Correspondence between proofs and programs

An epistemological reading for a didactical perspective

Simon Modeste

IMAG, University of Montpellier, CNRS
Montpellier, France
simon.modeste@umontpellier.fr

Project funded by the French National Research Agency: ANR-16-CE38-0006-01

Octobre 28-30, 2019

Context and questions

Computer Science at school

Looking for its place...

- Tool or object?
- Programming or using software?
- Which links with mathematics?

Computer Science at school

Looking for its place...

- Tool or object?
- Programming or using software?
- Which links with mathematics?

In France, now (after many reforms, still ongoing):

- CS in middle school (12-14 years): in maths and technology courses
- CS in high school (15-18 years): in maths courses, and specific (optional) CS courses

Computer Science at school

- Need for development of didactics of CS (as a scientific topic)
- Epistemological needs for this didactical purposes

Computer Science at school

- Need for development of didactics of CS (as a scientific topic)
- Epistemological needs for this didactical purposes

In particular, in links with mathematics:

- Many links between maths and CS
- Some common foundations between CS and maths
- Some concepts, theories and tools could from didactics of maths could be relevant for didactics of CS

Computer Science at school

- Need for development of didactics of CS (as a scientific topic)
- Epistemological needs for this didactical purposes

In particular, in links with mathematics:

- Many links between maths and CS
- Some common foundations between CS and maths
- Some concepts, theories and tools could from didactics of maths could be relevant for didactics of CS

In this communication

Specific epistemological dimension of the interactions between mathematics and Computer Science, in a didactical perspective: common foundations, role of logic, and links between the concepts of proof, algorithm and program.

Logic and proof in Mathematics and Computer Science

Some links between maths and CS

- Common/close logical foundations
- Role of language and relation to formalism
- A specific type of validation: proof

Some links between maths and CS

- Common/close logical foundations
- Role of language and relation to formalism
- A specific type of validation: proof

Hypothesis

First order logic, is relevant as a model to understand many didactical issues in maths, CS and their interactions.
In particular, it permits to highlight syntax-semantics dialectic.

Semantic perspective

In mathematics:

- **Semantics** concerns the the relation of signs to the objects they represent.
- **Syntax** concerns the relation of signs between them.
- **Pragmatics** concerns the articulation between syntax and semantics that is built by a subject.

→ Permits to define a semantic conception of truth.

Semantic perspective

In mathematics:

- **Semantics** concerns the the relation of signs to the objects they represent.
- **Syntax** concerns the relation of signs between them.
- **Pragmatics** concerns the articulation between syntax and semantics that is built by a subject.

→ Permits to define a semantic conception of truth.

In computer Science, the syntax-semantic distinction is somewhat more obvious

- **Syntax** can refer to the "rules of composition" of a valid text in a chosen programming language
- **Semantics** refers to the "expected" effects on the actual machine of each construct on that language and of their combinations

Formalisation

Similarities :

- Describing an algorithm as a machine-executable program
- Translating an informal math statement in some formal language

Formalisation

Similarities :

- ◇ Describing an algorithm as a machine-executable program
- ◇ Translating an informal math statement in some formal language

An algorithm can can be described informally and its formalisation (implementation) needs removing any possible ambiguity, and ensuring rendering correctly the ideas and principles permitting the algorithm to solve the problem at hand.
→ Pragmatics

Proof, algorithm and program

Proof

We call proof (in Mathematics or in Computer Science) a finite sequence of statements organized according some determined rules (explicitly or implicitly) in order to convince someone of the truth of a statement. The granularity of the details of the proof generally depend on the source and the receiver of the proof.

Formal proof

A formal proof is a text consisting in a finite sequence of statements, expressed in a well defined formal language, where the statements are deduced from the previous ones or from axioms following predefined deduction rules.

It is commonly admitted by mathematicians or computer scientists that any proof could be expressed as a formal proof (if the axioms and deduction rules were made explicit).
Nowadays, formal proofs are often produced using software called proof assistants which allows the automatic verification of proofs.

Algorithm

An algorithm is a finite sequence of organized instructions, that describes how to solve a problem, that is, how to obtain a defined goal starting from given data. The steps must be considered as elementary by the recipient of the algorithm, and the algorithm must not be ambiguous. In other words, the producer of the algorithm and its recipient must agree on the granularity of the details of the algorithm.

Program

A program is a text consisting of a finite sequence of instructions, written in a well defined (programming) language, that is, having a precise syntax (structure of the language) and semantic (effect of each instruction). An algorithm can be described with a program.

Two dialectics

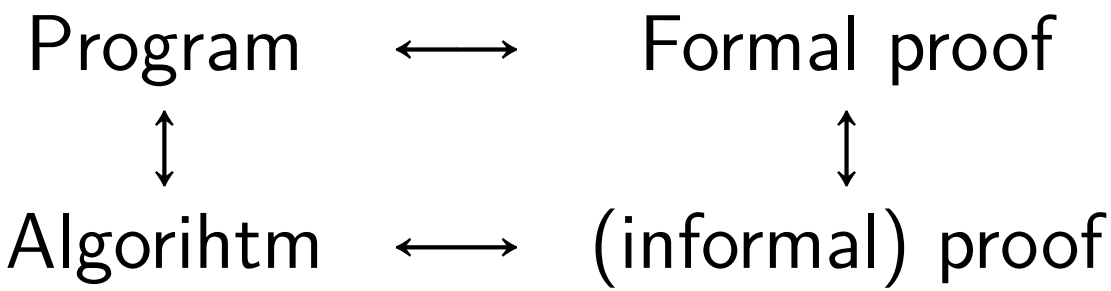
Formal-informal dialectic

- Informal: in general written to a "human" receiver, with a specific level of detail (strong semantic dimension)
- Formal: for a machine, or to be completely unambiguous and avoid "mistakes" (strong syntactic dimension)

Two dialectics

Validation-construction dialectic

- Can be analysed in terms of the Curry-Howard Correspondence (identification between proofs and programs)
- Transposition to the informal case: a constructive proof contains an algorithm, an algorithm (sometimes with its own proof) can be seen as a proof of a statement.


Program
Formal proof
Algorihtm
(informal) proof

At secondary school

At secondary school

Program

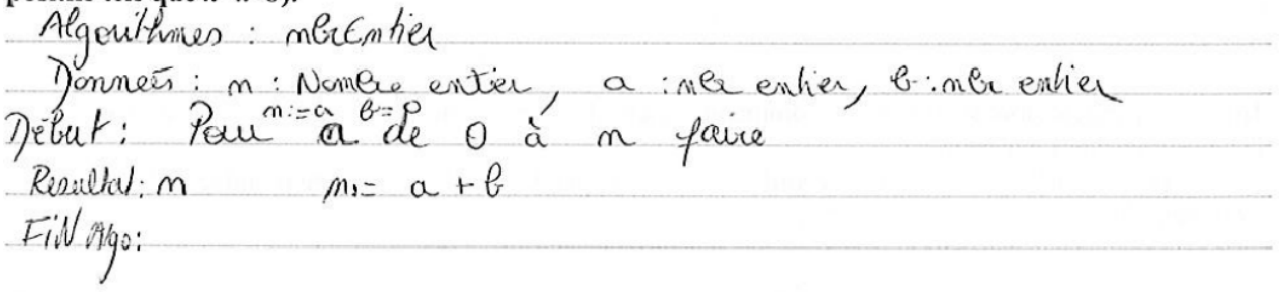
At secondary school

Program

Algo

At secondary school

Écrire un algorithme pour trouver toutes les écritures d'un nombre entier naturel *n* strictement positif en somme de deux entiers naturels strictement positifs (c'est-à-dire, les couples (*a*,*b*) d'entiers naturels positifs tels que *n*=*a*+*b*).



Translation :
Write an algorithm to find all the decompositions of a positive integer n as the sum of two positive integers (that is, the couples (a, b) such that $n = a + b$).
Algorithm: nbrInteger
Data: n:integer, a:integer, b:integer
n:=a b:=0
Begin: For a from 0 to n do
Result: n *n:=a+b*
End Algo:

At secondary school

Program

Algorithm

At secondary school

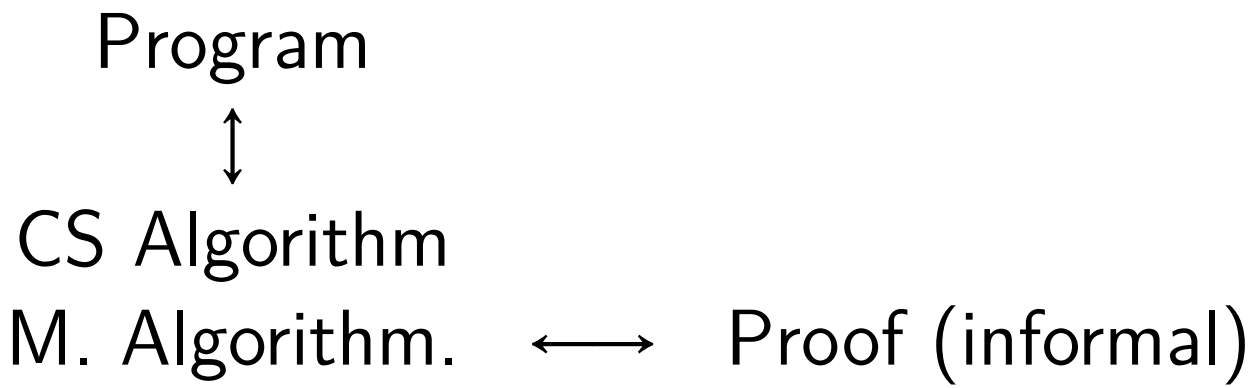
Program

CS Algorithm
M. Algorithm.

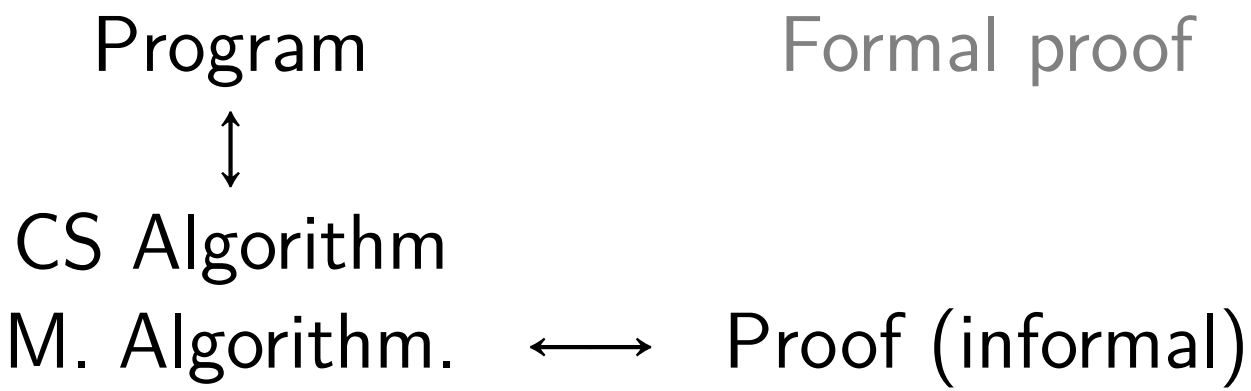
At secondary school

Program
↕
CS Algorithm
M. Algorithm. ⟷

At secondary school



At secondary school



At secondary school

Program	??	Formal proof
↕		?
CS Algorithm		?
M. Algorithm.	⟷	Proof (informal)

Didactical issues

Formal aspects at school

- Difficulties for students to understand proof
- Educative software to manage proof (proof assistants)
- Place of formal proof at school
- Some software taking in charge the syntactic aspects

Program-proof correspondence

- ◇ Some similar "gestures" in proving, designing algorithm? writing formal proof, programming?
- ◇ Syntax-semantic dialectic involved

→ Program/algorithm design and proof building as problem solving.

Issues and perspectives

- What type of logical model /theory can be adequate to address both math and CS activities (at secondary school) ?
- How far this reading of the Curry-Howard correspondence can be productive (in a didactical view)?
- Can working on formal proofs can be helpful for pupils in mathematics and CS?