

An inconsistent and complete programming language

Yesno is a programming language which is:

- object-oriented. Strings, integers, classes, methods (closures), and even control constructs (e.g., branches) are objects and are controlled through message-passing;
- dynamic (e.g., classes are defined by subclassing and adding methods at run-time);
- complete (i.e., every well-defined program yields a value);
- ultimately consistent (i.e., if a program would halt as by consistent semantics, then eventually yesno will yield the correct result and only the correct result).

Michael J. Burrell

mburrell@uwo.ca

Ph.D. candidate, Computer Science,
University of Western Ontario

What is y?

```
z := 5.
if cond: somePredicate
then: [| z := 6. |]
else: [| z := 7. |].
y := z + 2.
```

Initial inconsistent result

y ← {8, 9}

somePredicate returns true

Ultimate result

y ← {8}

What if somePredicate doesn't halt?

In the case that a program doesn't halt, or a portion of a program doesn't halt, we still have an immediate result. This result is inconsistent—it has some wrong answers in it—but the answers are meaningful in the context of the program.

Theory

Gödel's Incompleteness Theorem proved that a substantial logic must either be complete or consistent, but it cannot be both. When designing programming languages, we have a similar dilemma: a Turing-complete language must either offer consistent semantics, or allow that some programs do not halt. All languages to date have chosen consistent semantics.

Yesno is a language that has complete semantics: even though yesno is universally powerful, every yesno program yields a result. The trade-off is

that the results are inconsistent. Multiple results can be returned. For a decision problem, consider that a program might yield both true **and** false.

```
PunkRaw:~/Source/yesno mike$ cat hailstone.yesno
Integer addInstanceMessage: "isEven" withMethod: [:
(self / 2) * 2 = self
].
Closure addInstanceMessage: "haltsWith:" withMethod: [ x |
halts := false.
self doWith: x.
halts := true.
return halts.
].
hailstone := [ n :
if cond: (n = 1) then: [: 1 ]
else: [: if cond: (n isEven) then: [:
hailstone doWith: (n / 2)
] else: [:
hailstone doWith: ((3 * n) + 1)
]]
].
z := 2.
return if cond: (hailstone haltsWith: 7)
then: [: z * 3 ] else: [: z + 1 ].
PunkRaw:~/Source/yesno mike$ ./yesno <hailstone.yesno | tail -6
==== {3, 6}
==== {3, 6}
==== {6}
==== {6}
==== {6}
execution has now finished; displaying results
==== {6}
```

Execution in yesno is actually incremental. Initially all possible results are returned. As computation continues, the set of possible values is "narrowed down"

and, if the program were to halt as by consistent semantics, eventually only the correct result will be returned.

Inconsistency

Yesno is a pervasively object-oriented language—all execution is done via message-passing. Thus, to introduce inconsistency and completeness, we change the se-

mantics of sending a message. We consider the case where:

- the message has been sent; and
- where the message has not been sent.

We consider these two possibilities simultaneously; each yields different values for the variables, and the ability for a variable to take in multiple values is what introduces the inconsistency. Obviously if we consider that a message has not been sent, then every program must halt, since message-passing is the basis for all iteration and recursion.

Once a method has finished completely, we abandon the inconsistent states and consider only the state where the method finished successfully. Thus, if the program halts, eventually we get only the correct answer; if it never halts, we get some inconsistent results, which may change with time.

Download

<http://www.csd.uwo.ca/~mburrell/yesno/>

