomic {
  newCons {
    output b
  };
  atomic {
    b := b+1
  }
};
b := 10
```

Output:

4

5

10

What's the Output?

```
var a, b in
   a := 0;
   b := 0;
   newCons {
      output a;
      output b
   };
   atomic {
      a := a + 1;
      atomic {
         b := b + 1;
      }
   }
```

What's the Output?

```
var a, b in
   a := 0;
   b := 0;
   newCons {
      output a;
      output b
   };
   atomic {
      a := a + 1;
      atomic {
         b := b + 1;
      }
   }
```

Output:

0

0

|

|

|

What's the Output?

```
var a in
   a := 0;
   newCons {
      output a;
   };
   atomic {
      a := a + 1;
      atomic {
         a := a + 1;
      }
   }
```

What's the Output?

```
var a in
   a := 0;
   newCons {
      output a;
   };
   atomic {
      a := a + 1;
      atomic {
         a := a + 1;
      }
   }
```

Output:
0
2

test52.not

Question / Answer

Question

- Is it possible to switch from atomic mode to constraint mode without exiting an atomic block?

Answer

- Yes

```
var a in // normal mode
   a := 0;
   atomic { // atomic mode
      newCons { // constraint mode
         output a
      }
   };
   a := 10
```

Question

- After the constraint stack is implemented, does the `cSelf` field of `ConstraintMode` have any use?

Answer Part #1

- Yes
- `cSelf` indicates the constraint you are **currently executing**
- The constraint stack indicates how deeply nested you are within `newCons` blocks
- These are not exactly the same

Answer Part #2

```
var a in
  newCons { // pushes `a := a + 1`
    a := a + 1
  }; // pops `a := a + 1`
  a := 5 // doesn't push anything
```

Question

- Should all addresses that have been updated be in an atomic block be pushed onto the atomic stack?

Answer

- Only addresses that are reactive at the time of the update should be pushed
- If the address is not reactive at the time but becomes reactive before the end of the atomic block, this can result in constraints getting executed that should not be
- See `test52.not`