Automatically grading excercises on formal languages.

Most automatic grading of student excercises can grouped into two categories. Some excercises are simple enough that their solutions can easily be checked mechanically, like “calculate 2+2”. Another example, from the field of automata, is “write a deterministic automaton that checks if its input contains an even number of letters a”, which can be automatically graded by running an algorithm for equivalence of automata. For excercises in a more complex domain, one has to compromise on the thoroughness of evaluation. A typical approach is to use testing; this is the way programming excercises are usually evaluated.

The challenge is thorough evaluation of nontrivial tasks. An ideal scenario for mathematics and computer science would be this: a student tries to prove a mathematical fact, and the grading system checks the proof and provides feedback in case of errors. It seems that we are far from achieving this scenario in its full generality, one of the reasons being that it requires a format of proofs that is readable by both humans and machines. Existing systems which manipulate proofs, such as the theorem prover COQ, despite tremendous work, are far too difficult for casual users like undergraduate students. Another issue is the intrinsic computational complexity of the underlyling algorithmic problems.

One could, however, attempt to attack the problem in more specific fields. It seems that the field of automata and formal languages is uniquely suited for such an attempt. The models that are studied, like finite automata or pushdown automata, can do interesting computation, and yet most algorithmic problems about them, like emptiness or membership, are decidable (unlike in most other computer science problems, where one quickly faces the halting problem). This balance between expressivity of the models and good algorithmic properties of questions about them, which is essentially the whole point of the discipline, is also what makes it a good candidate for automated evaluation of student excercises.

Here is an example. A classic introductory exercise on nondeterministic automata is “prove that if a language $L$ is regular, then its reverse, i.e. the set of reverses of words from $L$, is also regular”. To prove this, the student shows that if $A$ is a nondeterministic automaton, then there exists a nondeterministic automaton that recognises reverse of words accepted by $A$. The solution is very simple: keep the same states and alphabet, swap the initial and final states, and define the new set of transitions to be

$$\{(q,a,p) : (p,a,q) \in \delta\}$$
where $\delta$ is the transitions of the original automaton. This solution, and other equivalent ones, can be expressed in a way that is readable by both humans and computers. For more advanced exercises, the student might need to use more sophisticated constructions, which can be expressed in the language of set theory.

The goal of this internship is to participate in creating a tool for automatic grading of exercises in formal language theory. The design goal is that the text entered by the student is machine readable, but uses the same syntax that could be used on a blackboard in class, without any visible interference of the tool.