Algebraic Methods

Weak division can be accomplished in $O(n \log n)$ time.

Systematic method exists for generating all the useful divisors, i.e., kernels, of a function.

After generating kernels, still need to select the right divisor(s).

Recap: Divisors and Kernels

Define divisors of f as the set

$$\mathcal{D}(f) = \{\ g \mid f/g \neq \phi\ \}$$

Define primary divisors of f as

$$\mathcal{P}(f) = \{\ f/c \mid c \text{ is a cube } \}$$

g **is termed "cube-free" if the only cube dividing g <u>evenly</u> is 1.**

Define the kernels of f as

$$\mathcal{K}(f) = \{\ k \mid k \in \mathcal{P}(f),\ k \text{ is cube-free } \}$$

Examples

Consider

$$f = acd + bcd + ae + be$$

		DIVISOR TYPE
f/a	$= cd + e$	kernel
f/c	$= ad + bd$	not a kernel
f/cd	$= a + b$	kernel
f/e	$= a + b$	kernel

Common Divisors and Kernels

Goal of multi-level logic optimizer is to find common divisors of two (or more) functions f and g

Theorem: f and g have a non-trivial common divisor d ($d \neq$ cube) if and only if there exist kernels

$k_f \in \mathcal{K}(f)$, $k_g \in \mathcal{K}(g)$ such that

$k_f \cap k_g$ is non-trivial, i.e., not a cube

$\therefore$ can use kernels of f and g to locate common divisors

Algorithm to find All Kernels

Kernels(f)

Find c_f so f / c_f is cube-free ;
$\mathcal{K}$ = Kernel1(0, f / c_f) ;
if (f is cube-free)
    return($f \cup \mathcal{K}$) ;
return($\mathcal{K}$) ;

Kernel1(j, g) {
    $\mathcal{R}$ = g ;
    for ($i = j + 1;\ i \leq n;\ i = i + 1$) {
        if (l_i in one or no terms) continue ;
        c_e = Max. literal cube evenly dividing g / l_i ;
        if (l_k not in c_e, for all $k \leq i$)
            $\mathcal{R}$ = $\mathcal{R} \cup$ Kernel1(i, $(g / l_i) / c_e$)
    }
    return($\mathcal{R}$) ;
}

Kerneling Example

$$f = abcd + abce + adgh + aegh + abde + acdeg + beh$$

<u>**co-kernel**</u>	<u>**kernel**</u>
1	$a(bc + gh)(d + e) + ade(b + cg) + beh$
a	$(bc + gh)(d + e) + de(b + cg)$
ab	$c(d + e) + de$
abc	$d + e$
abd	$c + e$
ac	$b(d + e) + deg$
acd	$b + eg$
ace	$b + dg$
ad	$b(c + e) + g(ce + h)$
ade	$b + cg$
adg	$ce + h$
$\vdots$	

Pruning Condition and Example

If the largest cube factor c_e contains an already selected literal, then terminate current branch

- All kernels found by continuing have already been seen

$$f = abc(d+e)(k+l) + agh + m$$

a

$$f/a = bc(d+e)(k+l) + gh$$

b

$$f/ab = c(d+e)(k+l)$$

$$(d+e)(k+l)$$

$\ldots\ldots$

c

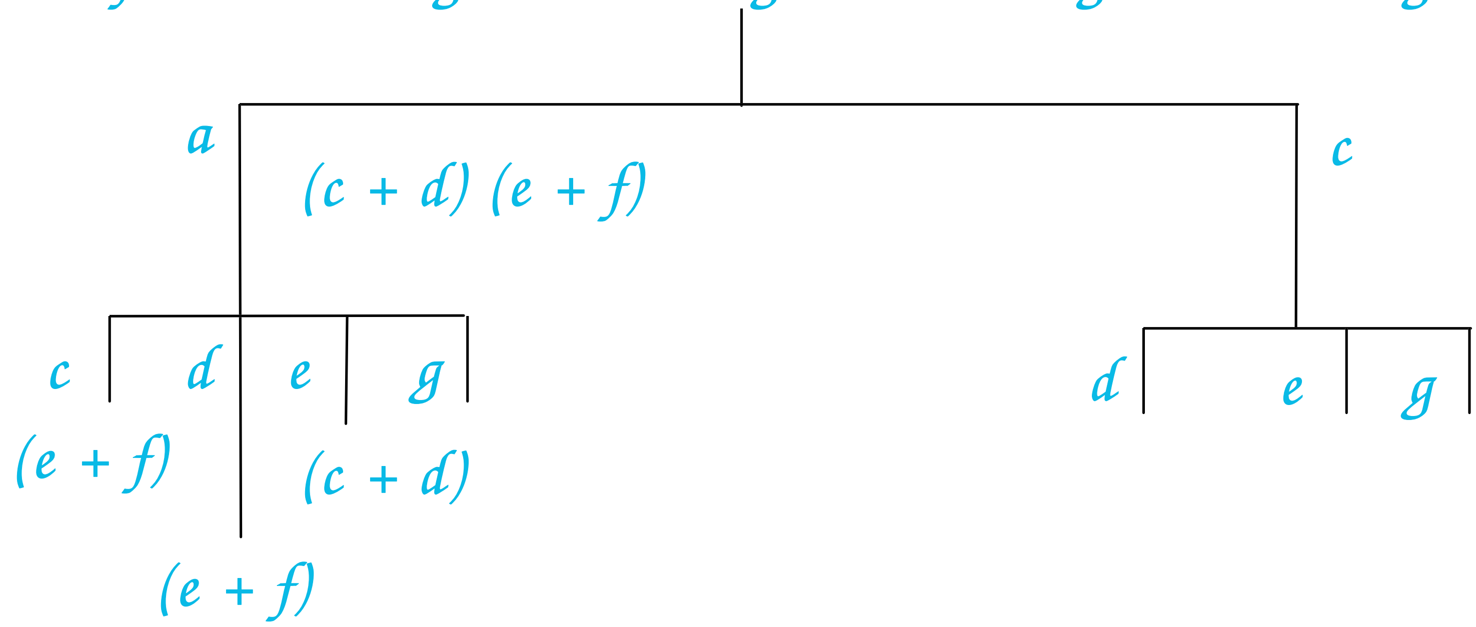
$$f/ac = b(d+e)(k+l)$$

b **already selected**

Number of Kernels

Number of kernels of a logic expression can be very large.

$$f = ace + acg + bce + bcg + ade + adg + bde + bdg$$



Kernels of Depth d

Too many kernels, need to select "useful" ones

It is more efficient, and almost as effective to compute a given subset of $\mathcal{K}(f)$ rather than entire set

$\Rightarrow$ $\mathcal{K}^d(f)$ defined as kernels of depth d

$\mathcal{K}^*(f)$ is all the kernels of f except for f

Then, $\mathcal{K}^0(f) = \{ k \in \mathcal{K}(f) \mid \mathcal{K}^*(k) = \phi \}$
i.e., the lowest level kernels

$\mathcal{K}^{n+1}(f) = \{ k \in \mathcal{K}(f) \mid \mathcal{K}^*(k) \subseteq \mathcal{K}^n(f) \}$
i.e., all its kernels are contained in the next lower level (at least one) or any level below n.

Decomposition Algorithm

DISTILL: Identify expressions in several functions $\{ f_i \}$

(1) Generate kernels for each f_i

(2) <u>Select</u> a pair of kernels

$k_1 \in \mathcal{K}(f_i)$, $k_2 \in \mathcal{K}(f_j)$ $\; i \neq j$ **such that**

$k_1 \cap k_2$ **is not a cube**

If none exist, stop.

(3) Set new variable $v \leftarrow k_1 \cap k_2$

(4) Update associated functions

$f_i = v\,(f_i/(k_1 \cap k_2)) + r$ **for all** i.

Go to (1).

Decomposition Algorithm - 2

CONDENSE: Pull out common cubes consisting of at least 2 literals

(1) Select a pair of cubes
$C_1 \in f_i$, $C_2 \in f_j$, $i \neq j$ **such that**
$C_1 \cap C_2$ **consists of 2 or more literals.**
If none exist, stop.

(2) Set new variable $x \leftarrow C_1 \cap C_2$

(3) Update each function f_i **with new variable** x **wherever possible.**

Decomposition Example

$$f_1 = \mathcal{AB}\,(\,\mathcal{C}\,(\,\mathcal{D} + \mathcal{E}\,) + \mathcal{F} + \mathcal{G}\,) + \mathcal{H}$$
$$f_2 = \mathcal{AI}\,(\,\mathcal{C}\,(\,\mathcal{D} + \mathcal{E}\,) + \mathcal{F} + \mathcal{J}\,) + \mathcal{K}$$

Level-0 kernels selected

$$\mathcal{K}^0(f_1) = \{\,\mathcal{D} + \mathcal{E}\,\}$$
$$\mathcal{K}^0(f_2) = \{\,\mathcal{D} + \mathcal{E}\,\}$$

Set $\mathcal{L} = \mathcal{K}^0(f_1) \cap \mathcal{K}^0(f_2) = \mathcal{D} + \mathcal{E}$

$$f_1 = \mathcal{AB}\,(\,\mathcal{CL} + \mathcal{F} + \mathcal{G}\,) + \mathcal{H}$$
$$f_2 = \mathcal{AI}\,(\,\mathcal{CL} + \mathcal{F} + \mathcal{J}\,) + \mathcal{K}$$

Next, 0-decomposition produces

$$\mathcal{K}^0(f_1) = \{\,\mathcal{CL} + \mathcal{F} + \mathcal{G}\,\}$$
$$\mathcal{K}^0(f_2) = \{\,\mathcal{CL} + \mathcal{F} + \mathcal{J}\,\}$$

Set $\mathcal{M} = \mathcal{K}^0(f_1) \cap \mathcal{K}^0(f_2) = \mathcal{CL} + \mathcal{F}$

Decomposition Example - 2

$$\therefore\ \mathcal{f}_1 = \mathcal{AB}\,(\mathcal{M} + \mathcal{G}) + \mathcal{H}$$
$$\mathcal{f}_2 = \mathcal{AI}\,(\mathcal{M} + \mathcal{J}) + \mathcal{K}$$

f_1 and f_2 have no kernel intersections that are non-trivial.

Apply condensation. $\mathcal{ABM} \in f_1$, $\mathcal{AIM} \in f_2$

Set $\mathcal{N} = \mathcal{ABM} \cap \mathcal{AIM} = \mathcal{AM}$

$$\therefore\ f_1 = \mathcal{B}\,(\mathcal{N} + \mathcal{AG}) + \mathcal{H}$$
$$f_2 = \mathcal{I}\,(\mathcal{N} + \mathcal{AJ}) + \mathcal{K}$$

Note: “Collapsing” $\mathcal{L}$ into $\mathcal{M}$ into $\mathcal{N}$ produces

$$\mathcal{L} = \mathcal{D} + \mathcal{E}$$
$$\mathcal{M} = \mathcal{CL} + \mathcal{F} = \mathcal{C}\,(\mathcal{D} + \mathcal{E}) + \mathcal{F}$$
$$\mathcal{N} = \mathcal{AM} = \mathcal{A}\,(\mathcal{C}(\mathcal{D} + \mathcal{E}) + \mathcal{F})$$

Factorization of Expressions

Generic factoring algorithm:

```
Factor ( f ) {
  if ( number of terms in f is 1 )
        return ( f ) ;
  K ← Choose_Divisor ( f )
  ( h, r ) ← Divide ( f, K )
  f ← Factor ( K ) * Factor ( h ) + Factor ( r ) ;
  return ( f ) ;
}
```

Choose_Divisor (f) typically selects a kernel of f

Kernel-based Factorization

We can produce several different factoring routines by changing Choose_Divisor

Example:

Choose_Divisor

- $\mathcal{LF}(\mathcal{X})$ — **Choose literal that occurs most frequently**
- $\mathcal{XF}(\mathcal{X})$ — **Choose kernel that produces the best factored form**

$$\mathcal{X} = ac + ad + ae + ag + bc + bd + be + bf + ce + cf + df + dg$$

$$\mathcal{LF}(\mathcal{X}) = a(c + d + e + g) + b(c + d + e + f) + c(e + f) + d(f + g)$$

$$\mathcal{XF}(\mathcal{X}) = (c + d + e)(a + b) + f(b + c + d) + g(a + d) + ce$$

Delay Optimization

Being able to meet performance requirements is absolutely essential

Timing Optimization

- **Technology independent**
- **Technology dependent**

Technology independent optimizations include global restructuring of network to minimize critical path lengths

Definitions

The arrival time of a signal s denoted $\mathcal{A}_s$ is the time at which the signal settles to its steady state value

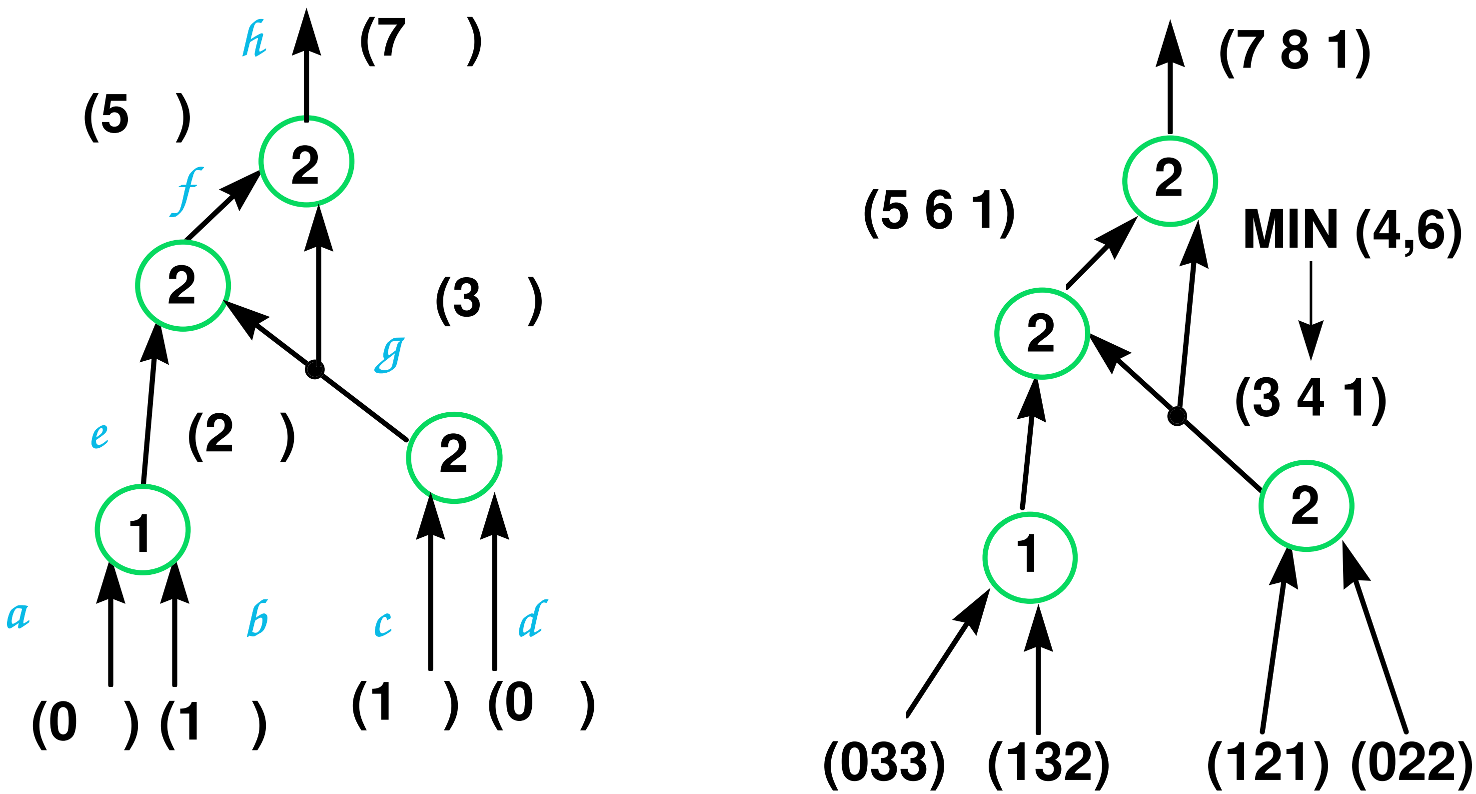
The required time of a signal s denoted $\mathcal{R}_s$ is the time at which the signal is required to be stable

The slack of a signal s

$$\mathcal{S}_s = \mathcal{R}_s - \mathcal{A}_s$$

Delay Tracing

Obtain arrival, required and slack times at each node given arrival times for inputs and required time at output

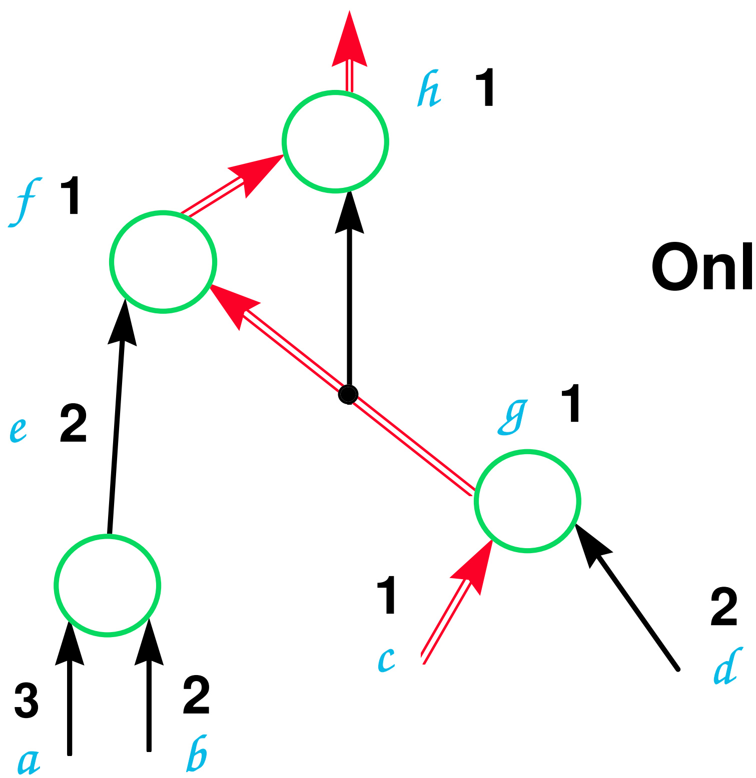


MAX for arrival, MIN for required

Critical Section

The critical section of the Boolean network is the set of nodes with minimum slack

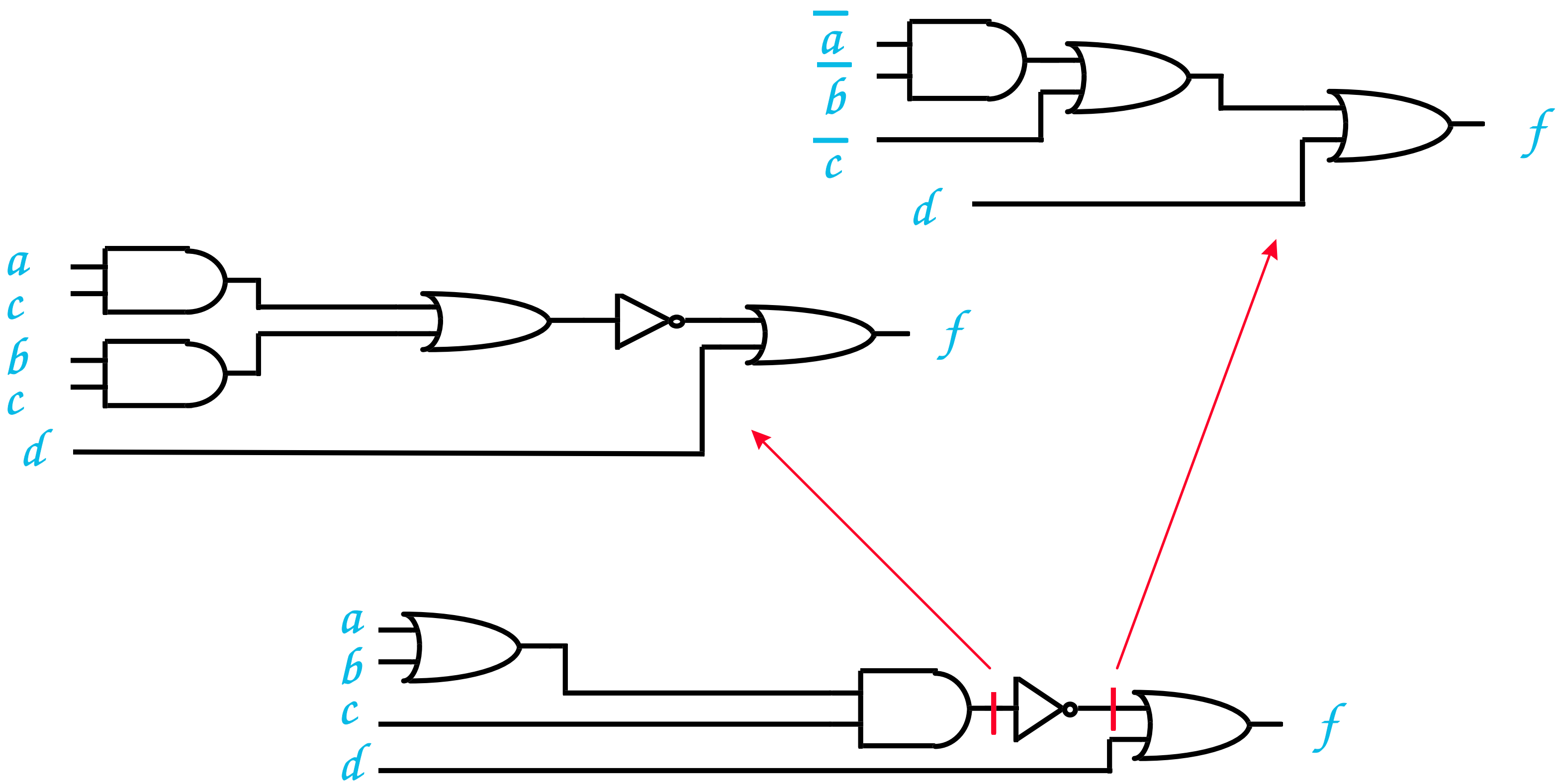
Paths through primary inputs and nodes with minimum slack are critical



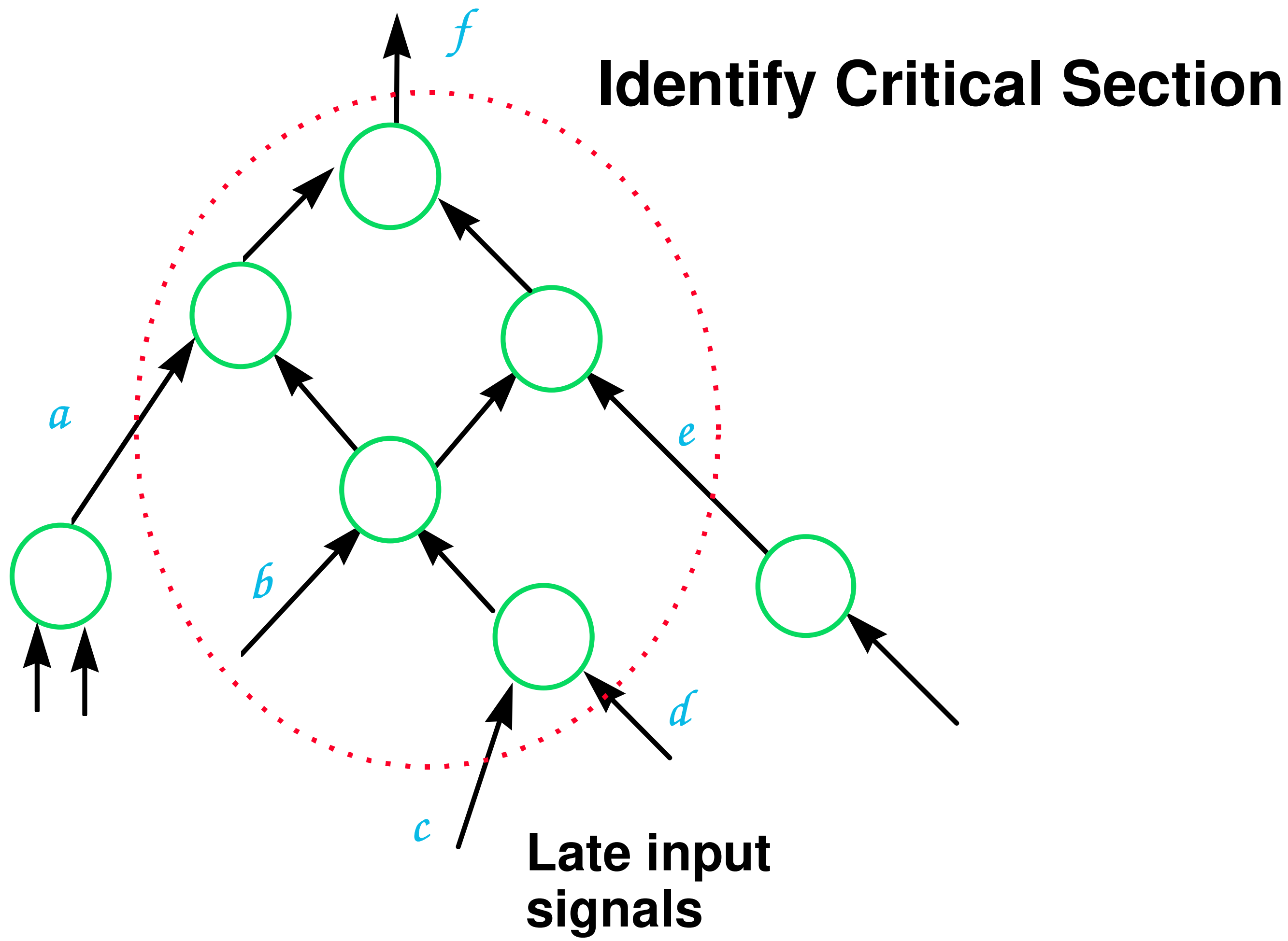
Only slacks are shown

Collapsing

Inverse operation to decomposition

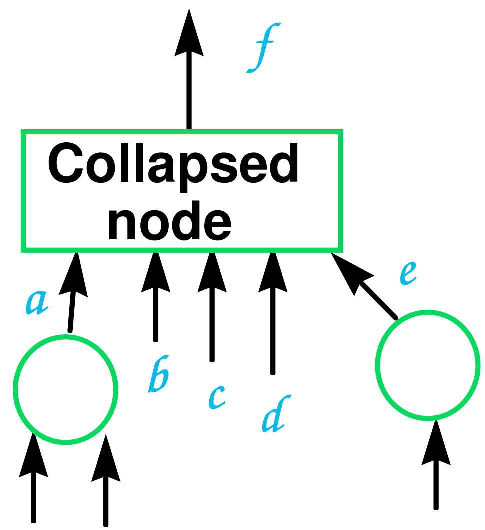


Restructuring to Improve Delay



Restructuring to Improve Delay - 2

Collapse critical section



Re-extract factor k