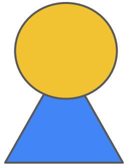
Automated Grading of Automata with ACL2s

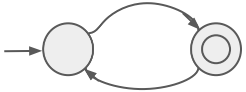
Ankit Kumar, Andrew Walter, Panagiotis Manolios

Northeastern University

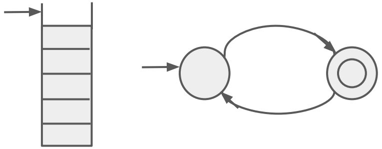
ThEdu ‘21

Theory of Computation

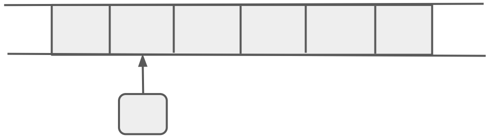




Finite Automata

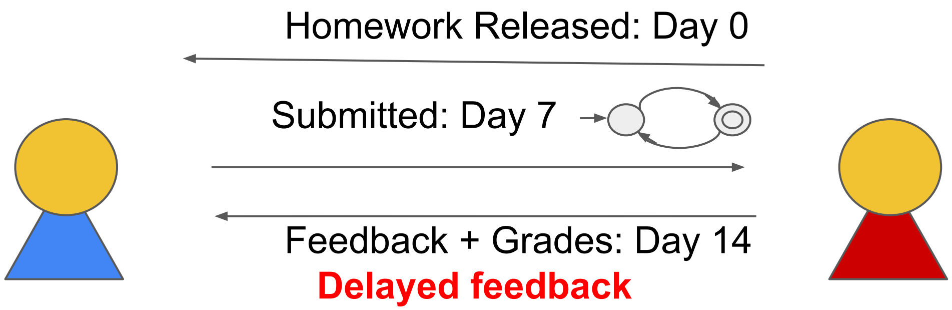


Pushdown Automata

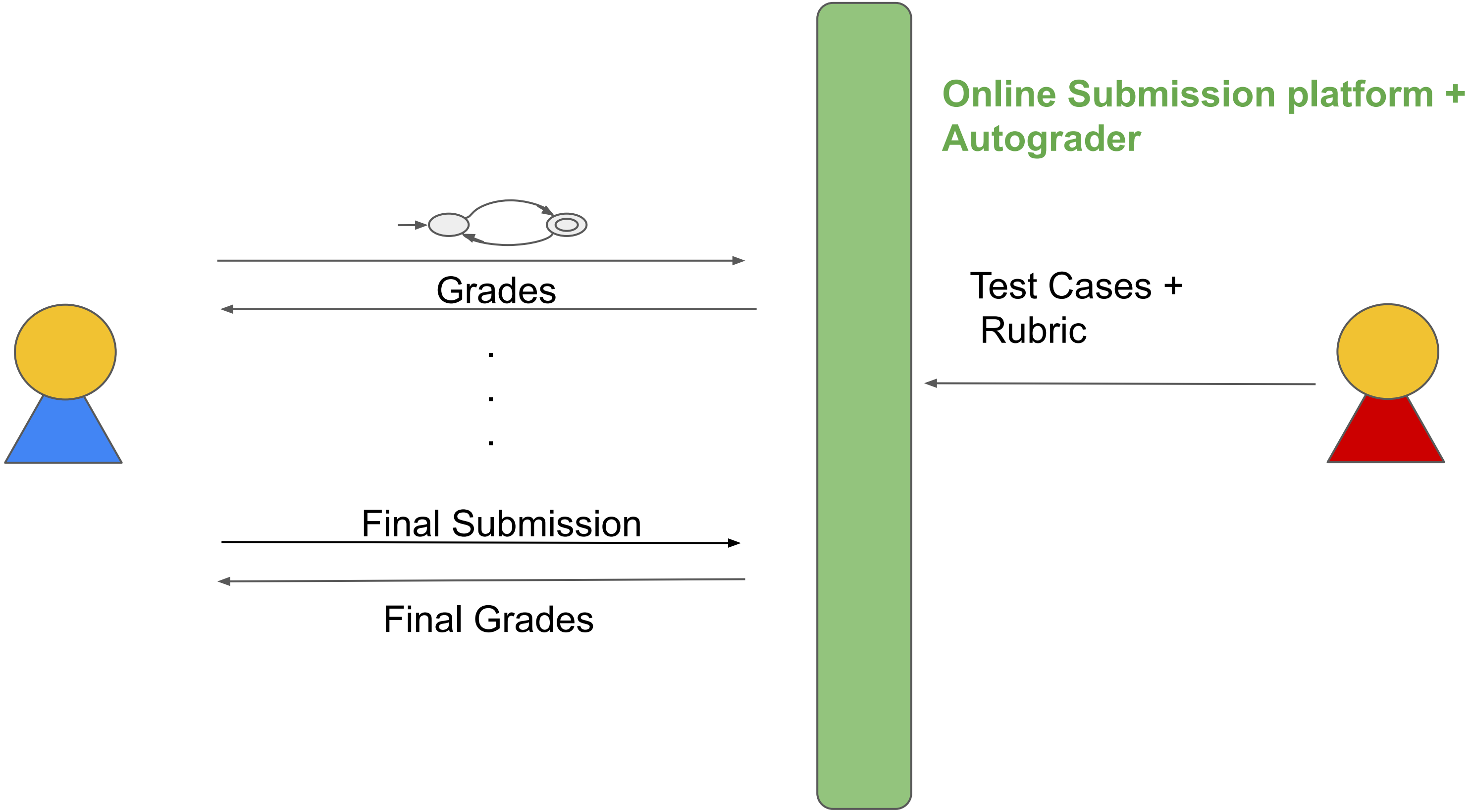


Turing Machines

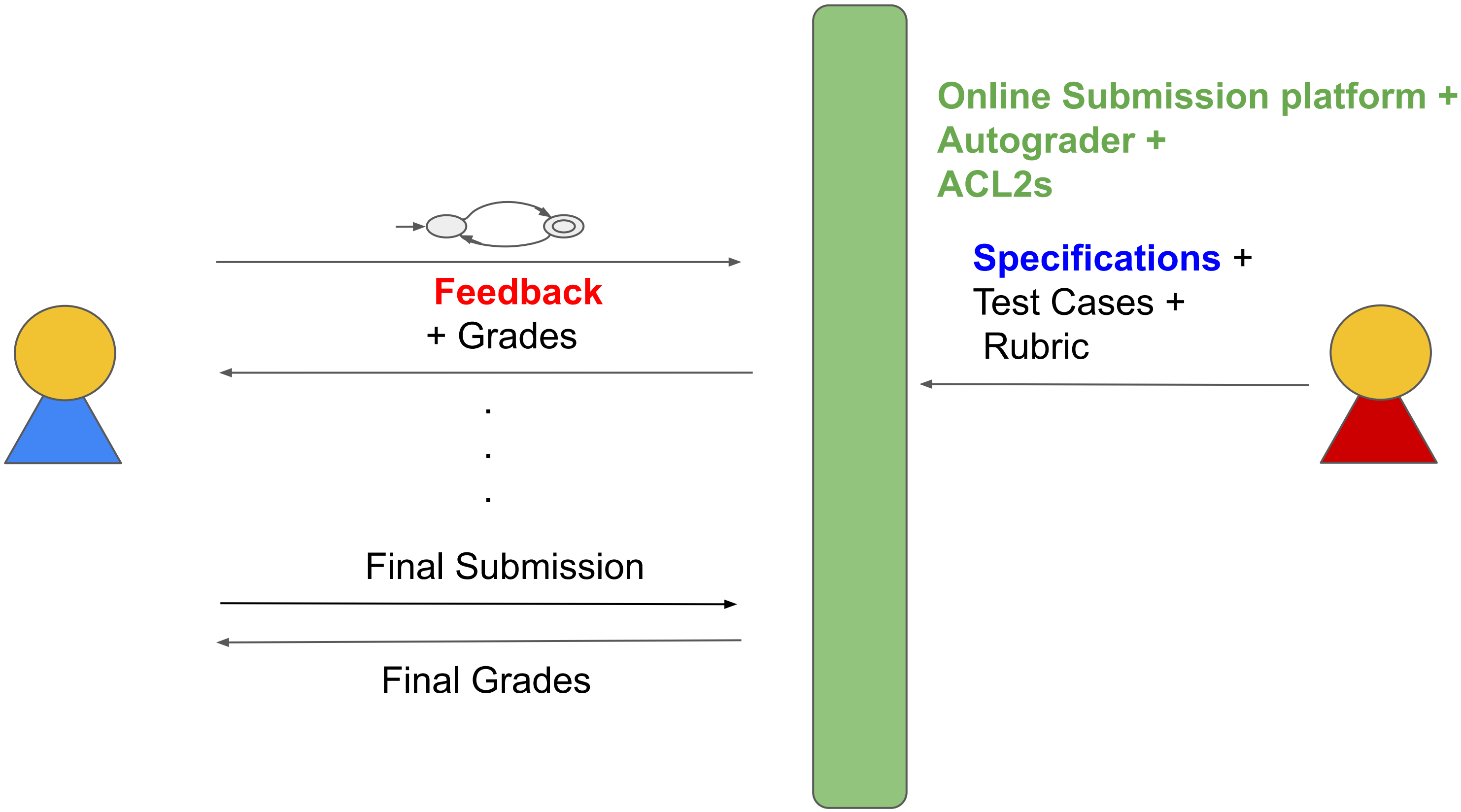
Manual Grading



Automatic Grading



Automatic Grading with ACL2s



ACL2s

- A powerful and user friendly system for integrated modeling, simulation, and interactive theorem proving in First Order Logic.
- Provides termination analysis and counterexample generation.
- Has been used in a Logic and Computation class at Northeastern to teach students logic and to reason about programs.

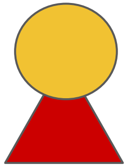
- ACL2s web-page. http://acl2s.ccs.neu.edu/acl2s/doc/
- Chamarthi, H., Dillinger, P.C., Manolios, P., Vroon, D.: The "ACL2" Sedan Theorem Proving System. In: TACAS (2011)
- Dillinger, P.C., Manolios, P., Vroon, D., Moore, J.S.: ACL2s: "The ACL2 Sedan". In: International Conference on Software Engineering (ICSE) (2007)

A sample problem

Construct a DFA that accepts words over {0,1}* consisting of an odd number of ones.

Specification (Instructor's solution)

```
(gen-dfa
 :name            instructor-dfa
 :states          (even odd)
 :alphabet        (0 1)
 :transition-fun  ((even 0 even)
                   (even 1 odd)
                   (odd  0 odd)
                   (odd  1 even))
 :start           even
 :accept          (odd))
```



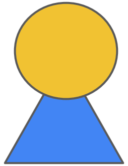
DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

- Sipser, M.: Introduction to the Theory of Computation. International Thomson Publishing, 1st edn. (1996)

Submission (Student’s solution)



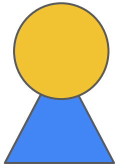
```
(gen-dfa
 :name              student-dfa
 :states            (e1 e2 o1 o2)
 :alphabet          (0)
 :start             e1
 :accept            (o1 o2)
 :transition-fun    ((e1  0  e1) (e1  2  o1)
                     (e2  0  e2) (e2  2  o2)
                     (o1  0  o2) (o1  2  e2)
                     (o2  0  o1) (o2  2  o1)))
```

test-legal-dfa (0.0/10.0)

```
[Domain of transition function is not of type : states x alphabet

((((STATE-ELEMENT-PAIR= '(O2 2))) ((STATE-ELEMENT-PAIR= '(O1 2)))
  ((STATE-ELEMENT-PAIR= '(E2 2))))))]
```

Submission (Student's solution)



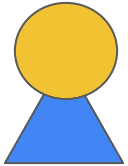
```
(gen-dfa
  :name               student-dfa
  :states             (e1 e2 o1 o2)
  :alphabet           (0)
  :start              e1
  :accept             (o1 o2)
  :transition-fun     ((e1  0  e1) (e1  2  o1)
                       (e2  0  e2) (e2  2  o2)
                       (o1  0  o2) (o1  2  e2)
                       (o2  0  o1) (o2  2  o1)))
```

Inside track : checking if DFA is legal

- All components provided
- Start state is one of states
- Accept states are a subset of given states
- Domain of transition function is of the right type
- Range of the transition function is of the right type

.
.
.

Submission (Student's solution)



```
(gen-dfa
  :name              student-dfa
  :states            (e1 e2 o1 o2)
  :alphabet          (0 2)
  :start             e1
  :accept            (o1 o2)
  :transition-fun    ((e1  0  e1) (e1  2  o1)
                      (e2  0  e2) (e2  2  o2)
                      (o1  0  o2) (o1  2  e2)
                      (o2  0  o1) (o2  2  o1)))
```

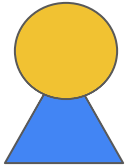
test-equivalence (0.0/10.0)

```
[Incorrect alphabet provided.]
```

test-legal-dfa (10.0/10.0)

```
Legal DFA : ((E1 E2 O1 O2) (0 2)
             (((E1 0) . E1) ((E1 2) . O1) ((E2 0) . E2) ((E2 2) . O2)
              ((O1 0) . O2) ((O1 2) . E2) ((O2 0) . O1) ((O2 2) . O1))
            E1 (O1 O2))
```

Submission (Student's solution)

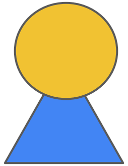


```
(gen-dfa
 :name            student-dfa
 :states          (e1 e2 o1 o2)
 :alphabet        (0 2)
 :start           e1
 :accept          (o1 o2)
 :transition-fun  ((e1  0  e1) (e1  2  o1)
                   (e2  0  e2) (e2  2  o2)
                   (o1  0  o2) (o1  2  e2)
                   (o2  0  o1) (o2  2  o1)))
```

Inside track : checking type equivalence

(defdata-equal instructor-dfa-alphabet student-dfa-alphabet)

Submission (Student’s solution)



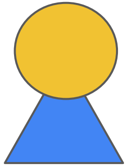
```
(gen-dfa
 :name              student-dfa
 :states            (e1 e2 o1 o2)
 :alphabet          (0 1)
 :start             e1
 :accept            (o1 o2)
 :transition-fun    ((e1  0  e1) (e1  1  o1)
                     (e2  0  e2) (e2  1  o2)
                     (o1  0  o2) (o1  1  e2)
                     (o2  0  o1) (o2  1  o1)))
```

test-equivalence (0.0/10.0)

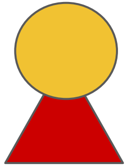
```
Transition function error. The following words
  were misclassified :
 ('(1 0 1 1) '(1 0 1 0) '(1 0 1))
```

test-legal-dfa (10.0/10.0)

```
Legal DFA : ((E1 E2 O1 O2) (0 1)
             (((E1 0) . E1) ((E1 1) . O1) ((E2 0) . E2) ((E2 1) . O2)
              ((O1 0) . O2) ((O1 1) . E2) ((O2 0) . O1) ((O2 1) . O1))
             E1 (O1 O2))
```



```
(gen-dfa
 :name            student-dfa
 :states          (e1 e2 o1 o2)
 :alphabet        (0 1)
 :start           e1
 :accept          (o1 o2)
 :transition-fun  ((e1  0  e1) (e1  1  o1)
                   (e2  0  e2) (e2  1  o2)
                   (o1  0  o2) (o1  1  e2)
                   (o2  0  o1) (o2  1  o1)))
```

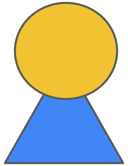


```
(gen-dfa
 :name            instructor-dfa
 :states          (even odd)
 :alphabet        (0 1)
 :start           even
 :accept          (odd)
 :transition-fun  ((even 0 even)
                   (even 1 odd)
                   (odd  0 odd)
                   (odd  1 even)))
```

Inside track : property based testing to check DFA equivalence

```
(test? (=> (instructor-dfa-wordp w)
           (== (accept-dfa instructor-dfa w)
               (accept-dfa student-dfa w))))
```

Submission (Student's solution)



```
(gen-dfa
  :name             student-dfa
  :states           (e1 e2 o1 o2)
  :alphabet         (0 1)
  :start            e1
  :accept           (o1 o2)
  :transition-fun   ((e1  0  e1) (e1  1  o1)
                     (e2  0  e2) (e2  1  o2)
                     (o1  0  o2) (o1  1  e2)
                     (o2  0  o1) (o2  1  e1)))
```

test-equivalence (10.0/10.0)

```
STUDENT-DFA is correct.
```

test-legal-dfa (10.0/10.0)

```
Legal DFA : ((E1 E2 O1 O2) (0 1)
             (((E1 0) . E1) ((E1 1) . O1) ((E2 0) . E2) ((E2 1) . O2)
              ((O1 0) . O2) ((O1 1) . E2) ((O2 0) . O1) ((O2 1) . E1))
             E1 (O1 O2))
```

Grading Turing Machines

A TM to flip 0s and 1s

```
(gen-tm
 :name student-tm
 :states (q0 q1 q2 q3)
 :alphabet (0 1)
 :tape-alphabet (0 1 nil)
 :start-state q0
 :accept-state q1
 :reject-state q2
 :transition-fun (((q0 1) . (q0 0 R))
                  ((q0 0) . (q0 1 R))
                  ((q0 nil) . (q3 nil R))
                  ((q3 nil) . (q1 nil L))))
```

Inside track : testing individual cases

```
(test? (== (tm-final-state student-tm (0 1 0 1 0 1))
           '(1 0 1 0 1 0)))
```

Test accept (1.0/1.0)

```
[TM1 ACCEPTED(Q0 0 1 0 1 0 1)
(1 Q0 1 0 1 0 1)
(1 0 Q0 0 1 0 1)
(1 0 1 Q0 1 0 1)
(1 0 1 0 Q0 0 1)
(1 0 1 0 1 Q0 1)
(1 0 1 0 1 0 Q0)
(1 0 1 0 1 0 NIL Q3)
(1 0 1 0 1 0 Q1 NIL NIL)
Passed test case]
```

Test accept (1.0/1.0)

```
[TM1 ACCEPTED(Q0 0)
(1 Q0)
(1 NIL Q3)
(1 Q1 NIL NIL)
Passed test case]
```

Test accept (1.0/1.0)

```
[TM1 ACCEPTED(Q0 0 0 0)
(1 Q0 0 0)
(1 1 Q0 0)
(1 1 1 Q0)
(1 1 1 NIL Q3)
(1 1 1 Q1 NIL NIL)
Passed test case]
```

Grading Turing Machines

A TM to flip 0s and 1s

```
(gen-tm
 :name student-tm
 :states (q0 q1 q2 q3)
 :alphabet (0 1)
 :tape-alphabet (0 1 nil)
 :start-state q0
 :accept-state q1
 :reject-state q2
 :transition-fun (((q0 1) . (q0 0 R)) ((q0 0) . (q0 1 R))
                  ((q0 nil) . (q3 nil R)) ((q3 nil) . (q1 nil L))))
```

```
Inside track : property based testing

(test? (=> (instructor-tm-wordp w)
           (== (tm-final-state instructor-tm w)
               (tm-final-state student-tm w))))
```

test-equivalence (10.0/10.0)

```
STUDENT-TM is correct.
```

test-legal-dfa (10.0/10.0)

```
Legal TM : ((Q0 Q1 Q2 Q3) (0 1) (0 1 NIL)
            (((Q0 1) Q0 0 R) ((Q0 0) Q0 1 R) ((Q0 NIL) Q3 NIL R)
             ((Q3 NIL) Q1 NIL L))
            Q0 Q1 Q2)
```

Using Automated grading in class

- Testing automata equivalence is not complete
- For the problems in class, this was not a limitation based on our testing
- Students interact via browser, exclusively
- ACL2s is invisible to students
- Instructors do not need experience with working in ACL2s; in fact, our class instructors who used our tools did not know any ACL2s
- Easy to use Instructor API : `grade, load-file, gen-dfa` and `check-dfa-equivalence`
- Publicly available
- Extensible, but requires familiarity with ACL2s

Class Observations

- Deployed automated grading using ACL2s in a ToC class of ~50 students.
- Before releasing each assignment, students were provided input format for submission.

In comparison to manual grading

- Significantly higher resubmissions as compared to manual grading.
- Higher grades in autograded assignments.
- On an average, more than 95% of the students got full credit on autograded problems, whereas less than 20% got full credit on manually graded problems.
- Positive feedback.

Tech Stack

AutograderToC — https://github.com/ankitku/AutoGradTOC/

gradescope-acl2s — https://github.com/ankitku/gradescope-acl2s

ACL2s interface — https://gitlab.com/acl2s/external-tool-support/interface

ACL2s (THE ACL2 SEDAN) — http://acl2s.ccs.neu.edu/acl2s/doc

ACL2 — https://github.com/acl2/acl2

docker — https://hub.docker.com/r/atwalter/acl2s_gradescope_autograder

gradescope — https://www.gradescope.com

Future Work

- Formal methods have been severely underutilized in education
- Our tool takes advantage of a full featured theorem prover with
  - counterexample generation capability
  - property based testing
- Can be extended to grade assignments in various other courses like Programming Languages, Software Engineering, Distributed Systems and Databases

Questions