COMPSCI 501: Formal Language Theory
Lecture 31: Circuit Complexity

Marius Minea
marius@cs.umass.edu
University of Massachusetts Amherst
10 April 2019

What can we implement via circuits?

- A tool for attacking P = NP?
- Identify parallelizable computation vs. sequential bottleneck
- Alternative NP-completeness for SAT

Circuit families and complexity

Turing machine can be fixed, variable-length input
Circuit is fixed-size \( \implies \) family of circuits, one per input length

**Def:** A circuit family \( C \) is an infinite list of circuits, \( (C_0, C_1, C_2, \ldots) \), where \( C_n \) has \( n \) input variables.

\( C \) decides a language \( A \) over \( \{0, 1\} \) if for every string \( w \), \( w \in A \iff C_n(w) = 1 \). \( (n = |w|) \)

Can define:

- Size, size-minimal circuits (no smaller equivalent)
- Size complexity of a family: \( f : \mathbb{N} \rightarrow \mathbb{N}, f(n) = \text{size}(C_n) \)
- Depth, depth minimal, depth complexity (measure of sequential-ness)

**Circuit complexity of a language = size complexity of minimal circuit family**

CIRCUIT-SAT is NP-complete

**CIRCUIT-SAT = \{ (C) | C is a satisfiable circuit \}**

Clearly, it’s in NP (witness = satisfying assignment, check in P)

Mapping reduction for any language \( A \in \text{NP} \) to CIRCUIT-SAT

\( f(w) = (C) \) with \( w \in A \iff C \) is satisfiable

\( A \in \text{NP} \implies \) has poly-time verifier \( V' \) with input \( \langle x, c \rangle \)

Construct circuit for \( V' \) using tableau method.
Inputs for \( w \) filled in, inputs for \( c \) free.
\( C \) satisfiable iff a certificate exists \( \iff w \in A \)

Circuit building time/size: square in verifier size \( \implies \) polynomial

Time vs. Circuit Complexity

**Theorem:** Let \( t : \mathbb{N} \rightarrow \mathbb{N} \) be a function with \( t(n) \geq n \).
If \( A \in \text{TIME}(t(n)) \), then \( A \) has circuit complexity \( O(t^2(n)) \).

Consequence: if we were to find a language in NP with more than polynomial circuit complexity, that language can’t be in P!

Assume running time \( t(n) \)

Build \( t(n) \times t(n) \) tableau of configurations
Build circuit \( C_n \) from tableau.

- Standardize accept: halt on leftmost cell, write blank
- Encode head together with tape symbol: \( |(Q \times \Gamma) \cup \Gamma| = k \)
- One bit (wire) saying “cell[i,j] has symbol \( s \)” \( \implies \) \( k t^2(n) \) bits
- \( bit[i,j,s] \) can be \( bit[i-1,j,s] \) (keep previous) or come from cells \( j - 1/j + 1 \) in row \( i - 1 \) (head move/write)
- Cells in row 1 wired to input word

From Sequential to Combinational

Circuits: AND, OR, NOT gates. Acyclic (why?)
Simplest computations: output = \( f(\text{input}) \).
May generalize to multiple output bits, \( \{0, 1\}^k \)
\( z = x + y \). Binary representation, combinational circuit.

How about:

- \( s = 0 \)
- For \( i = 1 \) to \( n \)
  - \( s = s + a[i] \)
- End for

Can view \( a[i] \) as inputs
Introduce successive copies of \( s \) and unroll:

\( s_0 = 0 \land s_1 = s_0 + a_1 \land \ldots \land s_n = s_{n-1} + a_n \)
Polynomial time algorithm \( \implies \) polynomial size circuit

\( n \) stages: \( \implies \) circuit depth matters (lecture on parallelism)

Circuit building time/size: square in verifier size \( \implies \) polynomial

What can we implement via circuits?

A tool for attacking P = NP?
Identify parallelizable computation vs. sequential bottleneck
Alternative NP-completeness for SAT

CIRCUIT-SAT is NP-complete

**CIRCUIT-SAT = \{ (C) | C is a satisfiable circuit \}**

Clearly, it’s in NP (witness = satisfying assignment, check in P)

Mapping reduction for any language \( A \in \text{NP} \) to CIRCUIT-SAT

\( f(w) = (C) \) with \( w \in A \iff C \) is satisfiable

\( A \in \text{NP} \implies \) has poly-time verifier \( V' \) with input \( \langle x, c \rangle \)

Construct circuit for \( V' \) using tableau method.
Inputs for \( w \) filled in, inputs for \( c \) free.
\( C \) satisfiable iff a certificate exists \( \iff w \in A \)

Circuit building time/size: square in verifier size \( \implies \) polynomial

Time vs. Circuit Complexity

**Theorem:** Let \( t : \mathbb{N} \rightarrow \mathbb{N} \) be a function with \( t(n) \geq n \).
If \( A \in \text{TIME}(t(n)) \), then \( A \) has circuit complexity \( O(t^2(n)) \).

Consequence: if we were to find a language in NP with more than polynomial circuit complexity, that language can’t be in P!

Assume running time \( t(n) \)

Build \( t(n) \times t(n) \) tableau of configurations
Build circuit \( C_n \) from tableau.

- Standardize accept: halt on leftmost cell, write blank
- Encode head together with tape symbol: \( |(Q \times \Gamma) \cup \Gamma| = k \)
- One bit (wire) saying “cell[i,j] has symbol \( s \)” \( \implies \) \( k t^2(n) \) bits
- \( bit[i,j,s] \) can be \( bit[i-1,j,s] \) (keep previous) or come from cells \( j - 1/j + 1 \) in row \( i - 1 \) (head move/write)
- Cells in row 1 wired to input word

From Sequential to Combinational

Circuits: AND, OR, NOT gates. Acyclic (why?)
Simplest computations: output = \( f(\text{input}) \).
May generalize to multiple output bits, \( \{0, 1\}^k \)
\( z = x + y \). Binary representation, combinational circuit.

How about:

- \( s = 0 \)
- For \( i = 1 \) to \( n \)
  - \( s = s + a[i] \)
- End for

Can view \( a[i] \) as inputs
Introduce successive copies of \( s \) and unroll:

\( s_0 = 0 \land s_1 = s_0 + a_1 \land \ldots \land s_n = s_{n-1} + a_n \)
Polynomial time algorithm \( \implies \) polynomial size circuit

\( n \) stages: \( \implies \) circuit depth matters (lecture on parallelism)
3-SAT reduction from \textit{CIRCUIT-SAT}

Express truth table for each gate as conjunction of implications:

\[ x = a \text{ AND } b: \text{ (similar for OR, NOT)} \]

\[ (a \land b \rightarrow x) \land (\neg a \land b \rightarrow \neg x) \land (a \land \neg b \rightarrow \neg x) \land (\neg a \land \neg b \rightarrow \neg x) \]

\[ (\neg a \lor \neg b \lor x) \land (a \lor \neg b \lor \neg x) \land (\neg a \lor b \lor \neg x) \land (a \lor b \lor \neg x) \]

Could simplify to \((\neg a \lor \neg b \lor \neg x) \land (a \lor \neg x) \land (b \lor \neg x)\)

(recall Tseitin transform)

Construction: linear time, linear-size formula \(\phi\)

(one variable for each input and wire; \(\leq 3\) literals/clause)

\(\phi\) precisely describes \(C\)’s computation: satisfiable iff \(C\) is