

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Use of tools in teaching logic

François Schwarzentruher
ENS Rennes, France

ICLA - Panel on logic education

March 4, 2021

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Why tools for teaching logic?



motivated student

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Why tools for teaching logic?

making logic
easy to learn

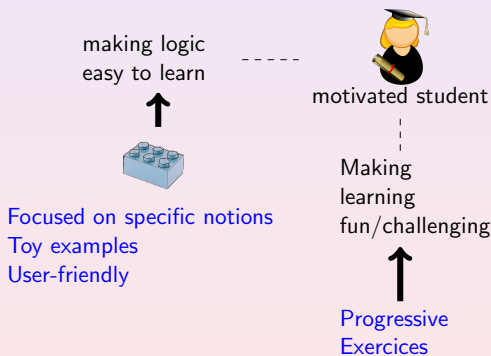


Focused on specific notions
Toy examples
User-friendly



motivated student

Why tools for teaching logic?



First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 Conclusion

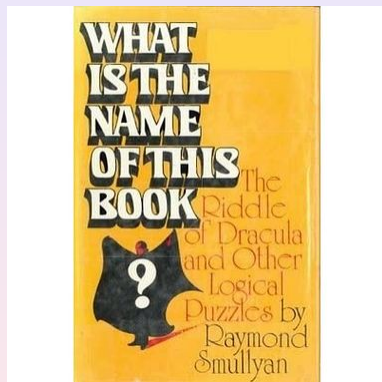
First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Textbooks and recreational Books



- Teaching with toy examples
- Very progressive list of exercises

First type of tools: books!

Second type of tools: teaching logic itself

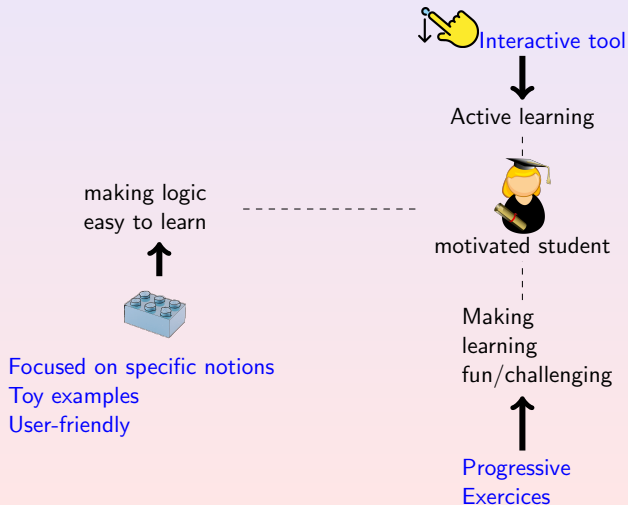
Third type of tools: teaching logic via applications

Conclusion

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 Conclusion

Why tools for teaching logic?



Interactive tools

<http://logicinaction.org/>

Modal Logic Playground

[Edit Model](#) [Evaluate Formula](#) [Link to Current Model](#)

Number of propositional variables:
 1 2 3 4 5

No state selected


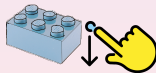
Click in the open space to **add a state**
 Drag between states to **add a transition**
 Ctrl-drag a state to **move** graph layout
 Click a state or a transition to **select** it

When a state is selected:

- R** toggles reflexivity
- Delete** removes the state

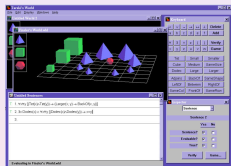
When a transition is selected:

- L**(left), **R**(right), **B**(both) change direction
- Delete** removes the transition

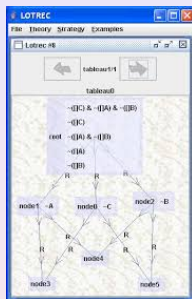



Play with epistemic puzzles with DEMO and SMCDEL:
<https://github.com/jrclogic/SMCDEL>

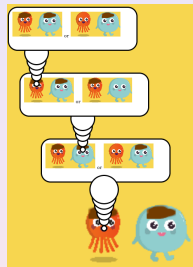
Toy tools for first-order and modal logics



Tarski's World
First-order logic



Kripke's World (Lotrec)
Modal logics



Hintikka's World
Dynamic Epistemic
Logic

~ serious games

Tools for teaching formal proofs

```

1 not (exists x P(x)), P(x) |- P(x)
2 not (exists x P(x)), P(x) |- exists x P(x)
3 not (exists x P(x)), P(x) |- not (exists x P(x))
4 not (exists x P(x)), P(x) |- bottom
5 not (exists x P(x)) |- not P(x)
6 not (exists x P(x)) |- forall x not P(x)
7 |- not (exists x P(x)) -> forall x not P(x)
8
9

```

- ✓ axiom
- ✓ intro exists(1)
- ✓ axiom
- ✓ elim not(2,3)
- ✓ intro not(4)
- ✓ intro forall(5)
- ✓ intro impl(6)

reset

$$\frac{}{\neg(\exists x P(x)), P(x) \vdash P(x)} \text{axiom}$$

$$\frac{}{\neg(\exists x P(x)), P(x) \vdash \exists x P(x)} \text{intro exists}$$

$$\frac{}{\neg(\exists x P(x)), P(x) \vdash \neg(\exists x P(x))} \text{axiom}$$

$$\frac{}{\neg(\exists x P(x)), P(x) \vdash \perp} \text{elim not}$$

$$\frac{}{\neg(\exists x P(x)) \vdash \neg P(x)} \text{intro not}$$

$$\frac{}{\neg(\exists x P(x)) \vdash \forall x \neg P(x)} \text{intro forall}$$

$$\frac{}{\vdash \neg(\exists x P(x)) \rightarrow \forall x \neg P(x)} \text{intro impl}$$

THE INCREDIBLE PROOF MACHINE

Current task:

$$\frac{(\exists x.P(x)) \rightarrow A}{\forall x.P(x) \rightarrow A}$$

Block count:

3
6

Logic blocks:

$P(x) \supset \forall \forall x.P(x)$

$\forall x.P(x) \supset \forall \forall P(y)$

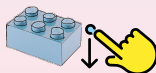
$P(y) \supset \exists \exists x.P(x)$

$(\exists x.P(x)) \rightarrow A$

$\forall x.P(x) \rightarrow A$

$\forall x.P(x) \rightarrow A$

$\forall x.P(x) \rightarrow A$



Many toy tools: Panda, Pravda, Pandora, $\mathcal{J}\forall P\exists$, Proof lab, Hyperproof (Openproof project), Bonsai, <http://incredible.pm/>, etc.

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Coq and Geogebra = Geocoq

<http://geocoq.github.io/GeoCoq/>

The screenshot shows the GeoGebra interface. The main window displays a geometric construction with several lines and points labeled A, B, C, D, E, F, G. The construction involves lines c , d , e , f , g and points A , B , C , D , E , F , G . The Coq proof window on the left contains the following code:

```
commandTitle | stateTitle |
passThrough_CA : c = line C A.
N.
isTLine_BC : d = tLine B c.
N.
isTLine_Ca : e = tLine C a.
O.
isIntersectPointOf_cd : D = intersectionPoint c d.
O.
isIntersectPointOf_ae : E = intersectionPoint a e.
I.
passThrough_DE : f = line D E.
O.
isMidof_ED : F = midPoint E D.
O.
isMidof_BC : G = midPoint B C.
N.
passThrough_GF : g = line G F.
```

The theorem window at the bottom is empty.

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Progressive exercises - interface inspired from video games



Progressive map in SuperTux

THE INCREDIBLE PROOF MACHINE

What do you want to prove today?

Session 1

A	A B	A B	A B
A	A	B A	AAB
A	AAB	AAB	AAB
AAA	A	A B	AAB
AAB	(AAB)AC	(AAB)AC	(AAB)AC
BAA	A B C	AxC	AA(BAC)
A			
A			
Add			

Session 2

A A-B	A A-B B-C	A A-B A-C B-D	A A-A
B	C	C-D	A
		D	
A-B B-C	A-B A-(B-C)	A-A	A-C B-C AAB
A-C	A-C		C

Progressive map
in the incredible proof machine

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
 - Database
 - CPU design
 - Constraint-solving tools
 - Program Verification
- 4 Conclusion

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

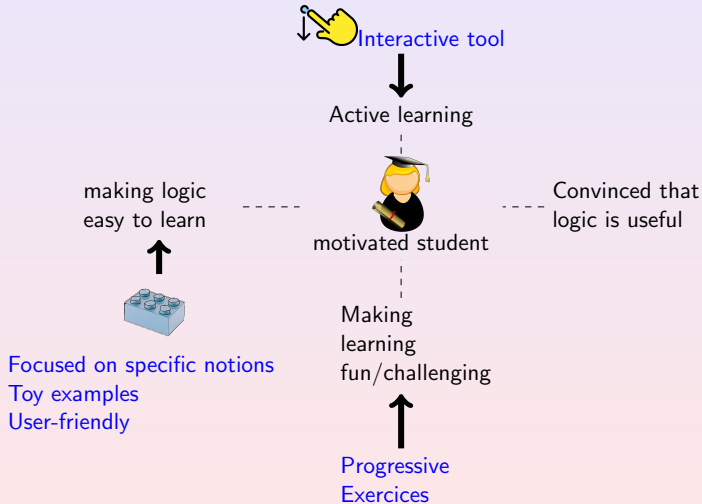
Database

CPU design

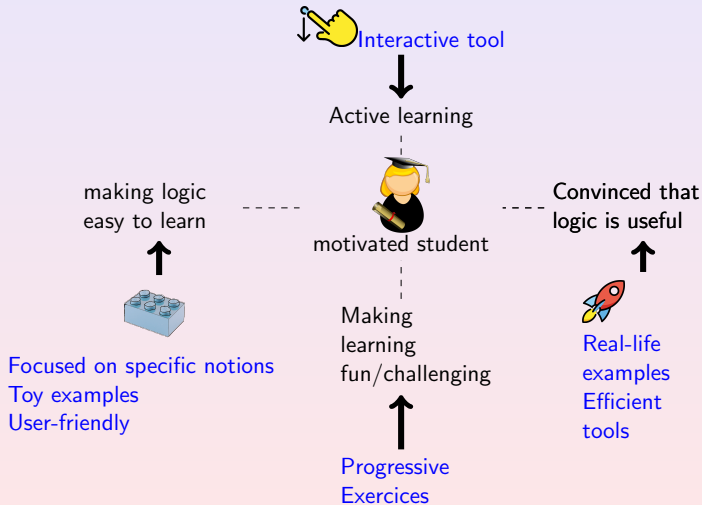
Constraint-solving tools

Program Verification

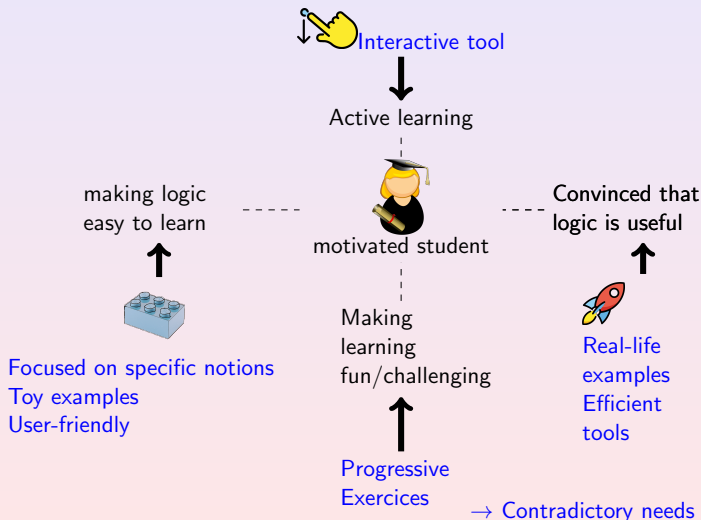
Why tools for teaching logic?



Why tools for teaching logic?



Why tools for teaching logic?



First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Database

CPU design

Constraint-solving tools

Program Verification

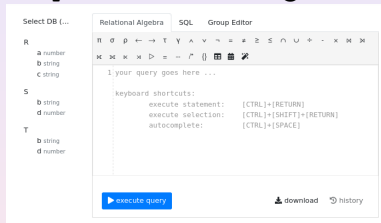
Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
 - Database
 - CPU design
 - Constraint-solving tools
 - Program Verification
- 4 Conclusion

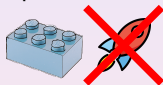
Database

RelaX

<https://dbis-uibk.github.io/relax/landing>



Specific for teaching relational algebra



VS

Direct manipulation of SQL



First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Database

CPU design

Constraint-solving tools

Program Verification

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
 - Database
 - **CPU design**
 - Constraint-solving tools
 - Program Verification
- 4 Conclusion

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

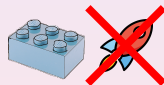
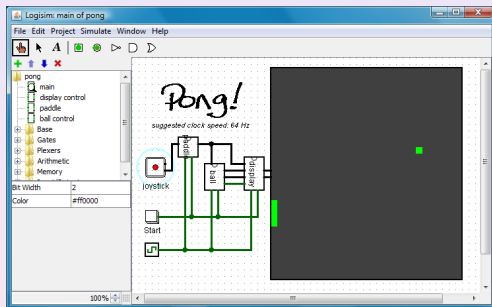
Database

CPU design

Constraint-solving tools

Program Verification

CPU design



Logisim for student to understand computer architectures

- Draw Boolean circuits
- Simulation

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Database

CPU design

Constraint-solving tools

Program Verification

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
 - Database
 - CPU design
 - **Constraint-solving tools**
 - Program Verification
- 4 Conclusion

SAT, SMT, FO Solvers + user-friendly software

The screenshot shows the Touist (english) v3.2.0-17-g4c5aaf1 interface. The left pane displays a logical encoding in Prolog-like syntax for a game involving matches and turns. The right pane shows the corresponding formal logic representation using quantifiers and logical connectives.

```

1 sM = 4
2 sM = {0..sM} :: set of matches
3 sT = {0..sM} :: set of turns
4
5 :: reste(1..n) = il reste n allumettes au temps
6 :: prend(1,2) = au temps 1, 2 allumettes si
7
8 exists prend(0,2) :: joueur 0 (nous)
9 forall prend(1,2) :: joueur 1 (adversaire)
10 exists prend(2,2) :: \ \ :: joueur 0 (nous)
11 forall prend(3,2) :: joueur 1 (adversaire)
12 exists prend(4,2) :: \ \ :: joueur 0 (nous)
13
14 :: But
15 not 0_a_perdu \ \
16 :: Conditions initiales
17 and reste(0, sM)
18 and tour_de_0(0) \ \
19 and
20
21 :: Resultat de l'action de la prise : si on
22 :: le nombre d'allumettes.
23 bigand st,sn in sT,0N when sM>2:
24 (reste(st,sn) and prend(st,2) => reste(st+
25 (reste(st,sn) and not prend(st,2) => reste
26 end \ \
27 and
28
29 :: On ne peut pas prendre 2 allumettes si il
30 bigand st in sT:
31 reste(st,1) => (not prend(st,2) and reste:
32 end \ \
33 and
34

```

$$\begin{aligned}
 & N_b \leftarrow 4 \\
 & N \leftarrow \{0..N_b\} \\
 & T \leftarrow \{0..N_b\} \\
 & \exists \text{prend}_{0,2}, \forall \text{prend}_{1,2}, \exists \text{prend}_{2,2}, \\
 & \forall \text{prend}_{3,2}, \exists \text{prend}_{4,2}, \\
 & \sim 0_a_perdu \\
 & \wedge \text{reste}_{0,N_b} \wedge \text{tour_de_0} \\
 & \wedge \bigwedge_{s,t \in T, N} \left(\text{reste}_{s,t} \wedge \text{prend}_{s,2} \Rightarrow \text{reste}_{s+1,t} \right. \\
 & \left. \wedge \bigwedge_{s,t \in T} \left(\text{reste}_{s,t} \Rightarrow (\sim \text{prend}_{s,2} \wedge \text{reste}_{s,t+1}) \right) \right) \\
 & \wedge \bigwedge_{s,t \in T} \bigvee_{s \in N} \text{reste}_{s,t} \\
 & \wedge \bigwedge_{s_1, s_2 \in T, N} \left(\text{reste}_{s_1, s_1} \Rightarrow \bigwedge_{s_2 \in N} \sim \text{reste}_{s_1, s_2} \right) \\
 & \wedge \bigvee_{s \in T} \left(\text{tour_de_0}_s \wedge \text{reste}_{s,0} \wedge \sim \text{reste}_{s-1,0} \right) \\
 & \wedge \bigwedge_{s \in T} \left(\sim \text{tour_de_0}_s \Leftrightarrow \text{tour_de_0}_{s+1} \right)
 \end{aligned}$$


Course with Logic4Fun, Touist

- Modelling problems (e.g. Sudoku, planning, etc.)
- Appreciate the succinctness/expressivity of a logical language
- Compare the size of encodings VS the complexity of the logic

Higher-level constraint languages

The screenshot displays the Alloy Analyzer 4.1.10 interface. The top window shows the source code for a puzzle, and the bottom window shows the generated state transition graph.

Source Code (Alloy Analyzer 4.1.10):

```

/private/var/folders/dz/dzKqURnBGQOSnZtGS-OQk+.../models/examples/puzzles/farmer.als
Alloy Analyzer 4.1.10 (build date: 2009/03/19 02:02 EDT)

Warning: Alloy4 defaults to SAT4J since it is pure Java and very reliable.
For faster performance, go to Options menu and try another solver like MiniSat.
If these native solvers fail on your computer, remember to change back to SAT4J.

Executing "Run solvePuzzle for 8 State expect 1"
Solver=sat4j Binwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=OFF
767 vars. 66 primary vars. 1511 clauses. 181ms.
Instance found. Predicate is consistent, as expected. 61ms.

/*
 * The classic river crossing puzzle. A farmer is carrying a fox, a
 * chicken, and a sack of grain. He must cross a river using a boat
 * that can only hold the farmer and at most one other thing. If the
 * farmer leaves the fox alone with the chicken, the fox will eat the
 * chicken; and if he leaves the chicken alone with the grain, the
 * chicken will eat the grain. How can the farmer bring everything
 * to the far side of the river intact?
 */
/*
 * authors: Greg Dennis, Rob Seater
 */

open util/ordering[State] as ord

/*
 * The farmer and all his possessions will be represented as Object
 * Some objects eat other objects when the Farmer's not around.
 */
abstract sig Object { eats: set Object }
one sig Farmer, Fox, Chicken, Grain extends Object {}

/*
 * Define what eats what when the Farmer's not around.
 * Fox eats the chicken and the chicken eats the grain.
 */
fact eating { eats = Fox->Chicken + Chicken->Grain }

/*
 * The near and far relations contain the objects held on each
 * side of the river in a given state, respectively.
 */
sig State {
  near: set Object,
  far: set Object
}

/*
 * In the initial state, all objects are on the near side.
 */
Line 15, Column 33
  
```

State Transition Graph (farmer) Run solvePuzzle for 8 State expect 1:

The graph shows 8 states (State0 to State7) and three objects: Farmer, Fox, and Chicken. Edges represent transitions between states, labeled with actions like 'near', 'far', 'eat', and 'eat'. A path is highlighted from State0 to State7, showing the sequence of moves to solve the puzzle.



Prolog, Answer set programming <https://potassco.org/>, Alloy

Teaching algorithmics behind logic

$$(p_1 \vee \neg p_3) \wedge (p_2 \vee p_3 \vee \neg p_1)$$

DIMACS format

1 -3 0

2 3 -1 0

Implement a SAT solver from scratch

- Direct reductions to the DIMACS format
- DPLL algorithm
- Heuristics
- Backjumping and Clause learning

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Database

CPU design

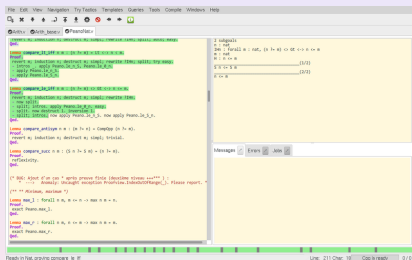
Constraint-solving tools

Program Verification

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
 - Database
 - CPU design
 - Constraint-solving tools
 - Program Verification
- 4 Conclusion

Proof assistant for program verification



```
File Edit View Navigation Try/Tests Templates Guides Tools Console Windows Help
[Navigation icons]
Activity: [File Browser] [File Explorer]
[Code Editor]
def
  loop (n: nat) : nat := fun n => if n < 0 then 0 else n
  loop 10
end
[Messages Panel]
Messages [Errors] [Info]
[Status Bar]
Line: 211 Char: 18 Copy/Ready 0/0
```

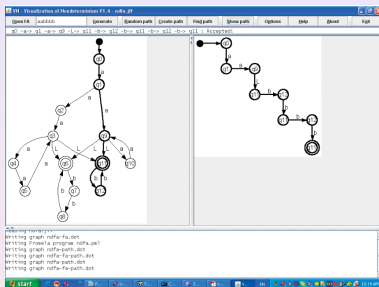


Course based on Scala and Isabelle/HOL

<http://people.irisa.fr/Thomas.Genet/ACF/>

- Either prove that a function is correct w.r.t. a specification
- or generate a counterexample

Model checking and temporal logic



The Spin model checker is one of the leading verification tools used by professional software engineers, but to my surprise I found that it is eminently suitable as a teaching tool. 📄 [Ben Ari, ACM 2010]

<http://spinroot.com/spin/Doc/p40-ben-ari.pdf>

Other tools: NuSMV, uPPAAL, MCMAS, etc.

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Learning analytics

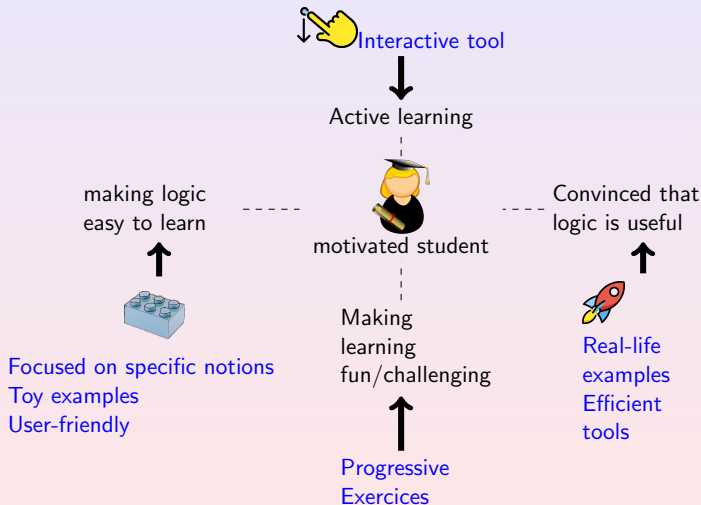
Tools as packages

Logicians and the rest of world

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 **Conclusion**
 - Learning analytics
 - Tools as packages
 - Logicians and the rest of world

Summary: Why tools for teaching logic?



Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 **Conclusion**
 - Learning analytics
 - Tools as packages
 - Logicians and the rest of world

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Learning analytics

Tools as packages

Logicians and the rest of world

Learning analytics

How do we prove/measure:


- the benefits of using some tools?
- the quality of toy examples?
- the relevance of interactivity?
- the quality of the progression in exercises?

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 **Conclusion**
 - Learning analytics
 - **Tools as packages**
 - Logicians and the rest of world

Tools are used and... reused!

“The SAT solver used as backend of Satoulouse a SAT GUI interface for teaching is SAT4J. The first advantage of this solver is that it is written in JAVA so it is easy to call it from the GUI.”

 [Gasquet et al. TICTTL 2011]



Designers of new pedagogical tools need:

- some other tools from the shelf
- Python and Javascript packages ready to use

Outline

- 1 First type of tools: books!
- 2 Second type of tools: teaching logic itself
- 3 Third type of tools: teaching logic via applications
- 4 **Conclusion**
 - Learning analytics
 - Tools as packages
 - **Logicians and the rest of world**

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Learning analytics

Tools as packages

Logicians and the rest of world

Pedagogical tools: an opportunity to reach other people!



Example (s)

- Tools for logic and argumentation?
- Tools for logic and art?
- Tools for teaching logic to the machine learning community?
 - Logical Neural Networks. Logic for guiding learning
 - Boolean circuits as explanation for machine learning process
 - Describe goals in reinforcement learning with temporal logic.

Special thanks

- ICLA organizers
- Thomas Genet (IRISA, Rennes) (program verification)
- Simon Rockiki (ENS Rennes) (architecture)
- Zoltan Miklos (IRISA, Rennes) (database)
- Serena Vilata (INRIA, Sophia Antipolis) (argumentation)
- Srdjan Vesic (CNRS, Lens) (argumentation)
- Fahima Djelil (IMT Atlantique) (learning analytics)

First type of tools: books!

Second type of tools: teaching logic itself

Third type of tools: teaching logic via applications

Conclusion

Learning analytics

Tools as packages

Logicians and the rest of world

Thank you for your attention!

