Lecture 8

Metastability-Containing Sorting

Last week we saw how to obtain an MC implementation of a node's logic for the Lynch-Welch algorithm. However, for this to matter, we need low-depth circuits performing the computations. Otherwise, we would lose the speed advantage gained from forgoing synchronizers, meaning that all that work is for nothing! Hence, our task today is to construct low-depth sorting networks — which, as we have seen, means to construct low-depth comparators.

Before constructing the circuits, we need to fix an encoding. We already decided that we (need) to use a Gray code, but not which one. One of the simplest, if not most natural, Gray codes turns out to be well-suited for our purposes.

Definition 8.1 (Binary Reflected Gray Code)**.** B-bit Binary Reflected Gray Code (BCRG) $G_B \colon [2^B] \to \{0,1\}^B$ *is defined recursively by*

$$\begin{aligned} G_1(0) &= 0 \\ G_1(1) &= 1 \\ \forall B > 1\, \forall x \in [2^{B-1}] \colon G_B(x) &= 0G_{B-1}(x) \\ \forall B > 1\, \forall x \in [2^B] \setminus [2^{B-1}] \colon G_B(x) &= 1G_{B-1}(2^B - 1 - x)\,. \end{aligned}$$

G_B is one-to-one, so we denote by D_B its inverse, the decoding function. In the following, we will write $D(g)$ instead of $D_B(g)$, as B can be inferred from the length of the decoded string g.

We know that we won't have to handle arbitrary metastable strings, as metastability is only introduced by a TDC up-count being interrupted.

Definition 8.2 (Valid Strings)**.** *The set* valid B-bit strings *is defined as*

$$V_B := \{G_B(x) \mid x \in [2^B]\} \cup \{G_B(x) * G_B(x+1) \mid x \in [2^B - 1]\}\,.$$

We define a total order $\leq_G$ on V_B according to the encoded values. The total order is given by the transitive closure of the partial order

$$\begin{aligned} \forall g, h \in V_B \cap \{0,1\}^B \colon g < h &\Leftrightarrow D(g) < D(h) \\ \forall x \in [2^B - 1] \colon G(x) < G(x) * G(x+1) &< G(x+1)\,. \end{aligned}$$

0	0000	4	0110	8	1100	12	1010
0-1	000M	4-5	011M	8-9	110M	12-13	101M
1	0001	5	0111	9	1101	13	1011
1-2	00M1	5-6	01M1	9-10	11M1	13-14	10M1
2	0011	6	0101	10	1111	14	1001
2-3	001M	6-7	010M	10-11	111M	14-15	100M
3	0010	7	0100	11	1110	15	1000
3-4	0M10	7-8	M100	11-12	1M10	---	---

Table 8.1: Valid 4-bit strings.

Denote by $\max_G$ *and* $\min_G$ *the maximum and minimum w.r.t. to* $\leq_G$.

Table 8.1 list V_B according to $\leq_B$. Our goal is to compute $\max_G$ and $\min_G$ for given valid strings $g, h \in V_B$. As you have shown in an exercise, for inputs that are valid strings the above definitions of $\max_G$ and $\min_G$ coincides with the metastable closure of their restrictions to stable values, i.e.,

$$\max_G\{g,h\} = \operatorname*{*}_{\substack{g \preceq g' \in \{0,1\}^B \\ h \preceq h' \in \{0,1\}^B}} \max_G(g',h')\,.$$

Thus, we need to figure out how to implement the closure of these (restricted) operators, at least for inputs that are valid strings.

8.1 4-valued Comparison of BRGC Strings

Our first step is to break down the task of determining $\max_G$ into smaller pieces. One way of doing this is to see how a (simple) state machine can perform the

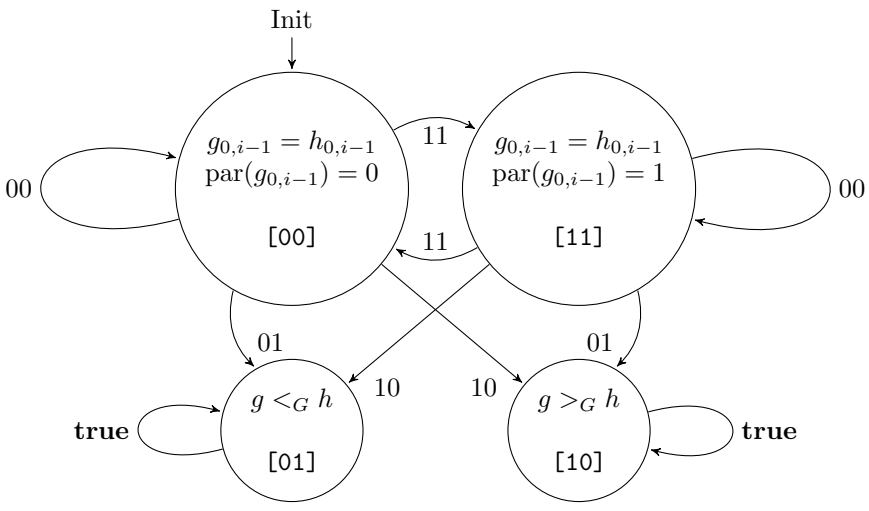

Figure 8.1: Finite state automaton determining which of two Gray code inputs $g, h \in \{0,1\}^B$ is larger. In each step, the machine receives $g_i h_i$ as input. State encoding is given in square brackets.

required computation. Our state machine is fed the input bits one pair at a time, see Figure 8.1, to determine which of the strings (if any) is larger; one then needs to determine the output accordingly. As we are dealing with Gray code, we do not have a 3-valued comparison to make (larger, smaller, or equal, non-trivially recursing only on state equal), but rather a 4-valued one: the possible states are larger, smaller, equal with even parity (standard recursion), and equal with odd parity (recurse with flipped meanings of larger and smaller).

It is straightforward to see that the state machine operates correctly on stable inputs. But what happens for unstable inputs? This is the reason why the state machine also specifies how to encode its states. We want that if at some point the state machine is not yet decided and one of the inputs is M (but the other not), the metastable closure of the state transition function yields a new "state" whose stabilizations correspond to the results of the comparisons if we had stabilized the inputs first. To formalize this, let us first fix some notation.

Definition 8.3 (Transition Operator)**.** *Given state $s \in \{0,1\}^2$ of the state machine in Figure 8.1 and inputs $b \in \{0,1\}^2$, denote by $s \diamond b$ the resulting state of the state machine, i.e.:*

meaning of state	$\diamond$	00	01	11	10
equal, par $= 0$	00	00	01	11	10
$<_G$	01	01	01	01	01
equal, par $= 1$	11	11	10	00	01
$>_G$	10	10	10	10	10

Note that $\diamond$ is associative and $00 \diamond b = b$, so the state of the machine after processing the input completely is $\diamondsuit(g,h) := \diamondsuit_{i=1}^{B} g_i h_i := g_1 h_1 \diamond g_2 h_2 \diamond \ldots \diamond g_B h_B$, where the order in which the $\diamond$ operations are executed is arbitrary.

Lemma 8.4. *For $g, h \in \{0,1\}^B$,*

$$\diamondsuit(g,h) = \begin{cases} 00 & \text{if } g = h \text{ and } \operatorname{par}(g) = 0 \\ 11 & \text{if } g = h \text{ and } \operatorname{par} g = 1 \\ 01 & \text{if } g <_G h \\ 10 & \text{if } g >_B h\,. \end{cases}$$

Proof. We proof this by induction on B. For $B = 1$, we readily see that the state machine transitions to the correct state. For $B > 1$, observe that if the machine is in state 01 or 10 before processing the last pair of bits, by induction hypothesis $g_{1\ldots B-1} \neq h_{1\ldots B-1}$ and the machine has already decided correctly. If $g_{1\ldots B-1} = h_{1\ldots B-1}$, the parity of $g_{1\ldots B-1}$ correctly kept track of whether the remaining (trivial, 1-bit) code is listed in default order (parity 0, current state 00) or reversed (parity 1, current state 11). Checking the state transitions of the machine, we see that the machine correctly which string is larger if $g_B \neq h_B$, and correctly adjusts the parity if $g_B = h_B$. □

The state machine will have to be implemented by some circuit. From Theorem 6.6, we know that we can implement $\diamond_{\mathrm{M}}$, the metastable closure of $\diamond$. Conveniently, this operator turns out to be associative as well.

Lemma 8.5. $\diamond_{\mathrm{M}}$ *is associative.*

Proof. While an elegant proof would be desirable, all we know is how to do this by a case distinction. This is not very practical by hand (3^8 cases!), so it has been checked by machine only. □

This means that we can apply the same notation as for $\diamond$ to $\diamond_{\mathrm{M}}$ with impunity, i.e.,

$$\diamondsuit_{\mathrm{M}}(g,h) \coloneqq \left(\diamondsuit_{\mathrm{M}}\right)_{i=1}^{B} g_i h_i \coloneqq g_1 h_1 \diamond_{\mathrm{M}} g_2 h_2 \diamond_{\mathrm{M}} \ldots \diamond_{\mathrm{M}} g_B h_B \,.$$

In order to show that we can decompose $(\diamondsuit)_{\mathrm{M}}$ into repeatedly applying $\diamond_{\mathrm{M}}$ (by Lemma 8.5 in arbitrary order!), we need the following helper lemma.

Lemma 8.6. *For $g, h \in V_B$ and any $(\diamondsuit)_{\mathrm{M}}(g,h) \preceq s' \in \{0,1\}^2$, there are $g \preceq g' \in \{0,1\}^B$ and $h \preceq h' \in \{0,1\}^B$ such that $s' = \diamondsuit(g',h')$.*

Proof. If $g, h \in \{0,1\}^B$, the statement is trivially true. W.l.o.g., assume that $g \in V_B \setminus \{0,1\}^B$ (if this holds only for h, reason symmetrically). Denote by d the distance of g and h in the total order on V_B (i.e., their distance in the list given in Table 8.1). If $d > 2$, Lemma 8.4 shows that for all stabilizations of g and h, the state machine outputs the same stable value; in this case, again the statement is trivially true.

If $d = 2$ or $d = 1$, checking all possibilities (again, this is easiest using the table) and using Lemma 8.4 we see that the different stabilizations result in outpus (i) 00 and 01, (ii) 01 or 11, (iii) 11 or 10, or (iv) 10 or 00. Either way, the claim of the lemma holds: we have exactly one metastable state bit in the end, and both respective outputs can be also generated by stabilizing the inputs first.

The final case is $d = 0$, i.e., $g = h$. In this case, any output can be generated depending on how we choose to stabilize the unstable bits in g and h, respectively, and $(\diamondsuit)_{\mathrm{M}}(g,h) = \mathrm{MM}$. □

We can now prove the key result that implementing the metastable closure of the statemachine's transition function is sufficient to implement the closure of the comparison.

Theorem 8.7. *Let $g, h \in V_B$. Then, for any $j \in \{1, \ldots, B\}$,*

$$(\diamondsuit)_{\mathrm{M}}(g_{1\ldots j}, h_{1\ldots j}) = \diamondsuit_{\mathrm{M}}(g_{1\ldots j}, h_{1\ldots j}) \,.$$

Proof. The recursive definition of BRGC and valid strings (see Definitions 8.1 and 8.2) ensure that prefixes of valid strings are valid strings, hence it is sufficient to consider the special case that $j = B$. We prove the claim by induction on B, where the base case of $B = 1$ is trivial. For the step from $B - 1 \in \mathbb{N}$ to B, the induction hypothesis yields that

$$s \coloneqq (\diamondsuit)_{\mathrm{M}}(g_{1\ldots B-1}, h_{1\ldots B-1}) = \diamondsuit_{\mathrm{M}}(g_{1\ldots B-1}, h_{1\ldots B-1}) \,.$$

By Lemma 8.6, for any $s \preceq s' \in \{0,1\}^2$, there are $g_{1\ldots B-1} \preceq g'_{1\ldots B-1} \in \{0,1\}^{B-1}$

and $h_{1\dots B-1} \preceq h'_{1\dots B-1} \in \{0,1\}^{B-1}$ so that $s' = \diamondsuit(g_{1\dots B-1}h_{1\dots B-1})$. Thus,

$$\begin{aligned}
(\diamondsuit)_{\mathrm{M}}(g,h) &= \mathop{*}_{\substack{g\preceq g'\in\{0,1\}^B\\ h\preceq h'\in\{0,1\}^B}} \diamondsuit(g'h') \\
&= \mathop{*}_{\substack{g_{1\dots B-1}\preceq g'_{1\dots B-1}\in\{0,1\}^{B-1}\\ h_{1\dots B-1}\preceq h'_{1\dots B-1}\in\{0,1\}^{B-1}\\ g_B\preceq g'_B\in\{0,1\}\\ h_B\preceq h'_B\in\{0,1\}}} (\diamondsuit(g_{1\dots B-1}h_{1\dots B-1})\diamond g_Bh_B) \\
&= \mathop{*}_{\substack{s\preceq s'\in\{0,1\}^2\\ g_B\preceq g'_B\in\{0,1\}\\ h_B\preceq h'_B\in\{0,1\}}} (s'\diamond g'_Bh'_B) \\
&= s\diamond_{\mathrm{M}} g_Bh_B \\
&= \diamondsuit_{\mathrm{M}}(g,h)\,.
\end{aligned}$$

□

Remarks:

- Lemma 8.6 can be generalized to arbitrary operators so long as only a single bit becomes metastable: In this case, we have two stabilizations of the output, meaning there must be two stabilizations of the input yielding different values.
- However, as soon as two output bits become metastable, this simple relation may break down. For instance, already making two copies of the Xor of two bits showcases this issue. Stabilizations of the inputs always result in identical output bits, but e.g. input 1M yields output MM, which may also stabilize to 01 and 10.
- We cannot always reliably decide which of the inputs to our comparator is larger, even if they are not equal. This means that computing the output is not as easy as in the binary world.

8.2 Determining the Output Bits

For stable strings, determining the output would now be straightforward. Compute $s = \diamondsuit(g,h)$, and then, e.g., pick g if $s_1 = 1$ (implying $g \geq_G h$) and h otherwise (implying $g \leq_G h$). By now, you already guess that we cannot simply use a standard MUX for this task, but need to use a $\mathrm{MUX}_{\mathrm{M}}$. Alas, we still run into trouble with this approach.

Example 8.8. *Consider* 1*-bit inputs* $g = \mathrm{M}$ *and* $h = 1$*, i.e.,* $s = \mathrm{M1}$*. Then* $\mathrm{CMUX}(g,h,s_1) = \mathrm{CMUX}(\mathrm{M},1,\mathrm{M}) = \mathrm{M}$*, yet* $\max_G\{g,h\} = h = 1$*.*

One can try around, but the problem persists. The issue is that we cannot reliably decide which value is larger, so the inputs need to "help" with masking metastability. For a single bit, of course all we need to do is to feed the inputs to an Or gate. However, when looking at longer codes, the parity comes into play. So let us see what we get if we combine the state

$$s^{(i-1)} := \diamondsuit_{\mathrm{M}}(g_{1\dots i-1}, h_{1\dots i-1})$$

of the state machine *before* processing the i^{th} bits (where $s^{(0)} := 00$) with the i^{th} bits themselves to determine the i^{th} bit of the output. Taking into account the meaning of the state bits, for stable inputs this results in the following mapping $\mathrm{out}\colon (s, g_i h_i) \mapsto \max_G\{g,h\}_i \min_G\{g,h\}_i$.

meaning of state	$s^{(i-1)}$	$\max_G\{g,h\}_i$	$\min_G\{g,h\}_i$
equal, par $= 0$	00	$\max\{g_i, h_i\}$	$\min\{g_i, h_i\}$
$<_G$	10	g_i	h_i
equal, par $= 1$	11	$\min\{g_i, h_i\}$	$\max\{g_i, h_i\}$
$>_G$	01	h_i	g_i

meaning of state	out	00	01	11	10
equal, par $= 0$	00	00	10	11	10
$<_G$	01	00	10	11	01
equal, par $= 1$	11	00	01	11	01
$>_G$	10	00	01	11	10

Note that out has 4 input bits, so the circuit implementing out_M guaranteed by Theorem 6.6 has constant size. However, this is useful only if indeed $\mathrm{out}_\mathrm{M}(s^{(i-1)}, g_i h_i) = \max_G\{g,h\}_i \min_G\{g,h\}_i$ all $g, h \in V_B$ and $i \in \{1, \ldots, B\}$. Proving this is simplified by the following observation on the structure of valid strings.

Observation 8.9. *If for a valid string $g \in \{0,1\}^B$ it holds that $g_i = \mathrm{M}$ for some $i < B$, then $g_{i+1\ldots B} = 10^{B-i-1}$, i.e., $g_{i+1\ldots B}$ is the codeword for the largest value that $(B-i)$-bit Gray code can encode.*

Theorem 8.10. *Given valid inputs $g, h \in V_B$, for all $i \in \{1, \ldots, B\}$ it holds that $\mathrm{out}_\mathrm{M}(s^{(i-1)}, g_i h_i) = \max_G\{g,h\}_i \min_G\{g,h\}_i$.*

Proof. Observe that $\mathrm{out}_\mathrm{M}(s^{(i-1)}, g_i h_i)$ does not depend on bits $i+1, \ldots, B$. As $g_{1\ldots i}, h_{1\ldots i}$ are valid i-bit strings, we may thus w.l.o.g. assume that $B = i$. For symmetry reasons, it suffices to show the claim for the first output bit $\mathrm{out}_\mathrm{M}(s_\mathrm{M}^{(B-1)}, g_B h_B)_1$ only; the other cases are analogous.

Using Lemma 8.4, Definition 8.1, and the definition of out, it is straightforward to verify that the claim holds for stable $g, h \in \{0,1\}^B$. Our task is to prove this equality also for the case where g or h contain a metastable bit. By Theorem 8.7, we have that $s^{(B-1)} = (\diamondsuit)_\mathrm{M}(g_{1\ldots B-1}, h_{1\ldots B-1})$.

Let j be the minimum index such that $g_j = \mathrm{M}$ or $h_j = \mathrm{M}$. Again, for symmetry reasons, we may assume w.l.o.g. that $g_j = \mathrm{M}$; the case $h_j = \mathrm{M}$ is symmetric. If $g_{1\ldots j-1} \neq h_{1\ldots j-1}$, applying Lemma 8.4 shows that either (i) $s^{(i-1)} = 01$ ($g <_G h$) or (ii) $s^{(i-1)} = 10$ ($g >_G h$). Assume (i); (ii) is treated analogously. As the state 01 is absorbing, it follows that $s^{(B-1)} = 01$, regardless of the further bits of g and h. As $\mathrm{out}(01, g_B h_B)_1 = h_B$ for all $g_B h_B \in \{0,1\}^2$, we conclude that $\mathrm{out}_\mathrm{M}(s^{(B-1)}, g_B h_B)_1 = h_B$, as desired.

Hence, suppose that $g_{1\ldots j-1} = h_{1\ldots j-1}$ for the remainder of the proof. We consider the case that $\mathrm{par}(g_{1\ldots j-1}) = 0$ first, i.e., $s^{(j-1)} = 00 = s^{(0)}$. By Definitions 8.1 and 8.2, $g_{j\ldots B}, h_{j\ldots B} \in V_{B-j+1}$, so we may w.l.o.g. assume $j = 1$ in the following. If $B = 1$,

$$\mathrm{out}_\mathrm{M}(s_\mathrm{M}^{(B-1)}, g_B h_B)_1 = \mathrm{out}_\mathrm{M}(00, \mathrm{M} h_B)_1 = \begin{cases} 1 & \text{if } h_1 = 1 \\ \mathrm{M} & \text{otherwise,} \end{cases}$$

which equals $\max_G\{g,h\}_B$ (we simply have a 1-bit code). If $B > 1$, Observation 8.9 yields that $g_{2\dots B} = 10\dots0$. We distinguish several cases.

$h_1 = \mathrm{M}$: Then also $h_{2\dots B} = 10\dots0$. Therefore $g_B = h_B$, $\mathrm{out}(s, g_B h_B)_1 = g_B = h_B$ for any $s \in \{0,1\}^2$, and

$$\mathrm{out}_\mathrm{M}(s_\mathrm{M}^{(B-1)}, g_B h_B)_1 = g_B = h_B = \max_G\{g,h\}_B\,.$$

$h_1 = 1$ and $B = 2$: Thus, $g <_G h$, i.e., we need to output $h_B = \max_G\{g,h\}_B$. Consider the two stabilizations of g, i.e., 01 and 11. If the first bit of g is resolved to 0, we would end up with $s^{(B-1)} = s^{(1)} = 01$, regardless of further bits. If it is resolved to 1, then $s^{(1)} = 11$. Thus,

$$\begin{aligned}\mathrm{out}_\mathrm{M}(s^{(B-1)}, g_B h_B)_1 &= \mathrm{out}_\mathrm{M}(01, 1h_B)_1 * \mathrm{out}_\mathrm{M}(11, 1h_B)_1\\ &= \mathop{*}_{h_B \preceq h'_B \in \{0,1\}} \{h'_B, \min\{1, h'_B\}\}\\ &= \mathop{*}_{h_B \preceq h'_B \in \{0,1\}} \{h'_B\} = h_B\,.\end{aligned}$$

$h_1 = 1$ and $B > 2$: Again, $h_B = \max_G\{g,h\}_B$. Consider the two stabilizations of g, i.e., $010\dots0$ and $110\dots0$. If the first bit of g is stabilized to 0, we end up with $s^{(B-1)} = s^{(1)} = 01$, as 01 is an absorbing state. If it is stabilized to 1, then $s^{(1)} = 11$. As $g_{2\dots B} = 10\dots0$, for any $h \preceq h' \in \{0,1\}^B$, the state machine will end up in either state 00 (if $h'_{2\dots B} = 10\dots0$) or state 01. Overall, we get that (i) $s^{(B-1)} = 01$, (ii) $s^{(B-1)} = 00 * 01 = 0\mathrm{M}$ and $h_{2\dots B} = 1\dots0$, or (iii) $s^{(B-1)} = 0\mathrm{M}$ and $h_{2\dots B} = 1\dots0\mathrm{M}$ (cf. Table 8.1). If (i) applies, $\mathrm{out}(s_\mathrm{M}^{(B-1)}, g_B h_B)_1 = h_B$. If (ii) applies, $\mathrm{out}_\mathrm{M}(s^{(B-1)}, g_B h_B)_1 = g_B = h_B$. If (iii) applies, then

$$\begin{aligned}\mathrm{out}_\mathrm{M}(s^{(B-1)}, g_B h_B)_1 &= \mathrm{out}_\mathrm{M}(00, 0\mathrm{M})_1 * \mathrm{out}_\mathrm{M}(01, 0\mathrm{M})_1\\ &= 0 * 1 * 0 * 1 = \mathrm{M} = h_B\,.\end{aligned}$$

$h_1 = 0$: This case is symmetric to the previous two: depending on how g is resolved, we end up with $s^{(1)} = 10$ or $s^{(1)} = 00$, and need to output g_B. Reasoning analogously, we see that indeed $\mathrm{out}_\mathrm{M}(s^{(B-1)}, g_B h_B)_1 = g_B$.

It remains to consider $\mathrm{par}(g_{1,\dots,j-1}) = 1$. Then $s^{(j-1)} = 11$. Noting that this reverses the roles of max and min, we reason analogously to the case of $\mathrm{par}(g_{1,\dots,j-1}) = 0$. □

Remarks:

- We have decomposed the task of computing the output into computing $s^{(i)}$, $i \in [B]$, and applying out_M.
- We have decomposed computing $s^{(i)}$ into applying $\diamond_\mathrm{M}$ $i-1$ times.
- As out_M and $\diamond_\mathrm{M}$ can be implemented by constant-sized circuits, we get a circuit of asymptotically optimal size $\mathcal{O}(B)$ computing $\max_G\{g,h\}$ and $\min_G\{g,h\}$.
- However, our main goal was to find a circuit of low *depth* performing this computation. Applying $\diamond_\mathrm{M}$ would yield a circuit of depth $\Omega(B)$!
- This is where we shamelessly exploit the associativity of $\diamond_\mathrm{M}$.

8.3 Parallel Prefix Computation

As $\diamond_{\mathrm{M}}$ is associative, $s^{(B-1)}$ is computed by *any* binary tree for which the leaves are the inputs $g_i h_i$, $i \in \{1, \ldots, B-1\}$, and whose inner nodes are $\diamond_{\mathrm{M}}$ (sub)circuits. Using a balanced tree then results in depth $\lceil \log(B-1) \rceil$. However, we need to compute *all* $s^{(i)}$, $i \in [B]$. We could simply use B trees, whose total number of inner nodes would be

$$\sum_{i=0}^{B-1} i = \frac{(B-1)B}{2} \in \Theta(B^2)\,,$$

still resulting in a circuit of the same depth. We can do much better!

Theorem 8.11. *Given a circuit C implementing an associative operator $\odot\colon D \times D \to D$ and inputs $g_i \in D$, $i \in [2^b]$ for some $b \in \mathbb{N}$, there is a circuit of size $\mathcal{O}(2^b|C|)$ and depth $\mathcal{O}(bd(C))$ outputting for each $i \in [2^b]\setminus\{0\}$ the value $\bigodot_{j=0}^{i} g_i$ (where $\bigodot_0^0 g_i = g_0$).*

Proof. Our circuit will have two stages. The first stage (see Figure 8.2) is a balanced binary tree whose leaves are the 2^b inputs and whose non-leaf nodes are copies of C. Each node receives as input the outputs of its two children. Enumerating the leaves in DFS order, leaf $i \in [2^b]$ "outputs" its assigned input value g_i. By induction on decreasing depth in the tree, we get that each node outputs $(\bigodot)_{j=i_{\min}}^{i_{\max}} g_i$, where $i_{\min}$ and $i_{\max}$ are the smallest and largest leaf in the subtree of the node, respectively.

The second stage (see Figure 8.3) of the circuit receives all the computed values as input and computes all $\bigodot_{j=0}^{i} g_i$, $i \in [2^b]$, using a recursive scheme. We can describe the recursion again as a binary tree, with a one-to-one correspondence of nodes to the ones from the first stage. However, now outputs flow from non-leaf nodes to their children as inputs. For notational convenience, introduce the special symbol $\epsilon \notin D$ with the semantics $\epsilon \odot s = s$ for all $s \in D \cup \epsilon$, i.e., ϵ means "do nothing" (clearly, the extended operator remains associative). Moreover, denote for each non-leaf node by its left child the one traversed first in the DFS tour and refer to the other as right child. Provide to each node two inputs: the output o of the "left" (see figure) child node in the first stage and the output p of its parent, where the root receives ϵ as second input. With this notation, each non-leaf node now outputs p to its left child and $p \odot o$ to its right child. Finally, the leaf i outputs $p \odot g_i$.

We claim that i outputs $\bigodot_{j=0}^{i} g_i$. We prove the claim by induction on the depth of the tree. It is trivial for a tree of depth 0, hence assume it is correct for depth $d \in \mathbb{N}_0$ and consider a tree of depth $d+1$. Applying the induction hypothesis to the left child of the root, we see that leaf $i \in [2^{b-1}]$ outputs $\epsilon \odot \bigodot_{j=0}^{i} g_i = \bigodot_{j=0}^{i} g_i$; note that we exploited that (the extended) $\odot$ is associative here. Applying the induction hypothesis to the right child, we see that leaf $i \in [2^b] \setminus [2^{b-1}]$ outputs $(\bigodot_{j=0}^{2^{b-1}-1} g_i) \odot (\bigodot_{j=2^{b-1}}^{i} g_i) = \bigodot_{j=0}^{i} g_i$.

Apart from wires, our construction has at each non-leaf node of each of the two trees one copy of C, plus a copy of C at each leaf in the second tree, for a total of $3 \cdot 2^b - 2 \in \mathcal{O}(2^b)$ copies of $|C|$. The depth of both trees is b. The claims on size and depth of the circuit follow. □

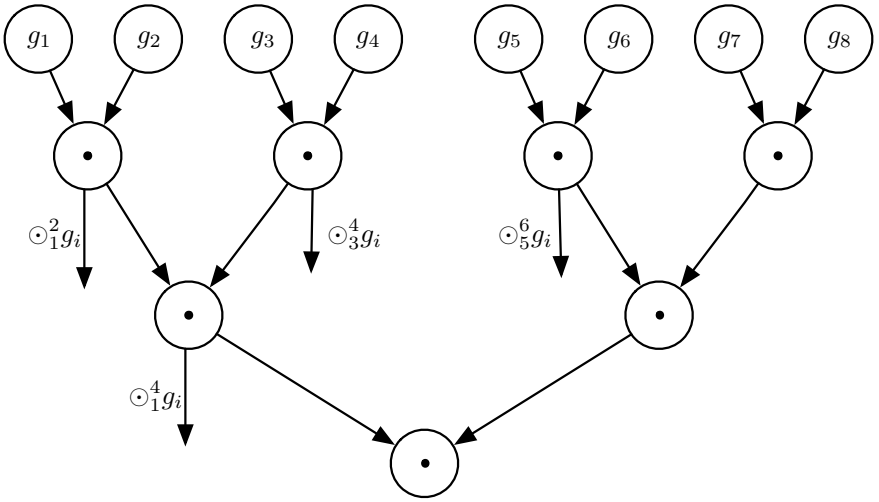

Figure 8.2: First stage of the construction in Theorem 8.11. Each tree node outputs the result of applying the operator to all leaves in its subtree. The output of the root and nodes reached from it by only "going to the right" are not needed; the nodes are there to show the tree structure.

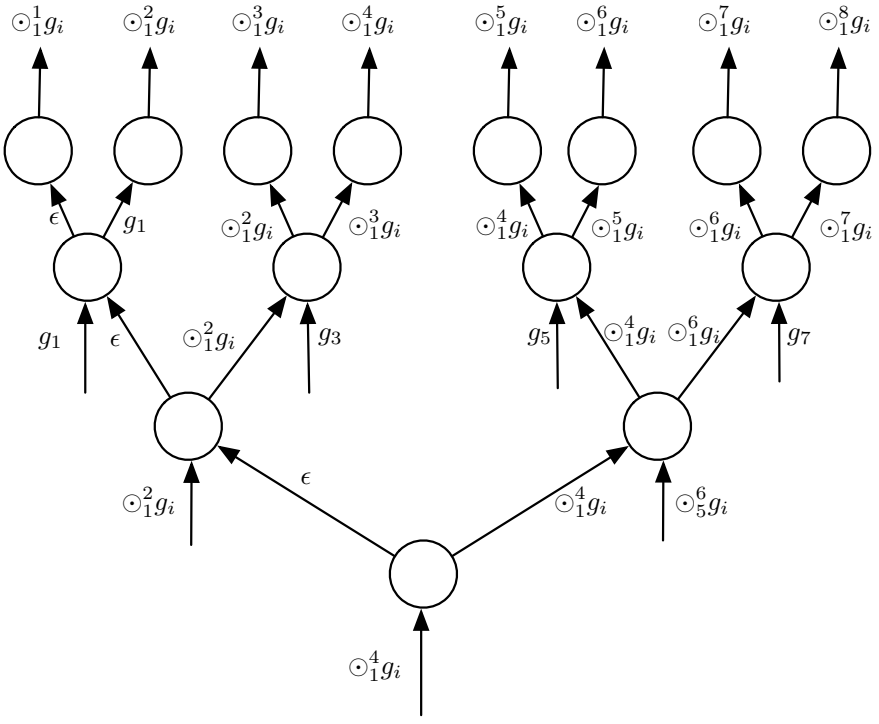

Figure 8.3: Second stage of the construction in Theorem 8.11. Using the outputs of the first stage, the nodes forward their input to left and the operator applied to the input from their parent and the one from the previous stage.

Corollary 8.12. *There is a comparator circuit of size $\mathcal{O}(B)$ and depth $\mathcal{O}(\log B)$ for valid strings (see Figure 8.4).*

Proof. By Theorem 6.6, there are circuits of constant size and depth that implement $\diamond_{\mathrm{M}}$ and $\mathrm{out}_{\mathrm{M}}$. We apply Theorem 8.11 to the circuit for $\diamond_{\mathrm{M}}$ and inputs $g_i h_i$, $i \in \{1, \ldots, B-1\}$ (for $b = \lceil \log B \rceil$, simply ignoring the unneeded inputs and outputs to the circuit), yielding a circuit of size $\mathcal{O}(B)$ and depth $\mathcal{O}(\log B)$ computing outputs $s^{(i-1)}$, $i \in [B] \setminus \{0\}$. As $s^{(0)} = 00$ is a constant, we do not need a circuit to compute it. We then feed for $i \in \{1, \ldots, B\}$ the inputs $s^{(i-1)}$ and $g_i h_i$ to a copy of the circuit implementing $\mathrm{out}_{\mathrm{M}}$, yielding the correct outputs. This adds $\mathcal{O}(B)$ to the size and increases the depth by a constant. □

Bibliographic Notes

There's almost nothing to add to the references given for the previous lecture. For the Parallel Prefix Computation (PPC) framework, see [LF80].

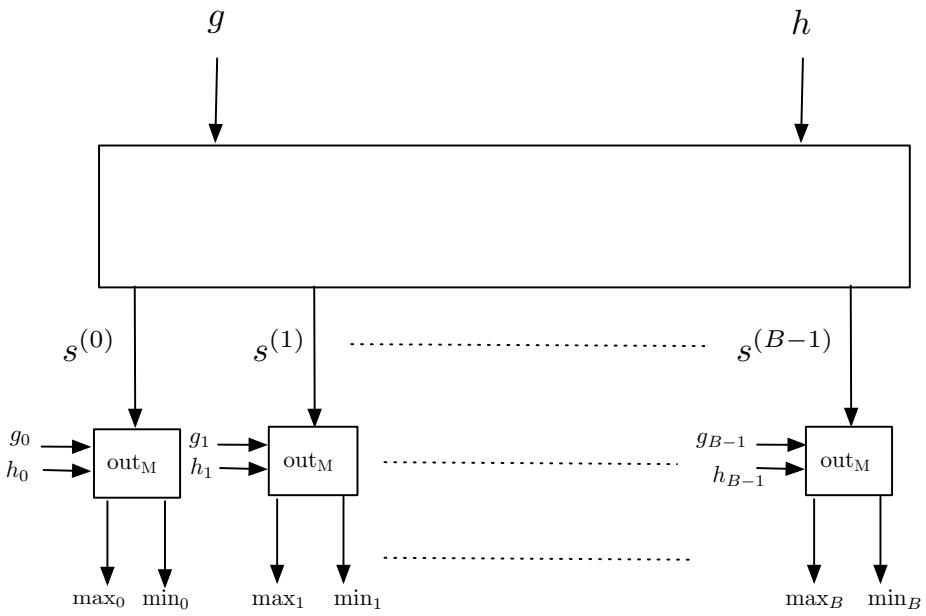

Figure 8.4: The gray code comparator.

Bibliography

[LF80] Richard E Ladner and Michael J Fischer. Parallel Prefix Computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.