Why do we care?
A simple problem whose decidability is not known

I am preparing for a talk aimed at undergraduate math majors, and as part of it, I am considering discussing the concept of decidability. I want to give an example of a problem that we do not currently know to be decidable or undecidable. There are many such problems, but none seem to stand out as nice examples so far.

What is a simple-to-describe problem whose decidability is open?
The Matrix Mortality Problem for 2x2 matrices. I.e., given a finite list of 2x2 integer matrices $M_1,...,M_k$, can the $M_i$'s be multiplied in any order (with arbitrarily many repetitions) to produce the all-0 matrix?

(The 3x3 case is known to be undecidable. The 1x1 case, of course, is decidable.)
UPDATE: The problem I mentioned here is now known to be undecidable! http://arxiv.org/abs/1605.05274 Moreover, the paper was inspired by reading this very answer. :)

Programmers in your math-major audience may be surprised to learn that the question "is this type implicitly convertible to that type?" is not known to be decidable in any of Java 5, C# 4 and Scala 2.

For more details, see Andrew Kennedy and Benjamin Pierce's paper "On Decidability of Nominal Subtyping with Variance". The paper gives some examples of additional restrictions to the type systems of these languages, under which nominal subtyping becomes known to be decidable or known to be undecidable.

Interestingly, the paper was written well before generic covariance and contravariance were added to C#, but the authors correctly anticipated the direction the language was heading. (This is unsurprising; the authors designed the underlying support for variance in the CLR that I took advantage of when adding variance to C#! They did the heavy lifting.)

share cite edit flag

answered Sep 4 '13 at 21:29

Eric Lippert
651 • 4 • 4
What about Java? We don’t know (see this paper, for example). Some people suspect that subtyping is undecidable for Java, but this has not been proved to this day. Is this a practical question? Oh, yes it is. Some people want to add reified generics to Java, which means that `foo instanceof Bar` would be checking whether the run-time type of `foo` is a subtype of `Bar`. And if there’s no algorithm that can decide that, we are in trouble: `instanceof` will hang sometimes.

That’s why it’s important to study type systems of main-stream languages. And it’s a pity that so few researchers do.
Formal Language Recognition with the Java Type Checker

Yossi Gil\textsuperscript{1} and Tomer Levy\textsuperscript{2}

1 Department of Computer Science, The Technion—Israel Institute of Technology, Haifa, Israel.
2 Department of Computer Science, The Technion—Israel Institute of Technology, Haifa, Israel.

"JAVA generics are 100\% pure syntactic sugar, and do not support meta-programming."

---
distinguished at ECOOP 2016
But . . .
Java is messy!
On Decidability of Nominal Subtyping with Variance

Andrew J. Kennedy
Microsoft Research Cambridge

Benjamin C. Pierce
University of Pennsylvania

Abstract
We investigate the algorithmics of subtyping in the presence of nominal inheritance and variance for generic types, as found in Java 5, Scala 2.0, and the .NET 2.0 Intermediate Language. We prove that the general problem is undecidable and characterize

22], and the declaration-site variance of Scala [18], algorithmic subtyping rules have never been presented, and decidability is still an open problem.1 This is the starting point for our investigation.

Language features Each of these languages supports generic inheritance, in which a named class is declared with type parameters
Problem

Input
- infinite directed graph
- two distinguished vertices $t$ and $t'$

Output
Is there a finite path $t \rightarrow t'$ in the given graph?
Vertices: Types

\[ f(\ g \ x, \ h \ () \) \]
Vertices: Types

\[ f^2(g^1(x), h^0()) \]
Vertices: Types

f g x h
Vertices: Types

- $x, y, \ldots$ are variables
- $C, D, \ldots$ have arity 1
- $Z$ has arity 0
- Example vertex/type: $CDCCDEZ$
- $s, t, \ldots$ are meta-variables, ranging over types
the inheritance rule

\[ Cx <:: t(x) \]

means that, for all ground types \( s \), there is an arc

\[ Cs \rightarrow t(s) \]
Inheritance Rule

The inheritance rule

```
Cx <:: t(x)
```

means that, for all ground types \( s \), there is an arc

```
Cs \rightarrow t(s)
```

class table = set of inheritance rules
if 

\( s \sim t \)

then

\( Ct \rightarrow Cs \)
Theorem

*It is undecidable whether* $t \sim t'$ *according to a given* unambiguous *class table.*
Theorem

It is undecidable whether $t \sim t'$ according to a given *unambiguous* class table.

Turing machine

class table

input tape

subtype query

type checker

Y/N/?

applies to Java!
Theorem

It is undecidable whether $t \sim t'$ according to a given *unambiguous* class table.
eof