Automatic Grading

“take a CEGAR and let the machine do your work"

(no Machine Learning in this talk)

Michaël Périn

May 17th 2018

Univ. Grenoble Alpes / VERIMAG, France

Autonomous Learning Process in brief

- **MOOC:** Massive Online Open Courses
  teachers'ubiquity: everywhere, whenever, subtitled in many languages

Autonomous Learning Process in brief

- **MOOC:** Massive Online Open Courses
  teachers'ubiquity: everywhere, whenever, subtitled in many languages
- **LMS:** Learning Management Systems
  - *help organizing the material: courses, exercises, exams,*
  - *manage user accounts and permissions*
  - *help managing the grading tasks of digitalized exams:* **Nb-Grader, Gradescope**

(*eg.* Moodle, Chamilo, Claroline, Dokeos, ...)

Autonomous Learning Process in brief

- **MOOC:** Massive Online Open Courses
  teachers'ubiquity: everywhere, whenever, subtitled in many languages
- **LMS:** Learning Management Systems
  - *help organizing the material: courses, exercises, exams,*
  - *manage user accounts and permissions*
  - *help managing the grading tasks of digitalized exams:* **Nb-Grader, Gradescope**

  (*eg.* Moodle, Chamilo, Claroline, Dokeos, ...)
- **DCMS:** Dynamic Content Management Systems
  you must pass the automatic test to unlock the next level

Autonomous Learning Process in brief

- **MOOC:** Massive Online Open Courses
  teachers'ubiquity: everywhere, whenever, subtitled in many languages
- **LMS:** Learning Management Systems
  - *help organizing the material: courses, exercises, exams,*
  - *manage user accounts and permissions*
  - *help managing the grading tasks of digitalized exams:* **Nb-Grader, Gradescope**

  (*eg.* Moodle, Chamilo, Claroline, Dokeos, ...)
- **DCMS:** Dynamic Content Management Systems
  you must pass the automatic test to unlock the next level
- **Training with Automatic Correction**
  - **mainly MCQ** *(Multiple-Choice Quiz)* **: most LMS**
  - **Web-notebook:** *online text editor + interpreter, for practice and examination* (*eg.* **Nb-Grader**, **Caseine**)
  - **Feeback:** ✓, ×

Autonomous Learning Process in brief

- **MOOC:** Massive Online Open Courses
  teachers'ubiquity: everywhere, whenever, subtitled in many languages
- **LMS:** Learning Management Systems
  - *help organizing the material: courses, exercises, exams,*
  - *manage user accounts and permissions*
  - *help managing the grading tasks of digitalized exams:* **Nb-Grader, Gradescope**

  (*eg.* Moodle, Chamilo, Claroline, Dokeos, ...)
- **DCMS:** Dynamic Content Management Systems
  you must pass the automatic test to unlock the next level
- **Training with Automatic Correction**
  - **mainly MCQ** *(Multiple-Choice Quiz)* **: most LMS**
  - **Web-notebook:** *online text editor + interpreter, for practice and examination* (*eg.* **Nb-Grader**, **Caseine**)
  - **Feeback:** ✓, ×
- Online exams are still marked by teachers

MOOC from a teacher's point of view

MOOC = Teachers' hell

- ⊖ time consuming... means reluctant to evolution
- ⊖ no student, no feedback from students,
- ⊖ only exams and markings

MOOC from a teacher's point of view

MOOC = Teachers' hell

- $\ominus$ time consuming... means reluctant to evolution
- $\ominus$ no student, no feedback from students,
- $\ominus$ only exams and markings

My dream as a teacher

- $\oplus$ interactive course with just chalks and black board,
- $\oplus$ interaction with students,
- $\bigcirc$ writing exams with solutions,
- $\oplus$ no markings = automatic grading

Evaluation in Learning Management Systems

- **Simple answers: boolean, string or numerical values**
  $\rightarrow$ Exams must be smart to evaluate elaborated reasoning through **MCQ**

Evaluation in Learning Management Systems

- **Simple answers: boolean, string or numerical values**
  → Exams must be smart to evaluate elaborated reasoning through **MCQ**
- **Test campaign** on **executable code** produced by students
  - script running on student projects
  - web-notebook for programming exercises
    Jupyter supports 40 programming languages,
    Caseine: local development at G-SCOP on top of **Moodle** and **Virtual Lab**
  - feedback on each test: ✓, ×

→ No evaluation of the code quality.

Evaluation in Learning Management Systems

- **Simple answers: boolean, string or numerical values**
  $\rightarrow$ Exams must be smart to evaluate elaborated reasoning through **MCQ**
- **Test campaign** on **executable code** produced by students
  - script running on student projects
  - web-notebook for programming exercises
    Jupyter supports 40 programming languages,
    Caseine: local development at G-SCOP on top of **Moodle** and **Virtual Lab**
  - feedback on each test: ✓, ×

  $\rightarrow$ No evaluation of the code quality.
- **Serious Game** *eg.* treasure hunt $\rightarrow$ What is evaluated ?
  "learning linux command and tools" by M.Moy **evaluates the capacity of self-training and finding information**
  $\rightarrow$ Students can be stuck.

This talk I

A brainstorming session

- I'm not an expert in Learning Management Systems, just a teacher fed up with grading
- a *1 brain* $\times$ *month* work
- hijacking verification techniques

Assumptions

- no digitalization or handwriting recognition
- students composing on machine becomes feasible 90% of CS students have a laptop

This talk II

… addresses **two problems of LMS**

- Exams must be smart to evaluate a precise knowledge or elaborated reasoning through **Multiple-Choice Quiz**
- Not enough feedback to students

Contributions

- **Automatic Correction** and **Automatic Grading**
- more freedom in exercises than Multiple-Choice Questions
- finer evaluation of knowledge
- more feedback to students for training before the exams

Grading exams (facts from teachers' interviews)

- **Grading takes times**
  - $10\,\mathrm{min} \sim 15\,\mathrm{min}$ by form
  - $240 \sim 300$ exam forms/year
  - $41 \sim 75$ hours/year
  - $2.5 \sim 5$ weeks/year as half-time job
  - it is not precisely taken into account as teaching time
- **Students have no return,**
  only the final grade ... weeks later
- **Grading is not uniform** between the first and the last exam forms

Automatic grading

Automatic Grading is for teachers:

1. it saves hours for research
   it automatically computes the grade
2. even if it is not perfect, it can do better than teachers
   it is uniform

Technical problem: most of student answers are incorrect but "partially correct". The difficulty of Automatic Grading is

the detection of "good ideas" in an incorrect answer.

Student training (facts from students' interviews)

The Y,Z generation asks for **Personalized Feedback**

1. Providing the solution of an exercice is not satisfactory:
   - it's all done!
   - is the teacher's solution the only way?
   - it does not explain why my reasoning was incorrect.
2. Supplementary exercices are not done

... and for **High Frequency Feedback**

1. feedback on tests comes too late, at best one week later
2. students moved to another topic ;
3. they only look at the grade

Automatic correction

Automatic Correction / Repair is for students:

1. training with *instantaneous feedback*
2. feedback on *their production*
3. self-evaluation of strengths and weaknesses *without external judgement*
4. they get used to the expected answers: *payback guaranty*

Technical problem: repairing student's productions toward the solution, and annotating with: correct, uncomplete, incorrect, correction

Technical part of the talk

1. a definition of **automatic grading** from automatic correction
2. a CEGAR approach for **automatic correction** (Counter-Example-Guided Answer Repair)

Automatic grading as editing distance

Definition (Automatic Grading)

Computing the distance between the student answer A and the solution S

Requirements

- a formal notation: $A, S \in \mathcal{T}erm$
- a decideable equivalence relation : $A \overset{?}{\simeq} S$

Example (finite automata)

- $\mathcal{T}erm = Automata(\Sigma = \{a, b\}, State = \{1, 2, 3\})$
- $A \simeq S$ iff $\mathscr{L}(A) = \mathscr{L}(S)$

Automatic grading as editing distance

Editing distance = minimal number of rewritings

- Given some transformers $\tau_1, \tau_2, \ldots : \mathcal{T}erm \to \mathcal{T}erm$
- $|A_0 \xrightarrow{\tau_1} A_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} A_n| = n$
- *edist*$(A, S) = \min\{\ |A \xrightarrow{\tau}{}^{*} S|\ \}$

Example (Transformers of finite automata)

- adding/removing a state
- changing status of a state: initial / accepting / non-accepting
- adding/removing a transition

A concrete example

Exercise

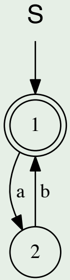
Design a finite automaton that recognizes $\{(a.b)^n \mid n \in \mathbb{N}\}$

Example (Solution provided by the teacher)

$$L = \{\epsilon,\ ab,\ abab,\ ab \cdot \ldots \cdot ab,\ \ldots\}$$

```
S:= automaton{
    ->((1))-a->(2)-b->((1))
}
```

$\xrightarrow{parse+output}$



A concrete example

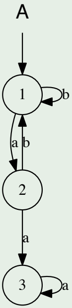
Exercise

Design a finite automaton that recognizes $\{(a.b)^n \mid n \in \mathbb{N}\}$

Example (student answer A)

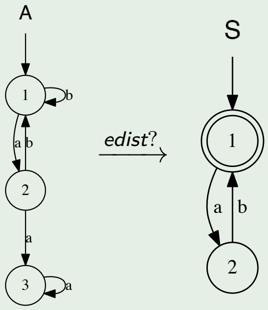
```
A:= automaton{
    ->(1)-a->(2)-b->(1) ;
    (1)-b->(1) ;
    (2)-a->(3) ;
    (3)-a->(3)
}
```

$\xrightarrow{parse+output}$



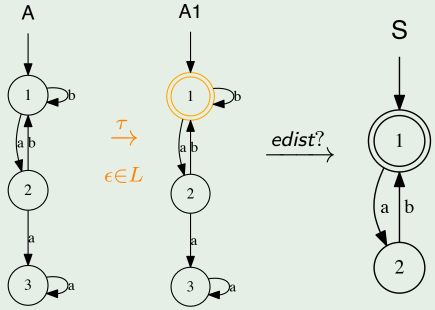
The editing distance to the provided solution S

Example (let's compute *edist*(A, S))



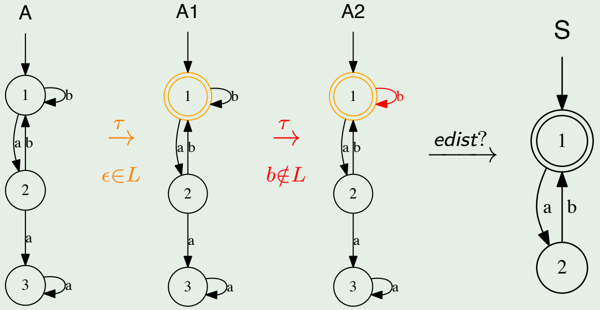
The editing distance to the provided solution S

Example (let's compute *edist*(A, S))



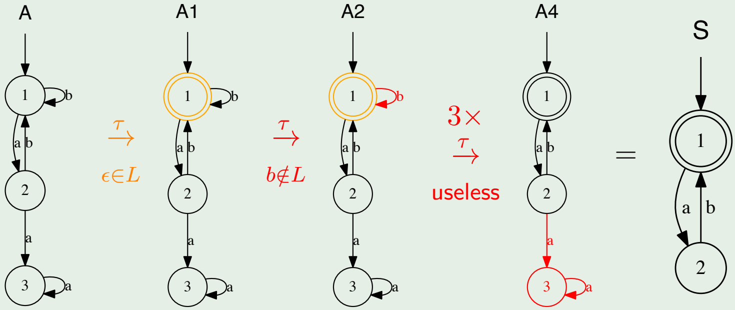
The editing distance to the provided solution S

Example (let's compute *edist*(A, S))



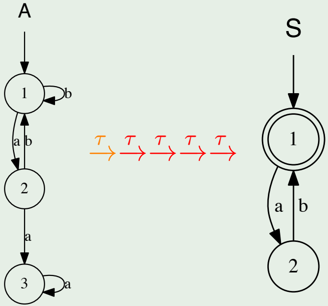
The editing distance to the provided solution S

Example (let's compute *edist*(A, S))



The editing distance to the provided solution S

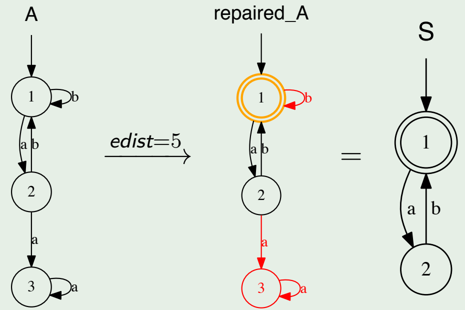
Example (let's compute *edist*(A, S))



$$\textit{edist}(A, S) = 5$$

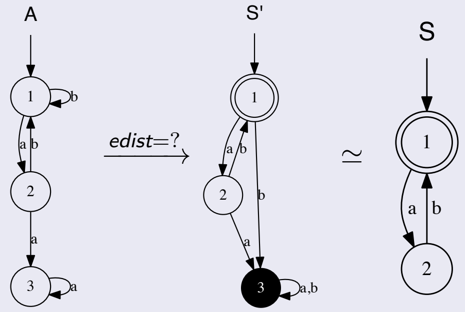
Feedback, a side-effect of computing *edist*

Example



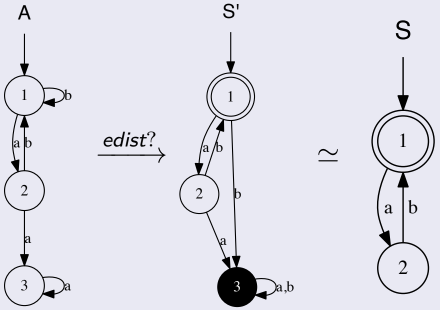
$edist(A, S) = 5$, is it fair?

Another repairing is possible!



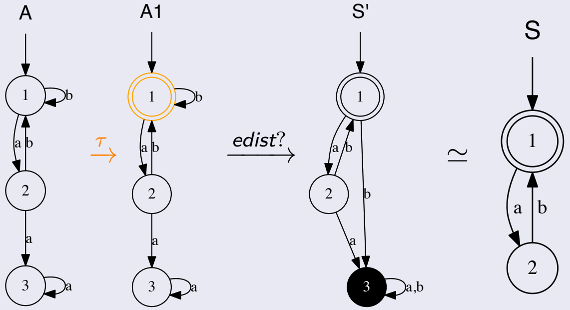
$\mathit{edist}(A, S) = 5$, is it fair?

Consider $S' = \mathit{FullSpec}(S)$ and compute $\mathit{edist}(A, S')$



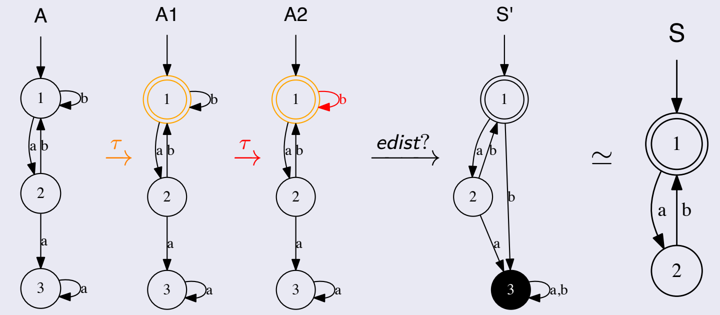
$\mathit{edist}(A, S) = 5$, is it fair?

Consider $S' = \mathit{FullSpec}(S)$ and compute $\mathit{edist}(A, S')$



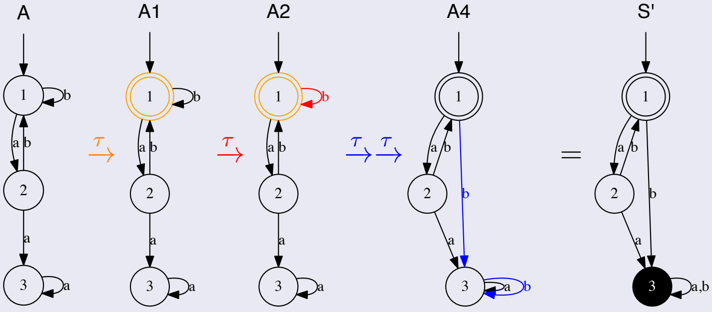
edist$(A, S) = 5$, is it fair?

Consider $S' = \mathit{FullSpec}(S)$ and compute *edist*(A, S')



$\textit{edist}(A, S) = 5$, is it fair?

Consider $S' = \textit{FullSpec}(S)$ and compute $\textit{edist}(A, S')$



$$\textit{edist}(A, S') = 4 < \textit{edist}(A, S) = 5 \quad \text{and} \quad S' \simeq S$$

Grading, a mix of syntax and semantics

The grade $\in [0, 1]$

… evaluates the *syntactic differences* between the answer A and a solution S' *semantically equivalent* to the provided one.

$$\textit{grade}(A) = 1 - \frac{\min\{\textit{edist}(A,S') \mid S' \simeq S\}}{\textit{size}(S)}$$

where $\textit{size}(S) = \min\{\textit{edist}(\emptyset, S') \mid S' \simeq S\}$ $=$ *the minimal number of steps to build a solution equivalent to* S

and $\textit{size}(S')$ is bounded

Roughly, $\textit{size}(S') < \underbrace{\textit{size}(A)}_{erase} + \underbrace{\textit{size}(S)}_{rebuild}$

Grading, a mix of syntax and semantics

The grade $\in [0,1]$

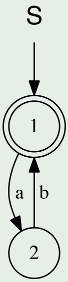
$$\begin{array}{lrcl} & \mathit{grade}(A) & = & 1 - \frac{\min\{\mathit{edist}(A,S') \mid S' \simeq S\}}{\mathit{size}(S)} \\ \text{where} & \mathit{size}(S) & = & \min\{\mathit{edist}(\emptyset, S') \mid S' \simeq S\} \\ \text{Roughly,} & \mathit{size}(S') & < & \underbrace{\mathit{size}(A)}_{erase} + \underbrace{\mathit{size}(S)}_{rebuild} \end{array}$$

Example (requirements on the grading function)

- $\forall S' \simeq S, \quad \mathit{grade}(S') = 1$ since $\mathit{edist}(S', S') = 0$
- $\mathit{grade}(\emptyset) = 0$ since $\mathit{grade}(\emptyset) = 1 - \frac{\mathit{size}(S)}{\mathit{size}(S)}$

Example: Grading + Feedback

Example



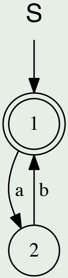
$$S \quad : \quad \text{2 states, 2 transitions, 1 initial state, 1 accepting state}$$

$$\textit{size}(S) = \min\{\textit{edist}(\emptyset, S') \mid S' \simeq S\} = 6$$

$$\textit{grade}(A) = 1 - \frac{\min\{\textit{edist}(A, S') \mid S' \simeq S\}}{6}$$

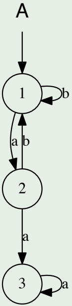
Example: Grading + Feedback

Example

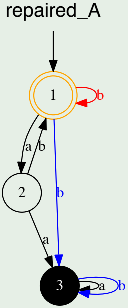


$$\mathit{edist}(A_1, S) = 4$$

$$\mathit{grade}(A_1) = 1 - \frac{4}{6}$$



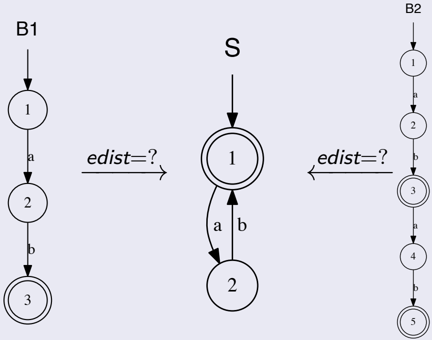
$\boxed{\frac{0.33}{1}}$



Interesting fairness of this grading function

B_1 and B_2 should receive the same grade

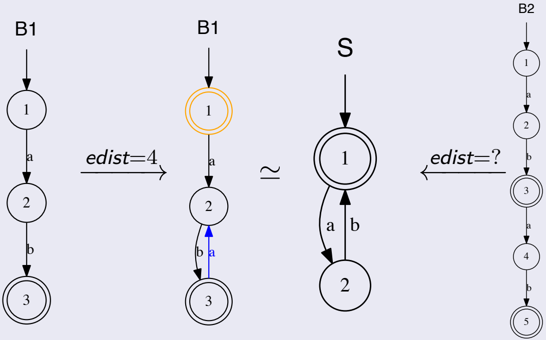
... they both forgot the loop and the word ϵ.



Interesting fairness of this grading function

B_1 and B_2 should receive the same grade

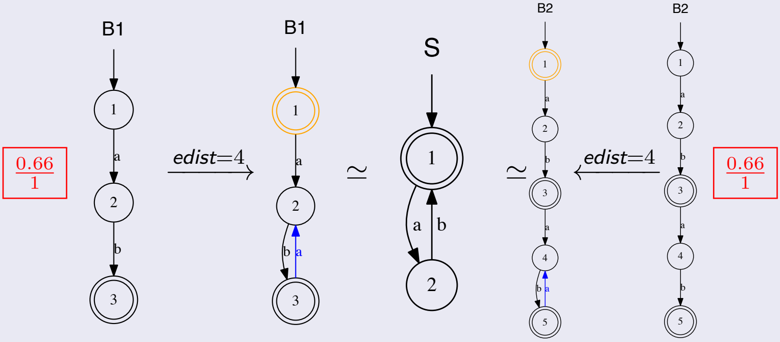
... they both forgot the loop and the word ϵ.



Interesting fairness of this grading function

B_1 and B_2 should receive the same grade

… they both forgot the loop and the word ϵ.



The repairing algorithm

The general idea, independant of the domain : CEGAR

- Iteratively repair $A_0 \xrightarrow{\tau_1} \dots \xrightarrow{\tau_i} A_i$ until $A_i \simeq S$ then return $\tau_1, \dots, \tau_i$
- SMT (Solver Modulo Theory) can be useful there to check the equivalence $A_i \overset{?}{\simeq} S$.
  - TRUE : return $\tau_1, \dots, \tau_i$
  - FALSE($\mathcal{C}$) : use the counter example $\mathcal{C}$
    1. as feedback
    2. for selecting a repair τ_{i+1} of A_i

  $A_i \xrightarrow{\tau_{i+1}} A_{i+1}$; recall with $A_{i+1} \overset{?}{\simeq} S$.

This *Counter-Example-Guided Answer Repair* approach is similar to the CEGAR method (Counter-Example-Guided Abstraction Refinment) used in computer aided verification.

Applications to Deterministic Finite Automata

(an ideal case study)

a Recap on DFA

Checking equivalence of automata $A \overset{?}{\simeq} S$ I

- The langage associated to the state q of an Automaton A

$$L_q \overset{\text{def}}{=} \mathscr{L}(A \text{ with } q_{init}(A) := q)$$

- States are equivalent, denoted $q \sim q'$, iff $L_q = L_{q'}$
- Equivalence of Automata

$$\begin{array}{rccc} & A & \simeq & S \\ \text{iff} & \mathscr{L}(A) & = & \mathscr{L}(S) \\ & = & & = \\ & L_{q_{init}(A)} & = & L_{q_{init}(S)} \\ \text{iff} & q_{init}(A) & \sim & q_{init}(S) \end{array}$$

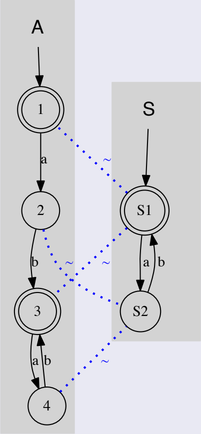
- **Algorithm:** compute the equivalent states of the automaton $A + S$ and check $q_{init}(A) \overset{?}{\sim} q_{init}(S)$

Checking equivalence of automata $A \overset{?}{\simeq} S$ II

Counter-Example-Guided Answer Repair

Compute $\sim$ in $A+S$: $1 \sim S_1 \sim 3, \quad 2 \sim S_2 \sim 4$

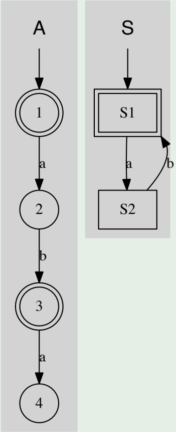
$\frac{1}{1}$



$$
\begin{array}{rrcl}
 & 1 & \sim & S_1 \\
ie. & q_{init}(A) & \sim & q_{init}(S) \\
ie. & L_{q_{init}(A)} & = & L_{q_{init}(S)} \\
ie. & \mathscr{L}(A) & = & \mathscr{L}(S) \\
thus, & A & \simeq & S
\end{array}
$$

Computing $\sim$ using partition refinment

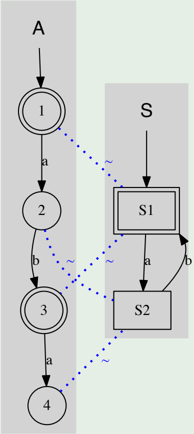
Example (Constructing equivalence classes of $\sim$ in $A+S$)



Initially, all states are $\sim$
$\{1,2,3,4,S_1,S_2\}$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



Initially, all states are $\sim$

$$\{1, 2, 3, 4, S_1, S_2\}$$

1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

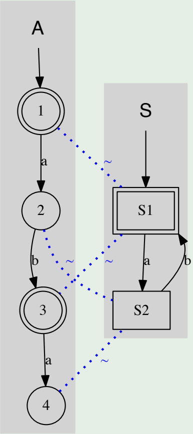
interpreted as

$1 \sim S_1,\ 3 \sim S_1$

$2 \sim S_2,\ 4 \sim S_2$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

interpreted as

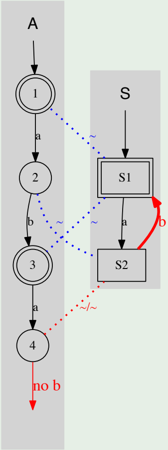
$$1 \sim S_1,\ 3 \sim S_1$$
$$2 \sim S_2,\ 4 \sim S_2$$

$\sim$ **is not valid:**

$$A \not\simeq S$$
$$abab \notin \mathscr{L}(A),\ abab \in \mathscr{L}(S)$$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

$\sim$ is not valid:

$$A \not\simeq S$$

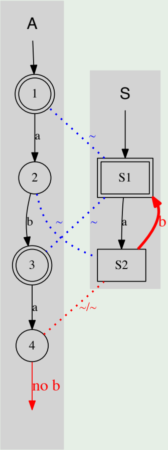
$$abab \notin \mathscr{L}(A),\ abab \in \mathscr{L}(S)$$

because

$4 \stackrel{b}{\not\to}$ **whereas** $S_2 \xrightarrow{b} S_1$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\underbrace{\{1, 3, S_1\}}_{}^{\textit{accepting}} \sqcup \overbrace{\{2, 4, S_2\}}^{\textit{non-acc.}}$$

$4 \not\xrightarrow{b}$ **whereas** $S_2 \xrightarrow{b} S_1$

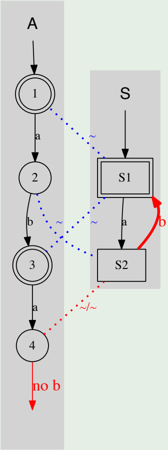
thus, $4 \not\sim S_2$

2nd partition refinment

$$\{1, 3, S_1\} \sqcup \{4\} \sqcup \{2, S_2\}$$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

$4 \overset{b}{\not\to}$ **whereas** $S_2 \xrightarrow{b} S_1$

thus, $4 \not\sim S_2$

2nd partition refinment

$$\{1, 3, S_1\} \sqcup \{4\} \sqcup \{2, S_2\}$$

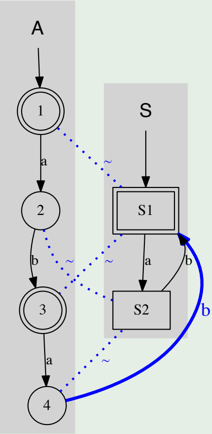
«$4 \sim \emptyset$» **triggers a repair:**

add $4 \xrightarrow{b} S_1$ to get $4 \sim S_2$

or remove 4

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

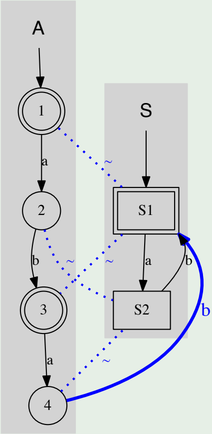
$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

«$4 \sim \emptyset$» **triggers a repair:**

add $4 \xrightarrow{b} S_1$ to get $4 \sim S_2$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

«$4 \sim \emptyset$» **triggers a repair:**

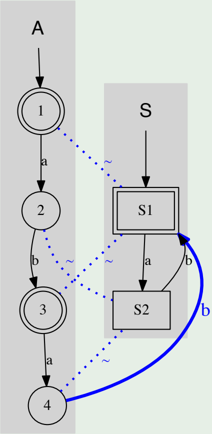
add $4 \xrightarrow{b} S_1$ to get $4 \sim S_2$

back to the 1st partition

$$\{1, 3, S_1\} \sqcup \{2, 4, S_2\}$$

Computing $\sim$ using partition refinment

Example (Constructing equivalence classes of $\sim$ in $A + S$)



1st partition refinment

$$\overbrace{\{1, 3, S_1\}}^{accepting} \sqcup \overbrace{\{2, 4, S_2\}}^{non-acc.}$$

back to the 1st partition

$$\{1, 3, S_1\} \sqcup \{2, 4, S_2\}$$

interpreted as

$$1 \sim S_1, 3 \sim S_1,$$
$$2 \sim S_2, 4 \sim S_2$$

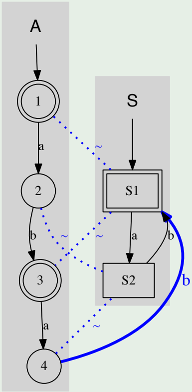
is valid in

the repaired automata

$$A + S + 4 \xrightarrow{b} S_1$$

Repairing transitions of A

Example (continued)



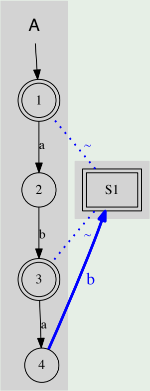
1. consider the graph of the repaired automaton $(A + S + 4 \xrightarrow{b} S_1)$ with $\sim$ edges

2. remove the $State(S)$ except the targets of $State(A)$

   eg. $4 \xrightarrow{b} \boxed{S_1}$

Repairing transitions of A

Example (continued)

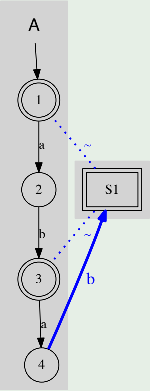


1. consider the graph of the repaired automaton $(A + S + 4 \xrightarrow{b} S_1)$ with $\sim$ edges
2. remove the $State(S)$ except the targets of $State(A)$

   eg. $4 \xrightarrow{b} \boxed{S_1}$

Repairing transitions of A

Example (continued)

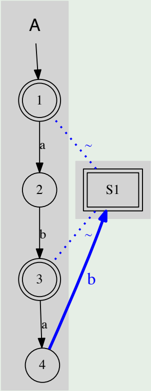


1. consider the graph of the repaired automaton $(A + S + 4 \xrightarrow{b} S_1)$ with $\sim$ edges
2. remove the $State(S)$ except the targets of $State(A)$

   eg. $4 \xrightarrow{b} \boxed{S_1}$
3. replace $q \xrightarrow{\ell} \boxed{S_i} \overset{\sim}{\cdots} q'$ by $q \xrightarrow{\ell} q'$

Repairing transitions of A

Example (continued)



1. consider the graph of the repaired automaton $(A + S + 4 \xrightarrow{b} S_1)$ with $\sim$ edges
2. remove the $State(S)$ except the targets of $State(A)$

   eg. $4 \xrightarrow{b} \boxed{S_1}$

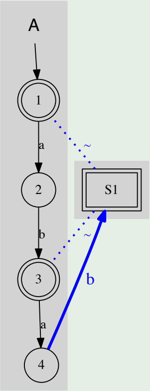
3. replace $q \xrightarrow{\ell} \boxed{S_i} \overset{\sim}{\cdots} q'$ by $q \xrightarrow{\ell} q'$

   eg. $4 \xrightarrow{b} \boxed{S_1} \overset{\sim}{\cdots} 1$

   becomes $4 \xrightarrow{b} 1$

Repairing transitions of A

Example (continued)



1. consider the graph of the repaired automaton $(A + S + 4 \xrightarrow{b} S_1)$ with $\sim$ edges

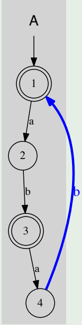
2. remove the $State(S)$ except the targets of $State(A)$

   eg. $4 \xrightarrow{b} \boxed{S_1}$

3. replace $q \xrightarrow{\ell} \boxed{S_i} \overset{\sim}{\ldots} q'$ by $q \xrightarrow{\ell} q'$

   eg. $4 \xrightarrow{b} \boxed{S_1} \overset{\sim}{\ldots} 1$

   becomes $4 \xrightarrow{b} 1$



$\frac{0.83}{1}$

A repair strategy guided by $\sim$ counter-examples

1. **Computes the $\sim$ relation using partition refinment:** it always succeeds and provides the greatest equivalence relation.

A repair strategy guided by $\sim$ counter-examples

1. **Computes the $\sim$ relation using partition refinment:** it always succeeds and provides the greatest equivalence relation.
2. **Reparing $A + S$ may be required to guaranty**
   - **(every state of A) $\sim$ (a state of S)**
   - **(every state of S) $\sim$ (a state of A)**

A repair strategy guided by $\sim$ counter-examples

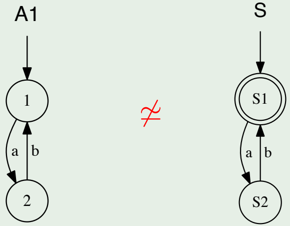
1. **Computes the $\sim$ relation using partition refinment:** it always succeeds and provides the greatest equivalence relation.
2. **Reparing $A + S$ may be required to guaranty**
   - **(every state of A) $\sim$ (a state of S)**
   - **(every state of S) $\sim$ (a state of A)**
3. **Available Repairs on Automata**
   - status (initial / accepting / non-accepting) of states
   - addition / removal of transitions and states

A repair strategy guided by $\sim$ counter-examples

1. **Computes the $\sim$ relation using partition refinment:** it always succeeds and provides the greatest equivalence relation.
2. **Reparing $A + S$ may be required to guaranty**
   - **(every state of A) $\sim$ (a state of S)**
   - **(every state of S) $\sim$ (a state of A)**
3. **Available Repairs on Automata**
   - status (initial / accepting / non-accepting) of states
   - addition / removal of transitions and states
4. *repaired*(A) **is reconstructed from** $A + S + \textit{repairs}$

Repairing accepting states

Example ($A_1 \overset{?}{\simeq} S$ by computing $\sim$ in $A_1 + S$)

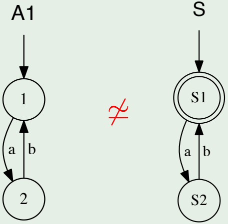


$1 \sim S_2, \quad 2 \sim S_2, \quad S_1 \sim \emptyset$

$S_1 \sim \emptyset$ **triggers a repair.**

Repairing accepting states

Example ($A_1 \overset{?}{\simeq} S$ by computing $\sim$ in $A_1 + S$)



Diagnostic:

S_1 is accepting, $S_1 \sim \emptyset$

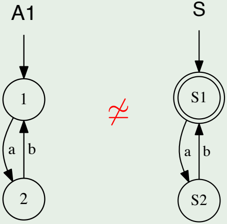
thus, missing accepting states

Counter-example:

$\epsilon \notin \mathscr{L}(A_1)$ $\epsilon \in \mathscr{L}(S)$

Repairing accepting states

Example ($A_1 \overset{?}{\simeq} S$ by computing $\sim$ in $A_1 + S$)



Diagnostic:

S_1 is accepting, $S_1 \sim \emptyset$

thus, missing accepting states

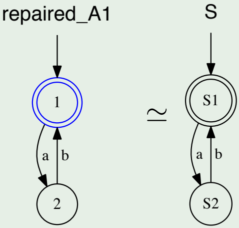
Counter-example:

$\epsilon \notin \mathscr{L}(A_1)$ $\qquad$ $\epsilon \in \mathscr{L}(S)$

then repairing

$\mathit{run}(A_1, \epsilon) = 1 \xrightarrow{\epsilon} 1$

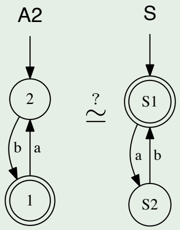
thus, 1 must be accepting



$\mathit{grade}(A_1) = \boxed{\frac{0.83}{1}}$

Repairing initial states

Example ($A_2 \overset{?}{\simeq} S$ by computing $\sim$ in $A_2 + S$)

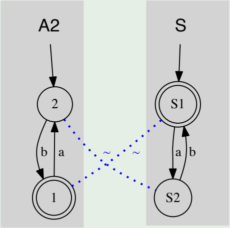


Repairing initial states

Example ($\sim$ in $A_2 + S$: $1 \sim S_1$, $2 \sim S_2$)

checking equivalence

$A_2 \not\approx S$



because

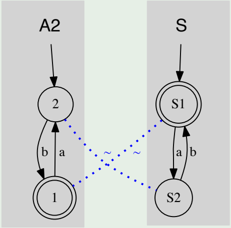
$q_{init}(A_2) \not\sim q_{init}(S)$

Repairing initial states

Example ($\sim$ in $A_2 + S$: $1 \sim S_1$, $2 \sim S_2$)

checking equivalence

$A_2 \not\simeq S$

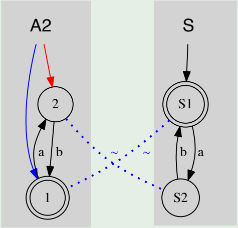


because

$q_{init}(A_2) \not\sim q_{init}(S)$

then repairing

$q_{init}(A_2) \sim q_{init}(S)$



entails $A_2 \simeq S$

Repairing initial states

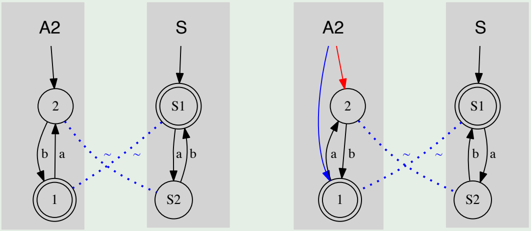
Example ($\sim$ in $A_2 + S$: $1 \sim S_1$, $2 \sim S_2$)

checking equivalence

$A_2 \not\equiv S$

then repairing

$q_{init}(A_2) \sim q_{init}(S)$



because

$q_{init}(A_2) \not\sim q_{init}(S)$

entails $A_2 \simeq S$

grade$(A_2) = \boxed{\frac{0.66}{1}}$

The repair algorithm for DFA

repair(A,S) : set of repaired automata

1. computes $\sim$ in $A + S$
2. return $\{A\}$ if
   - $A \simeq S$ *ie.* $q_{init}(A) \sim q_{init}(S)$
   - *every state of* A $\sim$ *state of* S
   - *every state of* S $\sim$ *state of* A

   otherwise:
3. use the counter-example to select possible repairs $\{\tau_1, \ldots, \tau_r\}$ *eg.* { add transition, remove state }
4. recursively call *repair* on each automaton $\tau_i(A)$

$$\textit{repair}(\tau_1(A), S) \ \cup \ldots \cup \ \textit{repair}(\tau_r(A), S)$$

A generic grading algorithm (using *repair*)

Grading is obvious from repairs

1. $\{A'_1, \ldots, A'_n\} := \mathit{repair}(A, S)$ where each $A'_i \simeq S$
2. select the repaired automaton with **minimal repair cost**

$$A' := \mathit{imin}\ [\ \mathit{edist}(A, A'_1)\ ; \ldots ;\ \mathit{edist}(A, A'_r)\ ]$$

where $\mathit{edist}(A, A') \stackrel{\text{def}}{=} n \qquad$ for $A' = \tau_1 \circ \ldots \circ \tau_n(A)$

3. return the **repaired automaton** and **the grade**

$$\mathit{grading}(A, S) \stackrel{\text{def}}{=} \left(\underbrace{\overbrace{\tau_1 \circ \ldots \circ \tau_n(A)}^{A'}}_{\text{feedback}}\ ,\ \underbrace{1 - \frac{\mathit{edist}(A, A')}{\mathit{size}(S)}}_{\text{grade}} \right)$$

Experiments in progress I

Experiments at Polytech

- 6 students, running a limited linux distribution (made by P.Corbineau): *guest login, no network*.
- an exam on Automata & Grammars with *additional instructions to provide answers in a given syntax*
- questions available as paper sheet, in pdf format (subject.pdf), and in ascii format in the answer file (subject.org)
- both files are produced automatically from the latex source (subject.tex)
- corrected in the standard way

Experiments in progress II

Lessons learned

- ⊖ students want a paper version of the test
- ▪ composing on machine is well-accepted
- ⊕ the imposed syntax helps students structuring their answers
- ⊕ digital exams on your laptop weigh 0kg
- ⊕ answers are organized and can be fold/unfold
- ▪ answer parsers must be available for students
- ▪ parsers must be user-friendly: tolerant syntax and feedback (answer.html)

Next steps

Alexandre Borthomieu, L3, magistère info

- development of compliant parsers and automatic grading
- extension non-deterministic automata:
  $A^D :=$ *det*(A) ; *repair*(A^D, S) ;
  back propagation of $\tau_1 \circ \ldots \tau_n$ to A?

Next steps

Alexandre Borthomieu, L3, magistère info

- development of compliant parsers and automatic grading
- extension non-deterministic automata:
  $A^D :=$ *det*(A) ; *repair*(A^D, S) ;
  back propagation of $\tau_1 \circ \ldots \tau_n$ to A?

Philippe Genin, ITA, Verimag

- booting USB key with a limited linux distribution
- *guest login only, mouse, keyboard, gedit, html viewer*
- no network or restricted domain,
- no hard disk, saving subject.org on the USB key,
- executable parsers that generate html feedback

Future work I

Larger experiments (2019 ?)

- **Convincing the school administration**
- **60 students:** 1st year Polytech CS engineers
- **90%** of CS students have a **laptop**
- **Automata & Grammars**' exams on machine
- **Deployment & costs:** new room organisation, separator, one USB key per student + screen privacy filter ?
- **Development:** the grading software must be available for training (parsers, repair, grading)
- **Integration in the Caseine web platform**

Future work II

Application to other teaching units?

- Automata, Regular Expression, Langage Equations
- Grammars: Context-Free Grammars
- Propositional Logic, Resolution
- Program Proof in restricted Hoare Logic
- Linear Programming (with Nicolas Catusse)

The CEGAR approach could probably be used,
and brut force is also possible:

- small problems (automata with ≤ 10 states)
- constraints on answers (4 states, initial state $= 1$)
- templates of logic invariant in program proofs

Future work III

Autograding

- Is it a research topic? a business? a new trend?
- just a convenient tool?

Plans for the next few years

1. Nice subject for internship
2. Application to my teaching units
3. Generalization using SMT ?
4. Publication ? In which community ?

Challenging problem of comparing two programs

Teaching Programming is difficult

1. Tests do not evaluate the code quality (O.Grüber's experience)
2. High Frequency Feedback to improve student skills

no off-the-shelf solution for comparing two programs
but
it is an active domain in program verification since 1998.

Equivalence of two programs

- Translation Validation for Optimizing Compilers [Pnueli+ TACAS'98, Necula PLDI'2000, Zuck+ JUCS'2003, Tristan+ PLDI'2011].
- The desired $\simeq$ relation resembles that of Proving inter-program properties [Voronkov+ SAS'2009] where nodes of the control Flow Graph of the orignial program S are related by $\sim$ to nodes of the optimized program A. Each node correspondance $\sim$ relation bears an invariant, that is a predicate on variables of both A and S.
- Coupling proofs are probabilistic product programs, [Barthe+ POPL'2017]

Thanks

- Questions?
- Comments?
- I'm looking for help