Logic Circuits

Chapter 2

Overview

- Many important functions computed with *straight-line programs*
  - No loops nor branches
  - Convenienly described with *circuits*
- Circuits are *directed acyclic graphs*
  - Characterized by *size* and *depth*
- Circuits where operations involved are Boolean are called *logic circuits*
Overview (cont.)

- Circuits where operations involved are algebraic are called \textit{algebraic circuits}
- Circuits where operations involved are comparisons are called \textit{comparator circuits}
- \textit{Logic circuits} are basic building blocks of real-world computers
- All machines with bounded memory can be built from logic circuits and binary memory units

Overview (cont.)

- Machines that perform finite-step computation can be simulated by logic circuits
- Chapter discusses:
  - Circuits \textit{vis-à-vis} straight-line programs
  - Class of functions computed by logic circuits
  - Circuit designs for a number of important functions
  - Important results about Boolean functions and logic circuits
  - \textit{Problem reduction} as a powerful tool of analysis
Designing Circuits

- Logic circuit
  - A directed, acyclic graph (DAG) whose vertices are labeled with Boolean functions (i.e., logic gates) or variables (i.e., inputs)
  - Computes a binary function
    \[ f : \mathcal{B}^n \rightarrow \mathcal{B}^m \]
    where \( n \) is the number of input variables and \( m \) is the number of outputs in the circuit

Designing Circuits (cont.)

- A binary function
  \[ f : \mathcal{B}^n \rightarrow \mathcal{B}^m \]

- Goal is to design efficient circuits
  - Small size (i.e., number of gates)
  - Small depth (i.e., length of longest path)
Designing Circuits (cont.)

- Logic circuits help provide framework for problem classification based on computational complexity
  - Used to identify hard computational problems such as \( P \)-complete languages and \( \textbf{NP} \)-complete languages (cf. Ch. 3)
- Show number of Boolean functions much greater than number of possible logic circuits of a given maximum size ⇒ most Boolean functions must be \textit{complex}

---

Straight-line Programs and Circuits

- Example from text (cf. pp. 36-38):

  ![Diagram](image)

  **Functional description:**
  
  \[
  \begin{align*}
  g_1 & := x; \\
  g_2 & := y; \\
  g_3 & := \neg g_1; \\
  g_4 & := \neg g_2; \\
  g_5 & := g_1 \land g_4; \\
  g_6 & := g_2 \land g_3; \\
  g_7 & := g_5 \lor g_6;
  \end{align*}
  \]

  **Straight-line program:**
  
  \[
  \begin{align*}
  1 & \text{ READ } x \\
  2 & \text{ READ } y \\
  3 & \text{ NOT } 1 \\
  4 & \text{ NOT } 2 \\
  5 & \text{ AND } 1 \text{ 4} \\
  6 & \text{ AND } 3 \text{ 2} \\
  7 & \text{ OR } 5 \text{ 6} \\
  8 & \text{ OUTPUT } 5 \\
  9 & \text{ OUTPUT } 7
  \end{align*}
  \]
Formal definitions

Definition 2.2.1: A straight-line program is a set of steps each of which is an input step, denoted by \((s \text{ READ } x)\), or an output step, denoted by \((s \text{ OUTPUT } i)\), or a computation step, denoted by \((s \text{ OP } i \ldots k)\).

Here \(s\) is the (ordinal) number of a step (allowing us to see the sequence by which the steps are to be executed).
\(x\) denotes an input variable.

The arguments \(i \ldots k\) for an OP step must be less than \(s\) the step number of that OP step, i.e., \(s > i, \ldots, k\).

Straight-line Programs and Circuits (cont.)

Formal definitions (cont.)

Definition 2.2.1: A circuit is the graph of a straight-line program. The fan-in of a circuit is the maximum in-degree of any vertex. The fan-out is the maximum out-degree of any vertex. A gate is any vertex associated with a computation step (i.e., an OP step).

In our example, both fan-in and fan-out are equal to 2. The gates are those vertices representing the NOT, AND, and OR operations.
Straight-line Programs and Circuits (cont.)

- Formal definitions (cont.)
  - **Definition 2.2.1**: The basis $\Omega$ of a circuit and its corresponding straight-line program is the set of operations that they use.
    The bases of Boolean straight-line programs and logic circuits contain only Boolean functions.
    The standard basis $\Omega_0$ is the set $\{\text{NOT}, \text{AND}, \text{OR}\}$.
    Our example uses the standard basis.

Functions Computed by Circuits

- Formal definition
  - **Definition 2.2.2**: Let $g_s$ be the function computed by the s-th step of a straight-line program.
    If the s-th step is the input step (s READ $x$), then $g_s = x$.
    If the s-th step is the computation step (s OP $i \ldots k$), then $g_s = \text{OP}(g_i, \ldots, g_k)$, where $g_i, \ldots, g_k$ are the functions computed by steps $i \ldots k$.
    If a straight-line program has $n$ inputs and $m$ outputs, it computes a function $f: \mathcal{B}^n \to \mathcal{B}^m$. If $s_1, s_2, \ldots, s_m$ are the output steps, then $f = (g_1, g_2, \ldots, g_m)$.
    The function computed by a circuit is the function computed by the corresponding straight-line program.
Example from text (cf. pp. 36-38):

Functions computed by circuit:

\[ g_1 := x; \]
\[ g_2 := y; \]
\[ g_3 := \neg x; \]
\[ g_4 := \neg y; \]
\[ g_5 := x \land \neg y; \]
\[ g_6 := y \land \neg x; \]
\[ g_7 := (x \land \neg y) \lor (y \land \neg x); \]

Straight-line program:

1. READ x
2. READ y
3. NOT 1
4. NOT 2
5. AND 1 4
6. AND 3 2
7. OR 5 6
8. OUTPUT 5
9. OUTPUT 7

\[ f(x,y) = (g_5, g_7) \]

Circuits That Compute Functions

- Given a circuit, we know how to determine the function it computes.
- Given a function, how do we construct a circuit (and straight-line program) that computes it?
  
  Method involves following steps:
  
  - Construct functional table
  - Express in normal form, can be transformed directly into a circuit
  - Simplify to reduce circuit complexity
Circuits That Compute Functions (cont.)

- Example: \( f: \mathbb{B}^3 \rightarrow \mathbb{B}^2 \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Express output in disjunctive normal form (DNF):

\[
\begin{align*}
y_1 &= (\neg x_1 \land \neg x_2 \land \neg x_3) \lor \\
     &\quad (\neg x_1 \land x_2 \land \neg x_3) \lor \\
     &\quad (x_1 \land \neg x_2 \land \neg x_3) \lor \\
     &\quad (x_1 \land x_2 \land x_3)
\end{align*}
\]

\[
\begin{align*}
y_2 &= (\neg x_1 \land \neg x_2 \land \neg x_3) \lor \\
     &\quad (\neg x_1 \land \neg x_2 \land x_3) \lor \\
     &\quad (\neg x_1 \land x_2 \land x_3) \lor \\
     &\quad (x_1 \land \neg x_2 \land \neg x_3) \lor \\
     &\quad (x_1 \land x_2 \land \neg x_3) \lor \\
     &\quad (x_1 \land x_2 \land x_3)
\end{align*}
\]

Each of the terms can be realized by a simple circuit:

- Circuit that computes the function is made up of the OR’s of these components.
Circuits That Compute Functions (cont.)

- Circuit that computes the function is made up of the OR’s of these simpler components:

- **THEOREM 2.3.1** Every function \( f : B^n \rightarrow B^m \) can be realized by a logic circuit.

Circuit Complexity Measures

- **Definition 2.2.3**: The size of a logic circuit is the number of gates it contains. The depth is the number of gates on the longest path through the circuit.

  The circuit size, \( C_\Omega(f) \), and circuit depth, \( D_\Omega(f) \), of a Boolean function \( f \) are defined as the smallest size and smallest depth of any circuit, respectively, over the basis \( \Omega \) of \( f \).

- Clearly, it is desirable to construct the smallest or most shallow circuit for a function –
  - If the circuit is small in size, the complexity of the function computed must also be modest
  - If the circuit is shallow in depth, the speed of computation tends to be faster when the circuit is physically realized