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 was 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*

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

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

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

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 lists 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.,

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

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

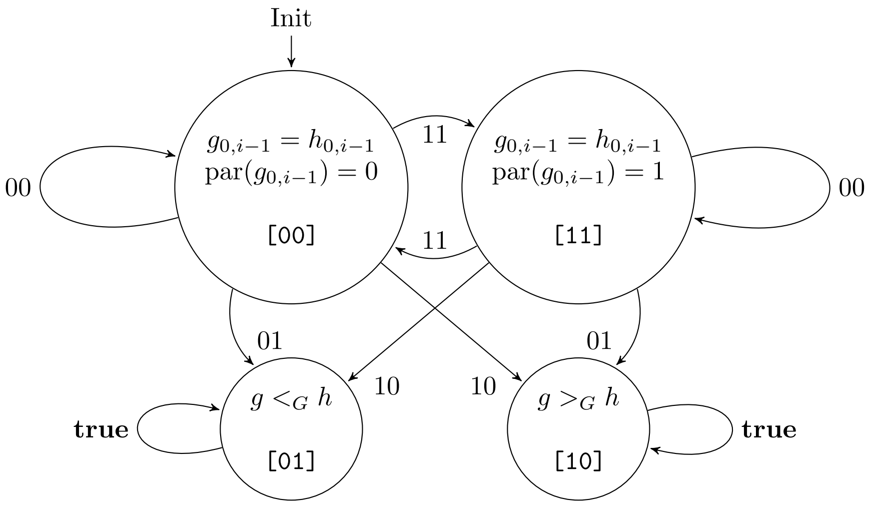

Figure 8.1: Finite state automaton determining which of two Gray code inputs $g, h \in \{0,1\}^B$ is larger. In each step, the machine receives $g_i h_i$ as input. State encoding is given in square brackets.

$s^{(i-1)}$	$\max_G\{g,h\}_i$	$\min_G\{g,h\}_i$
`00`	$\max\{g_i,h_i\}$	$\min\{g_i,h_i\}$
`10`	g_i	h_i
`11`	$\min\{g_i,h_i\}$	$\max\{g_i,h_i\}$
`01`	h_i	g_i

Table 8.2: Computing $\max_G\{g,h\}_i$ and $\min_G\{g,h\}_i$ from the current state $s^{(i-1)}$ and inputs g_i and h_i.

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).

Because the parity keeps track of whether the remaining bits are to be compared w.r.t. the standard or “reflected” order, the state machine performs the comparison correctly w.r.t. the meaning of the states indicated in Figure 8.1.

Lemma 8.3. *Let $g,h \in \{0,1\}^B$ and $i \in [B+1]$. Then*

- $s^{(i)} = 00 \Leftrightarrow (g_{1,i} = h_{1,i}$ *and* $(g <_G h \Leftrightarrow g_{i+1,B} <_G h_{i+1,B}))$,
- $s^{(i)} = 11 \Leftrightarrow (g_{1,i} = h_{1,i}$ *and* $(g <_G h \Leftrightarrow g_{i+1,B} >_G h_{i+1,B}))$,
- $s^{(i)} = 01 \Leftrightarrow g <_G h$, *and*
- $s^{(i)} = 10 \Leftrightarrow g >_G h$.

Proof. We show the claim by induction on i. It holds for $i = 0$, as $s^{(i)} = 00$, $g_{1,0} = h_{1,0}$ is the empty string, and $g <_G h$ if and only if $g_{1,B} = g <_G h = h_{1,B}$. For the step from $i-1 \in [B]$ to i, we make a case distinction based on $s^{(i-1)}$.

$s^{(i-1)} = 00$: By the induction hypothesis, $g_{1,i-1} = h_{1,i-1}$ and $g <_G h$ if and only if $g_{i,B} <_G h_{i,B}$. Thus, if $g_ih_i = 00$, $s^{(i)} = 00$, $g_{1,i} = h_{1,i}$, and by the recursive definition of the code, $g_{i,B} <_G h_{i,B} \Leftrightarrow g_{i+1,B} <_G h_{i+1,B}$. Similarly, if $g_ih_i = 11$, also $g_{1,i} = h_{1,i}$, but the code for the remaining bits is “reflected,” i.e., $g <_G h \Leftrightarrow g_{i+1,B} >_G h_{i+1,B}$. If $g_ih_i = 01$, the definition implies that $g <_G h$ regardless of further bits, and if $g_ih_i = 10$, $g >_G h$ regardless of further bits.

$s^{(i-1)} = 11$: Analogously to the previous case, noting that reflecting a second time results in the original order.

$s^{(i-1)} = 01$: By the induction hypothesis, $g <_G h$. As 01 is an absorbing state, also $s^{(i)} = 01$.

$s^{(i-1)} = 10$: By the induction hypothesis, $g >_G h$. As 10 is an absorbing state, also $s^{(i)} = 10$. □

This lemma gives rise to a sequential implementation based on the given state machine, for input strings in $\{0,1\}^B$. Table 8.2 lists the i^{th} output bit as function of $s^{(i-1)}$ and the pair $g_i h_i$. Correctness of this computation follows immediately from Lemma 8.3. Evaluating the table for the possible inputs $g_i h_i$ yields the truth table of the function returning $\max_G\{g,h\}_i \min_G\{g,h\}_i$ when given arguments $s^{(i-1)}$ and $g_i h_i$.

Definition 8.4 (Output Function). *Given state $s^{(i-1)} \in \{0,1\}^2$ of the state machine in Figure 8.1 after $i-1 \in \mathbb{N}_0$ steps and inputs $g_i h_i \in \{0,1\}^2$, denote by* $\operatorname{out}(s^{(i-1)}, g_i h_i)$ *the output bits at position i, i.e.:*

out	00	01	11	10
00	00	10	11	10
01	00	10	11	01
11	00	01	11	01
10	00	01	11	10

We can express the transition function of the state machine as an (as easily verified) associative operator $\diamond$ taking the current state and input $g_i h_i$ as argument and returning the new state.

Definition 8.5 (Transition Operator). *Given state $s^{(i-1)} \in \{0,1\}^2$ of the state machine in Figure 8.1 and inputs $g_i h_i \in \{0,1\}^2$, define $\diamond$ such that $s^{(i)} = s^{(i-1)} \diamond g_i h_i$ is 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 $00 \diamond g_i h_i = g_i h_i$, 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.

Noting that the initial state $s^{(0)} = 00$ and that $s^{(0)} \diamond x = 00 \diamond x = x$ for all $x \in \{0,1\}^2$, we arrive at the following corollary.

Corollary 8.6. *For all $i \in [1,B]$, we have that*

$$\max_G\{g,h\}_i \min_G\{g,h\}_i = \operatorname{out}\left(\diamondsuit_{j=1}^{i-1} g_j h_j, g_i h_i\right).$$

Remarks:

- This is all great, but it does not address the question of how to handle metastability.
- Without metastability, all one needs to do is compute $s^{(B)}$ and then apply the output function. As $\diamond$ is associative, a binary tree of $\diamond$ subcircuits can do this efficiently.

i	0	1	2	3	4
$g_i h_i$		00	M0	11	00
$s_{\mathtt{M}}^{(i)} = s_{\mathtt{M}}^{(i-1)} \diamond_{\mathtt{M}} g_i h_i$	00	00	M0	1M	1M
$\mathrm{out}_{\mathtt{M}}(s_{\mathtt{M}}^{(4)}, g_i h_i)$		00	MM	11	00
$\mathrm{out}_{\mathtt{M}}(s_{\mathtt{M}}^{(i-1)}, g_i h_i)$		00	M0	11	00

Table 8.3: Run of the FSM on inputs $g = 0\mathrm{M}10$ and $h = 0010$, showing that computing only the last state is insufficient. This yields $\mathrm{out}_{\mathrm{M}}(1\mathrm{M}, \mathrm{M}0) = *\{00, 01, 10\} = \mathrm{MM}$ as second output, but $\mathrm{out}_{\mathrm{M}}(00, \mathrm{M}0) = *\{00, 10\} = \mathrm{M}0$ is correct.

- Our strategy will be to replace $\diamond$ and out by their closure, i.e., $\diamond_{\mathrm{M}}$ and $\mathrm{out}_{\mathrm{M}}$. It turns out that this approach works, but requires to actually use $s^{(i-1)}$ for each $i \in [1, B]$ instead of $s^{(B)}$. Table 8.3 illustrates this by an example.

8.2 Dealing with Metastable Inputs

As stated above, our strategy is to replace the operators by their metastable closure. Moreover, we will exploit associativity of $\diamond_{\mathrm{M}}$ to efficiently compute $g_1 h_1 \diamond_{\mathrm{M}} g_2 h_2 \diamond_{\mathrm{M}} \ldots \diamond_{\mathrm{M}} g_{i-1} h_{i-1}$ for all $i \in [1, B]$ simultaneously, i.e., using a circuit of small depth. Before we can do so, there are three hurdles to overcome:

(P1) Show that $\diamond_{\mathrm{M}}$ is associative.

(P2) Show that repeated application of $\diamond_{\mathrm{M}}$ computes $s_{\mathrm{M}}^{(i)}$.

(P3) Show that applying $\mathrm{out}_{\mathrm{M}}$ to $s_{\mathrm{M}}^{(i-1)}$ and $g_i h_i$ results for all valid strings in $(\max_G)_{\mathrm{M}}\{g, h\}_i (\min_G)_{\mathrm{M}}\{g, h\}_i$.

Regarding the first point, we note the statement that $\diamond_{\mathrm{M}}$ is associative does not depend on B. In other words, it can be verified by checking for all possible $x, y, z \in \{0, 1, \mathrm{M}\}^2$ whether $(x \diamond_{\mathrm{M}} y) \diamond_{\mathrm{M}} z = x \diamond_{\mathrm{M}} (y \diamond_{\mathrm{M}} z)$. While it is tractable to manually verify all $3^6 = 729$ cases (exploiting various symmetries and other properties of the operator), it is tedious and prone to errors. Instead, we verified that both evaluation orders result in the same outcome by a short computer program.

Theorem 8.7. *(P1) holds, i.e., $\diamond_{\mathrm{M}}$ is associative.*

Apart from being essential for our construction, this theorem simplifies notation; in the following, we can apply the same notation as for $\diamond$ to $\diamond_{\mathrm{M}}$ with impunity, i.e.,

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

Remarks:

- In general, the closure of an associative operator needs not be associative.

8.3 Determining $s_{\mathbf{M}}^{(i)}$

Our reasoning will be based on distinguishing two main cases: one is that $s_{\mathrm{M}}^{(i)}$ contains at most one metastable bit, i.e., $|\operatorname{res}(s_{\mathrm{M}}^{(i)})| \leq 2$, and the other that $s_{\mathrm{M}}^{(i)} = \mathrm{MM}$. For each we need a technical statement.

FIXME: I'm not sure whether the resolution should already be introduced when defining the closure.

For the first statement, a careful look at the metastable closure will be the charm. The following notation will be useful.

Definition 8.8. *For $x \in \{0,1,\mathrm{M}\}^B$, define the resolution $\operatorname{res}(x) : \{0,1,\mathrm{M}\}^B \to \mathcal{P}\left(\{0,1\}^B\right)$ as follows:*

$$\operatorname{res}(x) := \{y \in \{0,1\}^B \,|\, \forall i \in \{1,\ldots,B\} \colon x_i \neq \mathrm{M} \Rightarrow y_i = x_i\}\,.$$

Note that with this definition, we have that $f_{\mathrm{M}}(x) = *\, f(\operatorname{res}(x))$, where for any function f and set S, $f(S) = \{f(s) \,|\, s \in S\}$.

Observation 8.9. *For any $x \in \{0,1,\mathrm{M}\}^B$, $*\operatorname{res}(x) = x$.*

Proof. By Definition 8.8,

$$\forall i \in \{1,\ldots,B\} \colon \operatorname{res}(x_i) = \begin{cases} \{x_i\} & \text{if } x_i \neq \mathrm{M} \\ \{0,1\} & \text{if } x_i = \mathrm{M}. \end{cases}$$

By Definition 7.9, thus

$$\forall i \in \{1,\ldots,B\} : (*\operatorname{res}(x))_i = x_i\,. \qquad \square$$

For example: $*\operatorname{res}(0\mathrm{M}10) = *\{0010, 0110\} = 0\mathrm{M}10\,.$

Observation 8.10. *For $\emptyset \neq S \subseteq \{0,1\}^B$, we have $S \subseteq \operatorname{res}(*\,S)$.*

Proof. Let $\emptyset \neq S \subseteq \{0,1\}^B$ and $s \in S$. From Definition 7.9, for all $i \in [1,B]$ we have that either $(*\,S)_i = s_i$ or $(*\,S) = \mathrm{M}$. By Definition 8.8, it follows that $s \in *\,S$. $\square$

In general, the reverse direction does not hold, i.e., $\operatorname{res}(*\,S) \not\subseteq S$. For example, consider $S = \{01, 10\}$ and thus $*\,S = \mathrm{MM}$ such that $\operatorname{res}(*\,S) = \{00, 01, 10, 11\} = \{0,1\}^2$. Hence, $S \subseteq \operatorname{res}(*\,S)$ but not $\operatorname{res}(*\,S) \subseteq S$. In contrast, for $|\operatorname{res}(*\,S)| \leq 2$, we can see that the reverse direction holds.

Observation 8.11. *For any subset of strings $\emptyset \neq S \subseteq \{0,1\}^B$, if $|\operatorname{res}(*\,S)| \leq 2$, then $\operatorname{res}(*\,S) = S$.*

Proof. By Observation 8.10, $S \subseteq \operatorname{res}(*\,S)$, so $0 \neq |S| \leq 2$. If $|S| = 1$, i.e., $S = \{s\}$ for some $s \in \{0,1\}^B$, then $\operatorname{res}(*\,S) = \operatorname{res}(s) = \{s\} = S$. If $|S| = 2$, we have that $S \subseteq \operatorname{res}(*\,S)$ and $|S| = 2 \geq |\operatorname{res}(*\,S)|$, also implying that $S = \operatorname{res}(*\,S)$. $\square$

Observation 8.12. *If $\left|\operatorname{res}\left(s_{\mathrm{M}}^{(i)}\right)\right| \leq 2$ for any $i \in [B+1]$, then $\operatorname{res}(s_{\mathrm{M}}^{(i)}) = \diamondsuit_{j=1}^{i} \operatorname{res}(g_j h_j)$.*

Proof. With $S := \bigdiamond_{j=1}^{i} \operatorname{res}(g_j h_j)$, we have that $\operatorname{res}\left(s_{\mathrm{M}}^{(i)}\right) = \operatorname{res}(* S)$. The claim thus follows from Observation 8.11. □

For the second case, $s_{\mathrm{M}}^{(i)} = \mathrm{MM}$, the structure of the code will be very helpful. Concretely, if in a valid string there is a metastable bit at position m, then the remaining $B - m$ following bits are the maximum codeword of a $(B-m)$-bit code.

Observation 8.13. *For $g \in V_B$, if there is an index $1 \leq m < B$ such that $g_m = \mathrm{M}$ then $g_{m+1,B} = 10^{B-m-1}$.*

Proof. List the codewords in order. By the recursive definition of the code, removing the first $m-1$ bits of the code leaves us with 2^{m-1} repetitions of $(B-m+1)$-bit code alternating between listing it in order and in reverse ("reflected") order. Also by the recursive definition, the m^{th} bit toggles only when the $(B-m)$-bit code resulting from removing it is at its last codeword, 10^{B-m-1}. □

Lemma 8.14. *Suppose that for valid strings $g, h \in V_B$, it holds that $s_{\mathrm{M}}^{(i)} = \mathrm{MM}$ for some $i \in [1, B]$. Then $g = h$ and $s_{\mathrm{M}}^{(j)} = \mathrm{MM}$ for all $j \in [i, B]$.*

Proof. For $R := \bigdiamond_{k=1}^{i} \operatorname{res}(g_k h_k) \subseteq \{0,1\}^2$, we have that $s_{\mathrm{M}}^{(i)} = * R$. As $* R = \mathrm{MM}$, it must hold that (i) $\{00, 11\} \subseteq R$ or (ii) $\{01, 10\} \subseteq R$. By Lemma 8.3, (i) implies that there are stabilizations $g', g'' \in \operatorname{res}(g_{1,i})$ and $h', h'' \in \operatorname{res}(h_{1,i})$ such that $g' = h'$, $\operatorname{par}(g') = 0$, $g'' = h''$, and $\operatorname{par}(g'') = 1$, while (ii) implies such g', g'', h', h'' with $g' <_G h'$ and $g'' >_G h''$. Checking the order Definition 8.2 (cf. Table 8.1), we see that both options necessitate that $g_{1,i} = h_{1,i}$ with some metastable bit.

Denote by $m \in [1, i-1]$ the index such that $g_m = h_m = \mathrm{M}$. Observation 8.13 shows that $g_{m+1,B} = h_{m+1,B} = 10^{B-m-1}$. In particular, $g = h$, showing (again by Lemma 8.3) that (i) or (ii) (in fact both) also apply to $\bigdiamond_{k=1}^{j} \operatorname{res}(g_k h_k)$ for each $j \in [i, B]$ (cf. Definition 8.5). We conclude that $s_{\mathrm{M}}^{(j)} = \mathrm{MM}$ for all such j. □

Putting these two pieces together yields (P2).

Theorem 8.15. *(P2) holds, i.e., $\forall g, h \in V_B, i \in [1, B]\colon s_{\mathrm{M}}^{(i)} = \bigdiamond_{\mathrm{M}}(g_{1,i}, h_{1,i})$.*

Proof. We show the claim by induction on i. Trivially, we have that $s_{\mathrm{M}}^{(0)} = s^{(0)} = 00$ and thus for $i = 1$ that

$$s_{\mathrm{M}}^{(1)} = s_{\mathrm{M}}^{(0)} \diamond_{\mathrm{M}} g_1 h_1 = 00 \diamond_{\mathrm{M}} g_1 h_1 = g_1 h_1 = \bigdiamond_{\mathrm{M}}(g_{1,1}, h_{1,1}) \,.$$

Hence, suppose that the claim has been established for $i - 1 \in [1, B-1]$ and consider index i. If $\left|\operatorname{res}\left(s_{\mathrm{M}}^{(i-1)}\right)\right| \leq 2$, Observation 8.12 states that $\operatorname{res}\left(s_{\mathrm{M}}^{(i-1)}\right) = s_{\mathrm{M}}^{(i-1)} = \bigdiamond_{j=1}^{i} \operatorname{res}(g_j h_j)$. Together with the induction hypothesis, this yields that

$$\begin{aligned}\Diamond_{\mathrm{M}}(g_{1,i},h_{1,i}) &= \Diamond_{\mathrm{M}}(g_{1,i-1},h_{1,i-1}) \diamond_{\mathrm{M}} g_ih_i\\ &= s_{\mathrm{M}}^{(i-1)} \diamond_{\mathrm{M}} g_ih_i = *\Big(\mathrm{res}\Big(s_{\mathrm{M}}^{(i-1)}\Big) \diamond \mathrm{res}(g_ih_i)\Big)\\ &= * \Diamond_{j=1}^{i} \mathrm{res}(g_jh_j) = s_{\mathrm{M}}^{(i)}\,.\end{aligned}$$

It remains to consider the case that $s_{\mathrm{M}}^{(i-1)} = \mathrm{MM}$. By Lemma 8.14, $s_{\mathrm{M}}^{(i)} = \mathrm{MM}$, too. Thus,

$$\Diamond_{\mathrm{M}}(g_{1,i},h_{1,i}) = s_{\mathrm{M}}^{(i-1)} \diamond_{\mathrm{M}} g_ih_i = \mathrm{MM} \diamond_{\mathrm{M}} g_ih_i = \mathrm{MM} = s_{\mathrm{M}}^{(i)}\,. \qquad \square$$

Remarks:

- As soon as two output bits become metastable, the simple relation given in Observation 8.12 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.
- Fortunately for us, MM is an absorbing "state" of the state machine, so we could handle this case without too much fuss.

8.4 Determining the Output Bits

We need to show that $\mathrm{out}_{\mathrm{M}}(s^{(i-1)}, g_ih_i) = \max_G\{g,h\}_i \min_G\{g,h\}_i$ for all $g, h \in V_B$ and $i \in \{1,\dots,B\}$.

Theorem 8.16. *(P3) holds, i.e., given valid inputs $g, h \in V_B$ and $i \in [1, B]$, $\mathrm{out}_{\mathrm{M}}(s_{\mathrm{M}}^{(i-1)}, g_ih_i) = (\max_G)_{\mathrm{M}}\{g,h\}_i(\min_G)_{\mathrm{M}}\{g,h\}_i$.*

Proof. Assume first that $\left|\mathrm{res}\left(s_{\mathrm{M}}^{(i-1)}\right)\right| \le 2$. Then

$$\begin{aligned}\mathrm{out}_{\mathrm{M}}(s_{\mathrm{M}}^{(i-1)}(g,h), g_ih_i) &= *\,\mathrm{out}\left(\mathrm{res}\left(s_{\mathrm{M}}^{(i-1)}(g,h)\right), \mathrm{res}(g_ih_i)\right)\\ &\overset{\text{Obs. 8.12}}{=} *\,\mathrm{out}\left(\Diamond_{j=1}^{i-1} \mathrm{res}(g_jh_j), \mathrm{res}(g_ih_i)\right)\\ &\overset{\text{Cor. 8.6}}{=} *\,(\max_G\{\mathrm{res}(g),\mathrm{res}(h)\}_i \min_G\{\mathrm{res}(g),\mathrm{res}(h)\}_i)\\ &= (\max_G)_{\mathrm{M}}\{g,h\}_i(\min_G)_{\mathrm{M}}\{g,h\}_i\,.\end{aligned}$$

Otherwise, $s_{\mathrm{M}}^{(i-1)} = \mathrm{MM}$. Then, by Lemma 8.14, $g = h$. In particular, $g_i = h_i$. Checking Definition 8.4, we see that for all $s \in \{0,1\}^2$ and $b \in \{0,1\}$, it holds that $\mathrm{out}(s, bb) = bb$. Therefore, $\mathrm{out}_{\mathrm{M}}(\mathrm{MM}, bb) = bb$ for all $b \in \{0,1,\mathrm{M}\}$ and

$$\mathrm{out}_{\mathrm{M}}(s_{\mathrm{M}}^{(i-1)}(g,h), g_ih_i) = g_ih_i = (\max_G)_{\mathrm{M}}\{g,h\}_i(\min_G)_{\mathrm{M}}\{g,h\}_i$$

in this case as well. $\square$

Remarks:

- We have decomposed the task of computing the output into computing $s_{\mathrm{M}}^{(i)}$, $i \in [B]$, and applying $\mathrm{out}_{\mathrm{M}}$.
- We have decomposed computing $s_{\mathrm{M}}^{(i)}$ into applying $\diamond_{\mathrm{M}}$ $i-1$ times.
- As $\mathrm{out}_{\mathrm{M}}$ 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 $\diamond_{\mathrm{M}}$ naively would yield a circuit of depth $\Omega(B)$!
- This is where we shamelessly exploit the associativity of $\diamond_{\mathrm{M}}$.

8.5 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 $g_i h_i$, $i \in \{1,\dots,B-1\}$, 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 8.17. *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 $i \in [2^b]$ the value $\pi_i := \bigodot_{j=0}^{i} g_j$ (where $\bigodot_{j=0}^{0} g_j = g_0$).*

Proof. We prove the statement by induction on b, where the claim is that a circuit of size (at most) $2^{b+1}|C|$ and depth (at most) $2bd(C)$ does the job. For $b=0$, the trivial circuit wiring the input to the output satisfies the claim; it has size and depth 0.

Hence, assume that the claim holds true for $b-1 \in \mathbb{N}_0$ and consider b. We recursively construct the desired circuit by using the circuit given by the induction hypothesis, see Figure **??**. Concretely, we use 2^{b-1} copies of C to (in parallel) compute $g'_i := g_{2i} \odot g_{2i+1}$, $i \in [2^{b-1}]$. We then feed these values into a (sub)circuit of size $?(b-1)|C|$ and depth $2(b-1)d(C)$ that outputs

$$\pi'_i := \bigodot_{j=0}^{i} g'_j = \bigodot_{j=0}^{2i} g_j$$

for all $i \in [2^{b-1}]$. Note that $\pi_i = \pi'_{(i-1)/2}$ for all odd $i \in [2^d]$. Then, adding another $2^{b-1}-1$ copies of C, in parallel we compute $\pi_i = \bigodot_{j=0}^{i} g_j = \pi_{i-1} \odot g_i = \pi'_{(i-2)/2} \odot g_i$ for all even $i \in [2^d] \setminus \{0\}$. Because $\pi_0 = g_0$ is already available as input, we thus obtain π_i for all $i \in [2^d]$. The size of the resulting circuit is bounded by $(2^{b-1} + 2^b + 2^{b-1} - 1)|C| < 2^{b+1}|C|$ and its depth by $d(C) + 2(b-1)d(C) + d(C) = 2bd(C)$. □

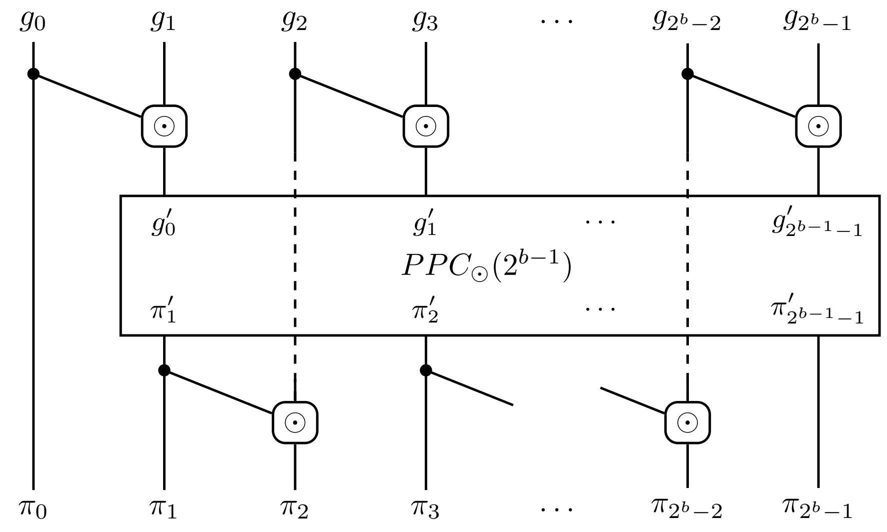

Figure 8.2: The recursive construction used in Theorem 8.17.

Corollary 8.18. *There is a comparator circuit of size $\mathcal{O}(B)$ and depth $\mathcal{O}(\log B)$ for valid strings (see Figure 8.3).*

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 8.17 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. □

Remarks:

- There are other parallel prefix computation circuit constructions. These are of interest only when caring about the factor-2 overhead in the depth of the circuit. Things then get interesting, as also fan-out becomes an issue (a large fan-out slows down the circuit as well).

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].

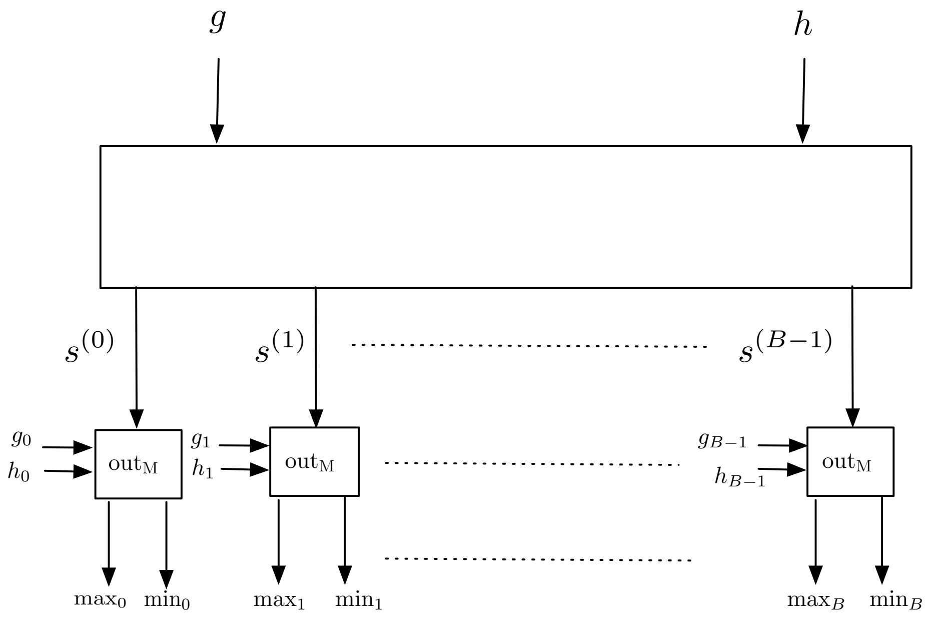

Figure 8.3: The Gray code comparator.

Bibliography

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