An Educational Vision

The Message of Programming

Gerald Jay Sussman

CSAIL & EECS

Massachusetts Institute of Technology

& the Free Software Foundation

Programming

- Programming is a linguistic phenomenon, like mathematics or English.
- In programming there is both Prose and Poetry.
  - Some programs are useful.
  - Some programs are beautiful.
  - Some programs are sad.
- Programming provides novel means for us to express ourselves, as individuals.
- Programming helps clarify ideas:
  - We must be precise.
  - We must be unambiguous.

Teaching and Learning are Hard

The traditional means we have of communicating ideas are weak by comparison to means that are available in programming languages!

In ordinary discourse the referents of identifiers are not clearly delimited or locally determined.

- Math:
  “What does Φ mean here?”
- History:
  “Edward Plantagenet was murdered. Which one?”
- Literature:
  “... that unctuous man? Uriah Heep or Mr. Chadband?”

Scope of Identifiers

In computer languages we have

- *declarations*

with well-defined

- *scope*.

The scope of an identifier is a region of text in which the referent of that identifier is clearly specified.

```
(let ((king (British-monarch ’(George II)))
      (birth (date ’(30 October 1683)))
      (death (date ’(25 October 1760))))
  (let ((lifetime
          (let ((- (date-arithmetic ’-)))
            (- death birth))))
    ...stuff about King George II...))
```

A Computational Advantage

A good hypertext document can link every occurrence of an identifier with the place it was introduced or defined.

There can be linkages to information about the identifier other than just an introduction or definition.

Part of the power of Wikipedia is the fact that it is actively and extensively cross-referenced.

Another part of the power is that it is not proprietary!

- This makes it possible to get contributions from everyone and everywhere.

Bugs

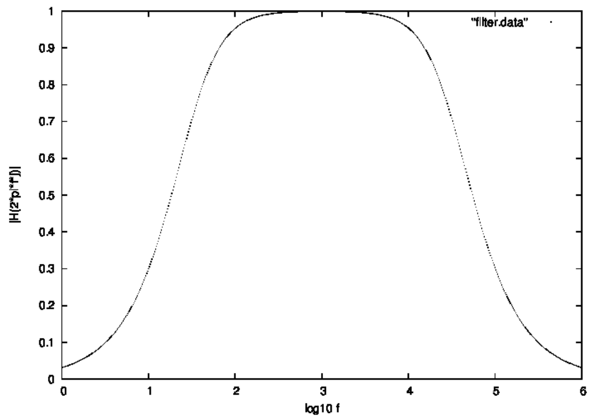
We *always* have bugs!

Why?

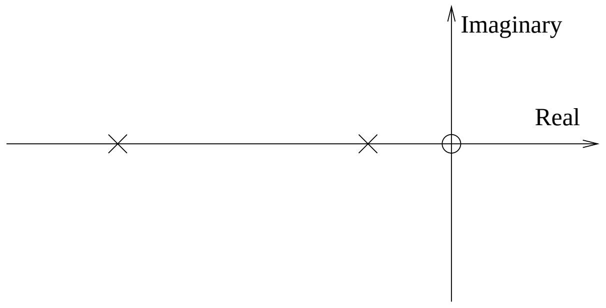
- Inadequate specification
- Unstable goals
- Unanticipated interactions
- PSBDARP!
  (Problem Solving By Debugging Almost-Right Plans)

PSBDARP—Electronic Design

Suppose I want a filter with a frequency response:



This can be accomplished with a system function with 2 poles and a zero:

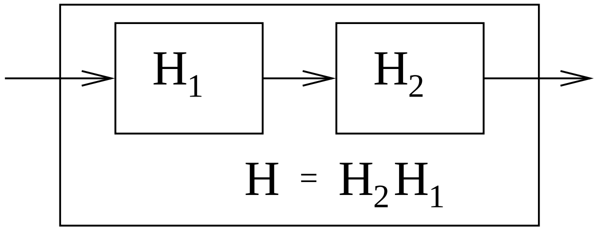


A Plan

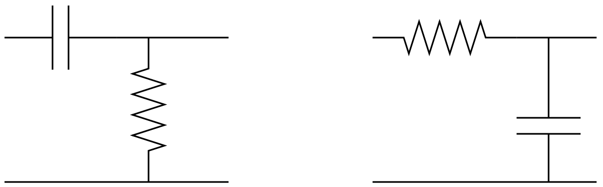
This can be synthesized by factoring the system function $H(s) = H_1(s)H_2(s)$:

$$H_1(s) = \frac{s}{s + 1/\tau_1} \quad \text{and} \quad H_2(s) = \frac{1/\tau_2}{s + 1/\tau_2}$$

which can be implemented with a cascade:

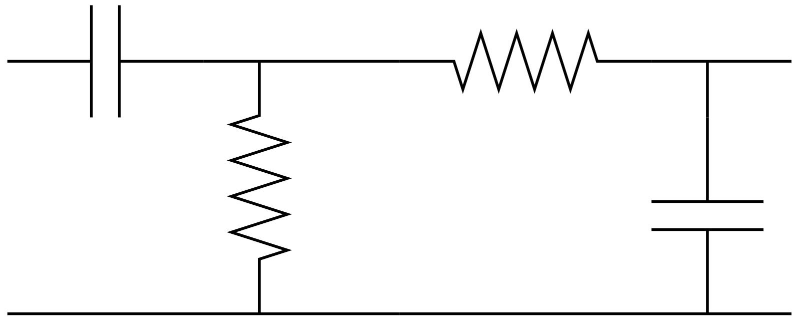


Where the factors are implemented as:

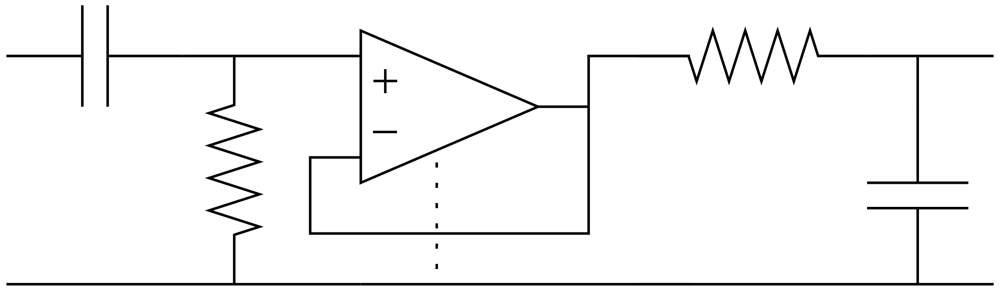


A Bug! and a Patch

But the cascade doesn't work because the second circuit "loads" the first one.



But for a loading bug in a cascade plan we have a fix:



PSBDARP is a powerful idea

A powerful strategy for solving problems is to use almost-right plans. We know that each of these plans often fails in characteristic ways. For each failure mode of a plan we have appropriate actions that we can take to fix the bug or propose an alternate plan.

This is an important idea

- It is applicable to all kinds of problem solving
  - engineering, science, math, music, ...
  - fundamental to the way we think!
- We don’t teach it explicitly to our students
- We don’t use it in our teaching

Debugging

If a compound mechanism fails to have the behavior that was intended then there are only two ways that can occur: some parts do not do what was expected of them, or the interconnection of the parts does not correctly achieve the goal.

How do we debug a student?

- What is the student's plan?
- Is the student's plan applicable to the problem?
- Is the arrangement of parts correct?
- Is there a part of the plan that is wrong?

How do we incorporate THIS plan into our teaching?
Can we teach students to debug themselves?

What is a Plan?

From architecture

- Understanding the problem
- Find a *parti*—an organizing principle for the design
  - Usually a sketch
  - May embody abstract ideas
    - served spaces
    - servant spaces
  - Decomposition separates out
    - infrastructural support
    - spaces to be supported
- Fill in the parti

Plans

All engineering (and programming) starts with plans. Probably most other human endeavors are also planned.

For example, a parti for a multistep numerical integrator for second-order systems is just

$$x(t+h) = 2x(t) - x(t-h) + h^2 \sum_{j=0}^{k} A(j)\ddot{x}(t-jh)$$

But the indicated subtraction and addition must be done with more precision than the addition of the weighted sums of accelerations.

The parti suppresses that complication.

Can we teach planning?

- Do WE have the skills of planning?
  - Is there a theoretical basis?
  - If not, we must develop one.
- Do we understand how to teach planning?
  - Is it said that it is hard to teach “design.”
  - Is this true? Have we tried?
- Do we have technology to support teaching design?
  - If not, can we develop it?

Not only do these ideas affect the way we will teach, they affect what we will teach. For example, debugging students’ problems is easier if the students know about debugging.

The Moral of this story

- We really don’t know much about teaching
- We really don’t know much about learning
- We really don’t know much about thinking

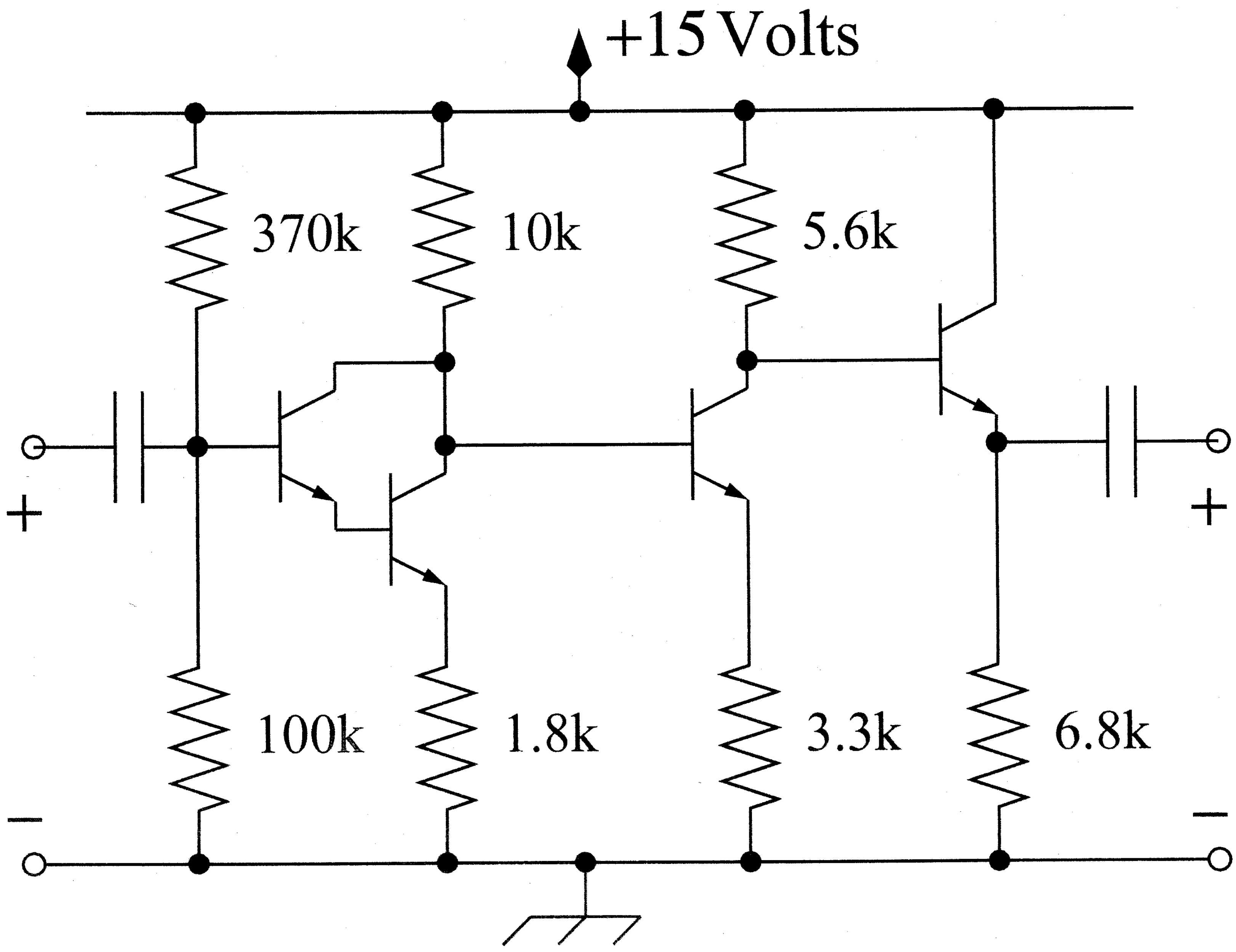
Seymour Papert taught us that Programming and Engineering gives us a lever for future success.

- Clarity and precision are important
- Planning and debugging are important

In such a foggy world, community is of the essence: Openness and sharing of ideas is necessary for a good educational future.

We must hang together

“We must, indeed, all hang together or, most assuredly, we shall all hang separately.”
—Benjamin Franklin


+15 Volts
370k
10k
5.6k
100k
1.8k
3.3k
6.8k
+
−
+
−