But, as Poincare observed, it is the man,
not the method, that solves a problem.

— E. T. Bell

To Optimize or Not to Optimize: Algorithm Design in VLSI CAD

Hai Zhou

Electrical Engineering and Computer Science
Northwestern University

Advantages of VLSI CAD or Its Difference From Mechanical CAD

- Any function can be automatically mapped to its implementation structure
- Thus it is also called "Design Automation"
- A long mapping process from very abstract functions to silicon and metal rectangles
- Extensive tool support from a broad industry

Challenges in VLSI CAD

- Moore's Law: #transistors on a chip is doubled every couple of years

Challenges in VLSI CAD

- Moore's Law: #transistors on a chip is doubled every couple of years
- It has happened for the past 30+ years: exponential increase

Challenges in VLSI CAD

- Moore's Law: #transistors on a chip is doubled every couple of years
- It has happened for the past 30+ years: exponential increase
- Communication from one corner to another takes substantial amount of time

Challenges in VLSI CAD

- Moore's Law: #transistors on a chip is doubled every couple of years
- It has happened for the past 30+ years: exponential increase
- Communication from one corner to another takes substantial amount of time
- Many previous ignorable physical effects become prominent and need to be considered, e.g. coupling noise, inductance, temperature

Challenges in VLSI CAD

- Moore's Law: #transistors on a chip is doubled every couple of years
- It has happened for the past 30+ years: exponential increase
- Communication from one corner to another takes substantial amount of time
- Many previous ignorable physical effects become prominent and need to be considered, e.g. coupling noise, inductance, temperature
- What we get on silicon is no longer what we see in design: process variations produce chips with random behaviors

Main Task in VLSI CAD: Algorithm Design

- Mapping from high level specification to low level design is always possible and straight-forward ("Silicon Compilation")
- But the main issue is optimization: small area, high performance, low power, low temperature

Main Task in VLSI CAD: Algorithm Design

- Mapping from high level specification to low level design is always possible and straight-forward ("Silicon Compilation")
- But the main issue is optimization: small area, high performance, low power, low temperature
- The problem sizes are huge and increasing: millions of devices

Main Task in VLSI CAD: Algorithm Design

- Mapping from high level specification to low level design is always possible and straight-forward ("Silicon Compilation")
- But the main issue is optimization: small area, high performance, low power, low temperature
- The problem sizes are huge and increasing: millions of devices
- The problems are complex: most are NP-hard

Main Task in VLSI CAD: Algorithm Design

- Mapping from high level specification to low level design is always possible and straight-forward ("Silicon Compilation")
- But the main issue is optimization: small area, high performance, low power, low temperature
- The problem sizes are huge and increasing: millions of devices
- The problems are complex: most are NP-hard
- New physical effects are increasing

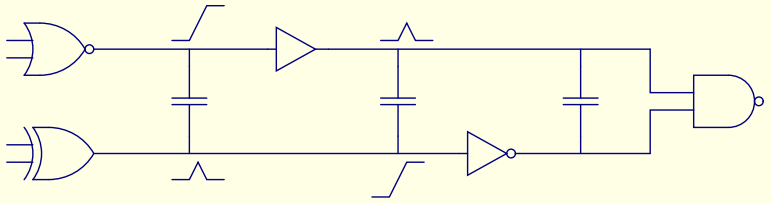
My Strategy for Algorithm Design in VLSI CAD

- Explore the structure of the problem
- Partition the problem into a sequence of simpler subproblems
- Design the most efficient algorithm for each subproblem

My Strategy for Algorithm Design in VLSI CAD

- Explore the structure of the problem
- Partition the problem into a sequence of simpler subproblems
- Design the most efficient algorithm for each subproblem
- We want to explore the structure for efficiency even a problem can be formulated as a Linear Program!

Gate Sizing for Coupling Noise Control



- Signal switching on a nearby wire will introduce a noise peak on a wire
- The noise is smaller is the driving gate of the victim is larger (stronger)
- The noise on a victim will be larger if the driving gate of an aggressor is larger
- How to find all gate sizes such that the noise on each wire is tolerable?

Gate Sizing for Coupling Noise Control

$$
\begin{array}{lll}
\textit{Minimize} & \sum_{i=1}^{n} w(i) * s(i) & \\
\textit{subject to} \quad N(i) \triangleq & f(s(i), s(i_1), \ldots, s(i_k)) \leq U(i) & \forall 1 \leq i \leq n \\
& l(i) \leq s(i) \leq u(i) & \forall 1 \leq i \leq n
\end{array}
$$

To Optimize

1. Find a feasible solution (gate sizes without noise violation)
2. Iteratively reduce gate sizes within feasible solutions

To Optimize

1. Find a feasible solution (gate sizes without noise violation)
2. Iteratively reduce gate sizes within feasible solutions

- But the main difficulty is how to find a feasible solution

To Optimize

1. Find a feasible solution (gate sizes without noise violation)
2. Iteratively reduce gate sizes within feasible solutions

- But the main difficulty is how to find a feasible solution
- Lagrangian relaxation? The noise function $N(i)$ may not be analytical

Not to Optimize

- Treat the optimization objective as another post-condition

$$(\forall s' : l \leq s' \leq u \wedge N(s') \leq U \Rightarrow \sum_{i=1}^{n} w(i) * s(i) \leq \sum_{i=1}^{n} w(i) * s'(i$$

$$Opt \triangleq (\forall s' : l \leq s' \leq u \wedge N(s') \leq U \Rightarrow s \leq s')$$

- Use Opt as an loop invariant of the algorithm
  - Satisfied by the initialization
  - Maintained in the loop

Not to Optimize

Sinha-Zhou Algorithm

`s := l;`

`do`

$\exists 1 \leq i \leq n : s(i) \leq u(i) \wedge N(i) > U(i)$

$\rightarrow s(i) :=$ `min size under` $N(i) \leq U(i)$

`od`

Correctness

Sinha-Zhou Algorithm

`s := l;`

`do` $\{\ Opt\ \}$

$\quad \exists 1 \leq i \leq n\ :\ s(i) \leq u(i) \wedge N(i) > U(i)$

$\qquad \rightarrow s(i) :=$ **`min size under`** $N(i) \leq U(i)$

`od` $\{\ Opt \wedge (\forall 1 \leq i \leq n\ :\ s(i) > u(i) \vee N(i) \leq U(i))\ \}$

Correctness

Sinha-Zhou Algorithm

`s := l;`

`do` $\{\ Opt\ \}$

$\quad \exists 1 \leq i \leq n\ :\ s(i) \leq u(i) \wedge N(i) > U(i)$

$\quad\quad \rightarrow s(i) :=$ **`min size under`** $N(i) \leq U(i)$

`od` $\{\ Opt \wedge (\forall 1 \leq i \leq n\ :\ s(i) > u(i) \vee N(i) \leq U(i))\ \}$

- Prove Opt is invariant based on math induction
- Based on the fact that $f(s(i), s(i_1), \ldots, s(i_k))$ is monotonically decreasing with increasing $s(i)$ and monotonically increasing with increasing $s(i_1), \ldots, s(i_k)$.

Interesting Features

- Algorithm is correct no matter what order is used for updating gates: "chaotic iterative scheme"

Interesting Features

- Algorithm is correct no matter what order is used for updating gates: "chaotic iterative scheme"
- The solution minimizes $\sum_{i=1}^{n} w(i) * s(i)$ no matter what $w \geq 0$ is

Interesting Features

- Algorithm is correct no matter what order is used for updating gates: "chaotic iterative scheme"
- The solution minimizes $\sum_{i=1}^{n} w(i) * s(i)$ no matter what $w \geq 0$ is
- Works for both continuous and discrete gate sizing

Interesting Features

- Algorithm is correct no matter what order is used for updating gates: "chaotic iterative scheme"
- The solution minimizes $\sum_{i=1}^{n} w(i) * s(i)$ no matter what $w \geq 0$ is
- Works for both continuous and discrete gate sizing
- Can be viewed as computing a fixpoint on a lattice:

  $$s(i) = \min(x : f(x, s(i_1), \ldots, s(i_k)) \leq U(i)) \stackrel{\Delta}{=} g(s(i_1), \ldots, s(i_k))$$

- Worst case running time is upper bounded by the height of the lattice, not its size

Interesting Features

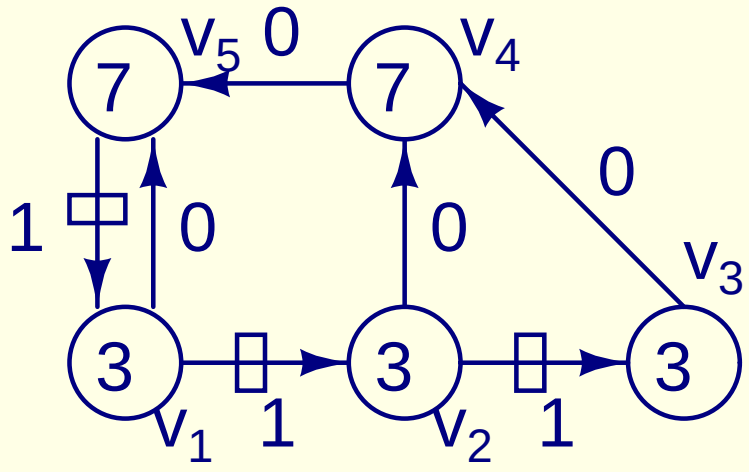
- Algorithm is correct no matter what order is used for updating gates: "chaotic iterative scheme"
- The solution minimizes $\sum_{i=1}^{n} w(i) * s(i)$ no matter what $w \geq 0$ is
- Works for both continuous and discrete gate sizing
- Can be viewed as computing a fixpoint on a lattice:

  $$s(i) = \min(x : f(x, s(i_1), \ldots, s(i_k)) \leq U(i)) \stackrel{\Delta}{=} g(s(i_1), \ldots, s(i_k))$$

- Worst case running time is upper bounded by the height of the lattice, not its size
- Very efficient practically: hundreds of thousands of gate can be sized within seconds

Problem of minimal period retiming

- Given a directed graph $G = (V, E)$ with delay $d : V \rightarrow \mathcal{R}^+$ and register number $w : E \rightarrow \mathcal{N}$
- Find a relocation of registers $w' : E \rightarrow \mathcal{N}$ such that the maximal delay between any consecutive registers is minimized



Notations

- relocation: $r : V \rightarrow \mathcal{N}$ represents # registers moved from outputs to inputs of each node

$$w'(u, v) = w(u, v) + r.v - r.u$$

- max path delay: $t : V \rightarrow \mathcal{R}^+$ the output arrival time of each node – need to satisfy

$$(\forall v \in V :: t.v \geq d.v)$$
$$(\forall (u, v) \in E : w(u, v) + r.v - r.u = 0 : t.v \geq t.u + d.v)$$

Problem formulation

$$
\begin{array}{rl}
\textit{Minimize} & max.t \\
\textit{Subject to} & (\forall (u,v) \in E :: w(u,v) + r.v - r.u \geq 0) \\
 & (\forall v \in V :: t.v \geq d.v) \\
 & (\forall (u,v) \in E \ : \ r.u - r.v = w(u,v) \ : \ t.v - t.u \geq d.v)
\end{array}
$$

Post-condition

$$
\begin{aligned}
P0(r) &\triangleq (\forall (u,v) \in E :: w(u,v) + r.v - r.u \geq 0) \\
P1(t) &\triangleq (\forall v \in V :: t.v \geq d.v) \\
P2(r,t) &\triangleq (\forall (u,v) \in E \,:\, r.u - r.v = w(u,v) \,:\, t.v - t.u \geq d.v) \\
P(r,t) &\triangleq P0(r) \wedge P1(t) \wedge P2(r,t) \\
P3 &\triangleq (\forall r', t' \,:\, P(r',t') \,:\, \max.t \leq \max.t') \\
Post &\triangleq P0 \wedge P1 \wedge P2 \wedge P3
\end{aligned}
$$

To Optimize

- Invariant: $P0$, $P1$, and $P2$
  – satisfied by initialization:

```
r,t:=0,d;
do
    (u,v)∈E∧w(u,v)=0∧t.v<t.u+d.v → t.v:=t.u+d.v
od
```

- Loop goal: $P3$ (optimality)

```
do
    ¬P3 → do something to improve
od
```

Loop condition: $\neg P3$

$$\begin{aligned}
\neg P3 &= \neg(\forall r', t' : P(r', t') : \max.t \leq \max.t') \\
&= (\exists r', t' : P(r', t') : \max.t > \max.t') \\
&= (\exists r', t' : P(r', t') : (\exists v \in V :: t.v > t'.v))
\end{aligned}$$

Furthermore, we can identify such $v \in V$:

$$\neg P3 \Rightarrow (t.v = \max.t \Rightarrow t.v > t'.v)$$

To improve...

...along the critical path for $t.v = \max . t$:

$$(\forall v : t'.v < t.v : (\exists u : t.u = d.u : r.u - r.v > r'.u - r'.v))$$

- Use a label $p : V \to V$ to identify u: `p.v:=u`
- Absolute values of r is not important: $r + c$ is the same as r
- Need to make r closer to r': `r.v:=r.v+1` or `r.p.v:=r.p.v-1`
- We select `r.v:=r.v+1`, which immediately makes `t.v:=d.v`

Effects of `r.v, t.v :=r.v+1, d.v`

- Violating $P2$: restored by adding to the loop

  `(u,v)`$\in$`E`$\wedge$`r.u-r.v=w(u,v)`$\wedge$`t.v-t.u<d.v` $\rightarrow$ `t.v,p.v:=t.u+d.v,p.u`

- May produce $v : t.v > \mathrm{maxT}$, where maxT is $\max.t$ before the loop: process v similarly by increasing $r.v$

  `(`$\exists$`r',t'::max.t'<maxT)`$\wedge$`t.v`$\geq$`maxT` $\rightarrow$ `r.v,t.v,p.v:= r.v+1,d.v,v`

- $P0$ may be violated if $(\exists(v,x) \in E :: w(v,x) + r.x - r.v = 0)$ when $r.v$ is increased: restored by adding

  `(u,v)`$\in$`E`$\wedge$`r.u-r.v>w(u,v)` $\rightarrow$ `r.v,t.v,p.v:=r.u-w(u,v),t.u+d.v,p.u`

```
r,t,p,maxT := 0,d,1,0;
do
   (u,v)∈E∧r.u-r.v=w(u,v)∧t.v-t.u<d.v → t.v,p.v := t.u+d.v,p.u
[]  maxT<t.v → maxT:=t.v
od
{P(r,t)∧max.t=maxT}
do
   (∃r',t'::max.t'<maxT)∧t.v≥maxT → r.v,t.v,p.v:= r.v+1,d.v,v
[]  (u,v)∈E∧r.u-r.v≥w(u,v)∧t.v<t.u+d.v → t.v,p.v := t.u+d.v,p.u
[]  (u,v)∈E∧r.u-r.v>w(u,v) → r.v,t.v,p.v:=r.u-w(u,v),t.u+d.v,p.u
[]  P(r,t)∧max.t<maxT → maxT:=max.t
od
{P(r,t)∧max.t=maxT∧(∀r',t'::max.t'≥maxT)}
```

How to check $(\exists r', t' :: max.t' < \mathrm{maxT})$

- Intuition: adjustment of r is conservative, thus, if $(\exists r', t' :: max.t' < \mathrm{maxT})$, the operations will finally stop
- $\{r.v - r.p.v < r'.v - r'.p.v\}$ `r.v:=r.v+1`
  $\{r.v - r.p.v \leq r'.v - r'.p.v\}$ or $\{r.v - r'.v \leq r.p.v - r'.p.v\}$
  This means that $(\max v \in V :: r.v - r'.v)$ cannot be increased
- Use $m.v$ to label the safe-guard $p.v$ when $r.v$ is increased, then

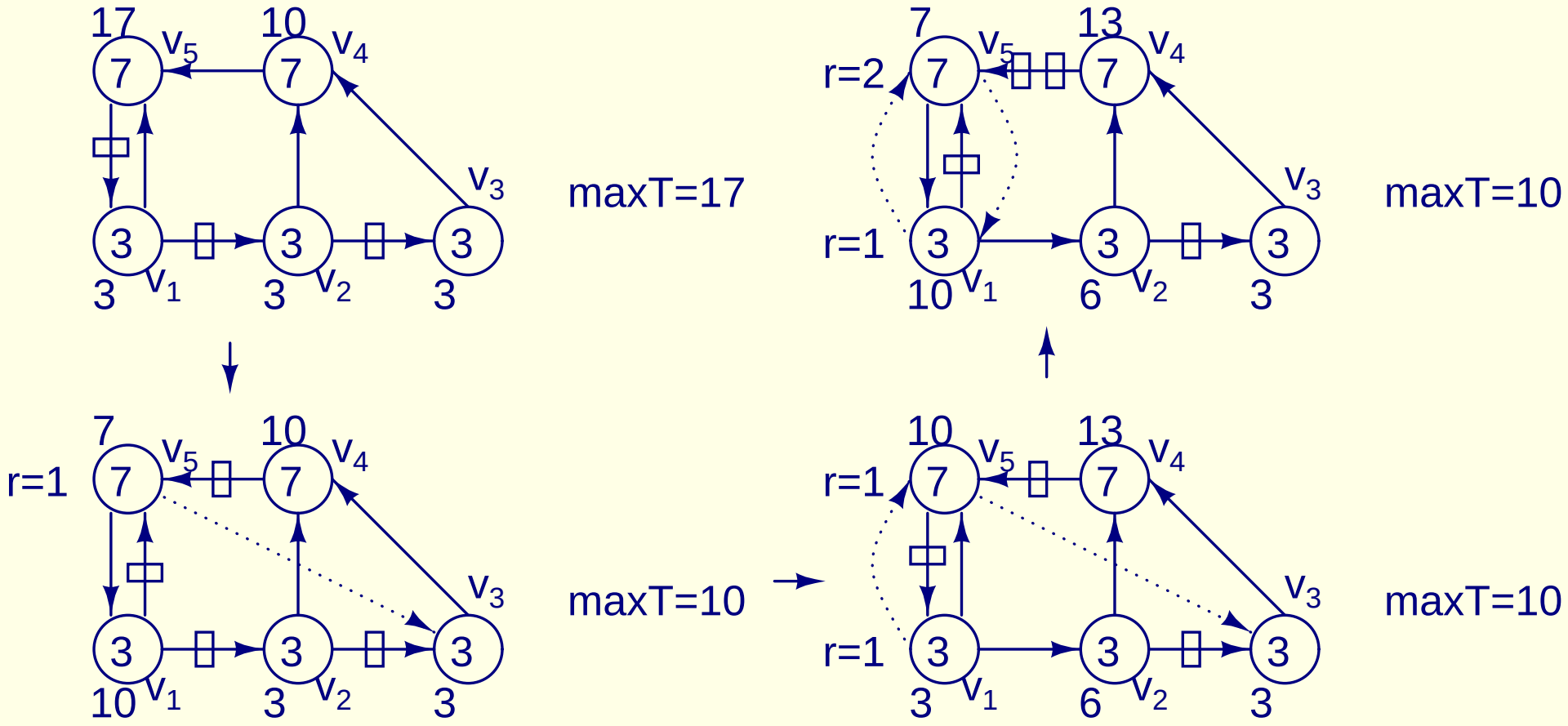
  $(\exists r', t' :: max.t' < \mathrm{maxT}) \Rightarrow (\forall v : m.v \in V : r.v - r.m.v \leq 1)$
- $(\exists r', t' :: max.t' < \mathrm{maxT})$ is not true if
  - m forms a cycle
  - $(\exists v \in V :: r.v > |V| - 1) \vee (\forall v \in V :: r.v > 0)$

Zhou's Algorithm

```
r,t,p,m,maxT,cycle:=0,d,1,0,0,0;
do
   (u,v)∈E∧r.u-r.v=w(u,v)∧t.v-t.u<d.v → t.v,p.v := t.u+d.v,p.u
[] maxT<t.v → maxT:=t.v
od
do
   ¬cycle∧t.v≥maxT →
      if
         m.v≠0 → cycle:=(m forms a cycle)
      [] m.v=0 → skip
      fi;
      r.v,t.v,m.v,p.v := r.v+1,d.v,p.v,v
[] (u,v)∈E∧r.u-r.v=w(u,v)∧t.v<t.u+d.v → t.v,p.v:=t.u+d.v,p.u
[] (u,v)∈E∧r.u-r.v>w(u,v) →
```

```
      r.v,t.v,m.v,p.v:=r.u-w(u,v),t.u+d.v,u,p.u
□   P(r,t)∧max.t<maxT → maxT:=max.t
od
```

Algorithm in working



Merits of the new algorithm

- Simple: no need for upper or lower bounds, or a list of candidate periods
- Efficient: lazy update of t and r; improvement and possibility check are done at the same time; only last iteration is infeasible
- Incremental: valid retiming is maintained during iterations, may be combined with other optimizations

Experimental results

name	#gates	clock period		time(s)	ASTRA(s)
		before	after		
s1423	490	166	127	0.02	0.04
s1494	558	89	88	0.02	0.01
s9234	2027	89	81	0.12	0.19
s9234.1	2027	89	81	0.16	0.20
s13207	2573	143	82	0.12	0.49
s15850	3448	186	77	0.36	0.57
s35932	12204	109	100	0.28	0.86
s38417	8709	110	56	0.58	1.46
s38584	11448	191	163	0.41	1.12
s38584.1	11448	191	183	0.48	1.26

Lessons Learned

- Optimization objective should be considered as another post-condition

Lessons Learned

- Optimization objective should be considered as another post-condition
- To optimize, or not to optimize: that is the question:
  Whether 'tis nobler to iteratively improve a feasible solution,
  Or to take the risk of trespassing the infeasible region?
  To construct: to improve;
  No more; ...

Let's Look at the *Elements*

- The Greatest Common Divisor problem: Given two integers a and b, find their greatest common divisor.

$$\begin{aligned} &Maximize && x \\ &subject\ to && x|a, x|b \end{aligned}$$

To Optimize, or Not to Optimize

- To optimize, we may start with a divisor of a and b (say 1), and find larger and larger ones
- No to optimize, we may start with a sure upper bound (say $\min(a, b)$), and decrease it while it is not a common divisor

What Euclid Did

Euclid's Algorithm

```
x, y := a, b;
do
   x>y → x:=x-y
□  y>x → y:=y-x
od
```

How He Did It

Euclid's Algorithm

```
{ a>0 ∧ b>0 }
x, y := a, b;
do
{ gcd(x,y)=gcd(a,b) }
   x>y → x:=x-y
   y>x → y:=y-x
od
{ x=gcd(a,b) }
```

He is a Master

- He combined optimization objective and constraints into one condition

$$x = \gcd(a, b)$$

- He decomposed it into

$$x = y \wedge \gcd(x, y) = \gcd(a, b)$$

- He selected an invariant: $\gcd(x, y) = \gcd(a, b)$
- He used a loop goal: $x = y$
- The termination is guaranteed by the monotonic decreasing of $x + y$

To Optimize, or Not to Optimize: That is not a Big Question...

- if you know how to manipulate symbols.

  Computing Science is –and will always be– concerned with the interplay between mechanized and human symbol manipulation, which usually referred to as "computing" and "programming" respectively. (E. W. Dijkstra)