Figure 1: Most of the reasons for taking the class fall under type systems.

Comp 411 Notes - 12 Jan 2005
COURSE CLASSIFICATION
Reproduced by Seth James Nielson (sethn@cs.rice.edu)

1 Why Take Comp 411?
Class members suggested a number of reasons for taking this class. The two
broad categories are Theory and Type Systems. All of the other suggested
reasons were grouped under Type Systems. One significant reason for Type
Systems is that they can be used to provide guarantees about some aspect
of a program (the most typical guarantee provided by a type system in
mainstream languages is type safety). See figure 1.

2 Type Systems and Semantics
This course is Programming Language Semantics. What is the relationship
between semantics and type systems?

1. We study type systems\(^1\) to create, analyze, etc. guarantees.

\(^1\)A language with a type system has types by definition.
2. There are other methods (besides type systems) to provide guarantees about programs. These include Model Checking, Theorem Proving, and Monitoring. Type systems are a subtype of the more general category of Static Analysis (see figure 2). Type systems, unlike any of the other methods for providing guarantees, are expected to be statically decided\(^2\) and to be “quick” in execution\(^3\).

Note that type systems are conservative approximations of “good” programs. Some programs that are “good” are not type safe. The domain of type safety is efficiently provably good programs. See fig 3.

So, because of the properties/advantages of type systems, in this class, we will study building type systems on top of language semantics to provide guarantees about some aspect of programs written in that language.

3 Providing Guarantees

Type systems and typed languages are built on top of untyped languages. Multiple typed languages could be derived from the same untyped language.

\(^2\)Not everyone agrees on this. Cardelli proposed a type system that was undecidable in theory, but (according to him) effective in practice

\(^3\)Quick in practice. Hindley-Milner’s type system is linear in practice but worse than exponential in theory (edge cases). This “requirement” for quick execution can be tricky to achieve
Figure 3: Type systems exclude some “good” programs. They even may exclude some provably good programs.

Figure 4: Multiple typed languages from a single untyped language.

See fig 4.

Type systems are built on top of an untyped language to provide certain guarantees. This construction is displayed in figure 5 and described below.

- **Syntax** Described by a context-free grammar such as BNF.

- **Semantics** Provided by either an interpreter or a compiler. In this class, we will generally use an interpreter because it is a simple definition of implementation. Compilers focus on efficient execution, a subject beyond the scope of this course.

- **Type Systems** Are built on top of syntax and semantics. They are described by a context-sensitive grammar.

- **Guarantees** Provable based on the type system.
<table>
<thead>
<tr>
<th>Guaranettes</th>
<th>Type Systems</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
</table>

Figure 5: Building a language with guarantees.

4 Administrative