User Evaluation of Correctness Conditions: A Case Study of Cooperability

Caitlin Sadowski and Jaeheon Yi

What is the semantics of x++?

What is the semantics of x++?

sequential semantics

simple increment

What is the semantics of X++?

sequential semantics

simple increment

preemptive semantics

potentially non-atomic
read-modify-write

What is the semantics of X++?

sequential semantics

simple increment

preemptive semantics

potentially non-atomic read-modify-write

```
t:=x
t:=t+1
x:=t
```

What is the semantics of x++?

sequential semantics

simple increment

preemptive semantics

potentially non-atomic read-modify-write

```
t:=x
t:=t+1
x:=t
```

```
x:=1
x:=2
x:=3
x:=4
```

context switches anytime!

thread interference!

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic read-modify-write

x:=1

t:=x

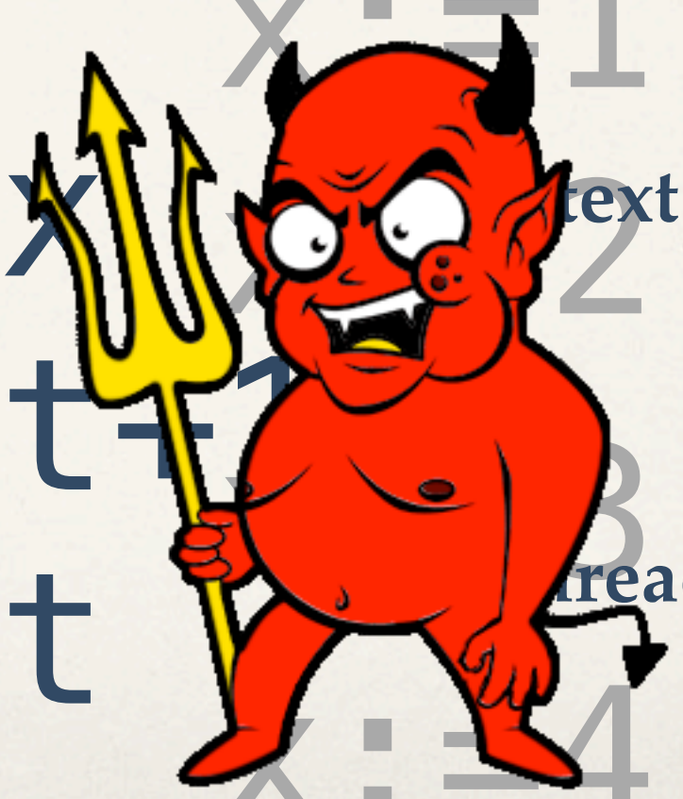
x:=2

t:=t+1

x:=3

x:=t

x:=4



text switches anytime!

read interference!

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic read-modify-write

```
t:=x
t:=t+1
x:=t
```

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic read-modify-write

```
yield
t:=x
t:=t+1
x:=t
yield
```

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic
read-modify-write

cooperative semantics

x++ ***always***
a simple increment

interference at **`yield`** annotations

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic
read-modify-write

cooperative semantics

x++ ***always***
a simple increment

interference at **`yield`** annotations

What is the semantics of X++?

Good!

Bad!

sequential semantics

simple increment

preemptive semantics

potentially non-atomic
read-modify-write

cooperative semantics

x++ ***always***
a simple increment

Cooperable

behaves **as if** ***running with cooperative semantics***

interference at **`yield`** annotations

A **method** of developing multithreaded programs (Yi & Flanagan, TLDI 2010)

program is cooperable

A **method** of developing multithreaded programs (Yi & Flanagan, TLDI 2010)

program is cooperable $\wedge$ correct under cooperative semantics

A **method** of developing multithreaded programs (Yi & Flanagan, TLDI 2010)

program is cooperable $\wedge$ correct under cooperative semantics $\Rightarrow$ correct under preemptive semantics

A **method** of developing multithreaded programs (Yi & Flanagan, TLDI 2010)

program is cooperable $\wedge$ correct under cooperative semantics $\Rightarrow$ correct under preemptive semantics

- Does cooperability make multithreaded programming easier?

Does cooperability help programmers identify bugs?

- H1: Participants have an easier time finding bugs in cooperable code, i.e., with correct yield annotations.
- H2: Participants have a higher subjective impression of whether they understand code with correct yield annotations, as compared to code without such annotations.
- H3: Expert participants issue higher quality bug reports, as compared with novice participants.

Methodology

- Online survey (15 to 90 minutes)
- Three code examples
- Between-group study
  - with annotations (13)
  - without annotations (15)

Bug reports coded with A/B/C scale

- A: found concurrency bug
  - “sb can be modified by other threads, so the value len could be out of date by the time we do the getChars call.”
- B: found other problems
  - “weird return in append. index is never updated.”
- C: confused / wrong / ”no idea” etc.
  - “I have no idea what that yield statement is doing there.”

H1: Yield annotations **did** help participants find bugs

	A	B	C	Total
Yield Annotations	30	3	3	36
No Yields	17	6	21	44

- Total difference is statistically significant ($p < .001$)
- For 2/3 individual code samples, difference is statistically significant

H2: Subjective impressions largely unaffected by yields

- In 2/3 code samples ***no*** significant difference in rating
  - (a) I understand this code
  - (b) I have identified any bugs in the code
- In 1/3 code sample difference for (b)
  - 1.8 vs 2.2 (Likert scale)
  - statistically significant

Most participants found annotations helpful

- “I hadn't been thinking about how the synchronized command didn't protect the contents of passed in parameters from being modified.”

- “I just started looking for the yields and that made it almost trivial to pick out the issues.”

Yield annotations not a panacea

- “They pointed out places for potential trouble. However, I could not, in all cases, identify what the trouble in that spot might be”

- “"yield" doesn't mean anything in particular to me in this context. "Oh noes!" would work just as well.”

H3: failed to reject null hypothesis

- **Novice**: Occasional experience with multithreaded code
- **Expert**: Frequent experience with multithreaded code

- Found ***no*** statistically significant differences between "Novices" and "Experts"
  - Topic of future work

Future work

- improving yield annotations
  - discount usability
  - yield annotations and mined bugs
  - explaining cooperability
- how do users debug concurrent programs?
  - mental models

Conclusions

- Small, portable, survey with statistically significant results
  - Overall, yield annotations **were** helpful
  - However, concept not as easy as we thought it was