CMOS Silicon Algebra

Compositional Semantics and Correct Lowering from Logic to Physical Computation

Adrian Diamond
AAD Systems
adrian@aadsystems.com

September 2026

Abstract

Modern hardware compilation moves computations through multiple representational layers, from logical specifications to gate networks, transistor structures, and ultimately physical electrical behavior. The semantic contracts connecting these layers are often implicit, distributed across tools, or stated only at selected abstraction boundaries. This paper develops *CMOS Silicon Algebra*, a compositional framework for representing and lowering computation from formal logic to CMOS realization.

The intended framework separates logical denotation, gate-network semantics, transistor-network semantics, and physical interpretation; introduces explicit lowering and abstraction maps between these layers; and formulates correctness as semantic-preservation conditions that compose across a lowering pipeline. This provides a mathematical basis for certified circuit transformations and for a future CMOS Silicon Compiler whose executable lowerings can be checked against the formal model.

A further goal is to distinguish exact logical correctness from nonideal physical behavior—including voltage margins, propagation delay, loading, and switching effects—so that software-level meaning and physical computation can be connected without conflating their semantic regimes.

Contents

1 Introduction

Modern digital computation is realized through a sequence of representations. A computation may begin as a formal Boolean specification, be transformed into a network of logic gates, be lowered further into transistor structures, and ultimately be realized as physical electrical behavior in silicon.

These stages are related, but they are not identical. A Boolean expression is not a gate network; a gate network is not a transistor network; and an ideal transistor network is not the same mathematical object as its physical electrical realization. Each representation introduces distinctions that are absent from the one above it.

The central problem of this paper is therefore to make the semantic relationships between these representations explicit.

The principal descent studied here is

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys.}}$$

The objective is not merely to provide translations between these layers. The objective is to state what it means for each translation to be *correct*, to prove that the corresponding correctness conditions compose, and to identify which properties remain exact as the computation approaches physical realization.

1.1 Motivation

Logic synthesis, transistor-level modeling, hardware verification, compiler intermediate representations, and physical implementation have each developed substantial mathematical and engineering theories. Switch-level models of MOS circuits provide abstractions of transistor conduction [1, 2]. Logic synthesis and technology mapping transform Boolean structure into implementation-oriented networks [4, 5]. Formal hardware systems such as Kami demonstrate that hardware descriptions and their refinements can be treated as proof-bearing objects [6]. Modern compiler infrastructures such as MLIR and CIRCT make multiple intermediate representations and explicit lowering boundaries central architectural concepts [7, 8]. Verified compilation similarly demonstrates that semantic preservation may be proved compositionally across compiler passes [9, 10].

The present problem lies at the intersection of these perspectives.

A hardware compiler may correctly manipulate Boolean functions while leaving implicit the relationship between those functions and the transistor networks that realize them. Conversely, a transistor-level model may describe conduction and switching without specifying how its behavior relates to a higher-level source language. Physical circuit analysis introduces still finer distinctions—voltages, delays, loading, capacitance, and technological parameters—that cannot simply be identified with Boolean semantics.

A unified framework therefore requires explicit semantic boundaries.

At each boundary, the target representation may contain more structure than the source. Correctness does not require that this additional structure disappear. It requires that the intended source meaning be recoverable through an appropriate abstraction.

This observation motivates the basic architecture of CMOS Silicon Algebra.

1.2 Research Question

The governing question is:

How can a computation specified in formal logical terms be lowered compositionally through gate and CMOS representations while preserving its meaning down to a physically interpreted implementation?

A satisfactory answer must address several distinct problems.

First, each representation requires its own syntax or structural domain and its own semantics. Second, lowering maps between representations must be defined. Third, the semantic relationships between those maps must be stated precisely. Fourth, circuit transformations performed during compilation must preserve the appropriate notion of equivalence. Finally, the transition from ideal digital semantics to physical computation must state explicitly the assumptions under which electrical behavior realizes the discrete model.

1.3 Semantic Preservation Across Representation Boundaries

Consider a generic lowering

$$L_{A\to B} : A \to B$$

between representation spaces A and B.

Let

$$[\![-]\!]_A : A \to S_A \qquad \text{and} \qquad [\![-]\!]_B : B \to S_B$$

be their semantic maps, and let

$$\alpha_{B\to A} : S_B \rightharpoonup S_A$$

abstract target-level behavior into the semantic domain of the source.

The characteristic correctness condition is

$$\boxed{\alpha_{B\to A}\left([\![L_{A\to B}(x)]\!]_B\right) = [\![x]\!]_A.}$$

This equation is the principal semantic pattern of the paper.

The target representation may introduce gates, transistor topology, rail states, continuous voltages, timing behavior, or technology-dependent parameters. Those distinctions need not exist at the source level. A lowering is correct when the abstraction appropriate to that boundary recovers the source denotation.

For the logic-to-gate boundary developed below, source and target ultimately share Boolean behavior, so preservation can be stated directly as

$$[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}} = [\![P]\!]_{\mathsf{Logic}}.$$

For the gate-to-CMOS boundary, the CMOS representation contains rail and conduction structure that must first be abstracted:

$$\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho).$$

At the physical boundary, correctness becomes conditional on electrical and timing assumptions. Under an admissibility contract and a sufficient settling interval Δ, the required relationship has the form

$$\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho,\lambda)\right) = [\![C]\!]_{\mathsf{CMOS}}(\rho).$$

When the individual contracts hold, they compose. The complete physical correctness condition therefore becomes

$$\boxed{\alpha_C^*\Big(\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}}(\rho,\lambda)\right)\Big) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

The exact discrete theory and the conditional physical theory are thus joined without identifying their semantic domains.

1.4 Contributions

This paper develops the following components of that architecture.

1. It defines distinct semantic layers for logical computation, gate networks, ideal CMOS transistor networks, and physical realization.
2. It develops a compositional conduction algebra for complementary static CMOS in which series and parallel transistor structures have exact Boolean conduction semantics.
3. It defines CMOS duality and a complementary-cell construction that produces valid static CMOS cells from supported conduction kernels.
4. It defines a gate-to-CMOS lowering and proves both local cell correctness and local-to-global correctness for finite acyclic gate networks.
5. It formulates semantic lowering contracts abstractly and proves that compatible contracts compose, yielding end-to-end logic-to-CMOS preservation.
6. It develops semantics-preserving transformation relations at the gate, conduction, rail, and abstract CMOS levels, thereby separating correctness from implementation cost.
7. It introduces the physical layer as an explicit refinement boundary in which continuous electrical behavior is related to ideal CMOS semantics through a settling abstraction and an admissibility contract.
8. It shows how the same architecture extends from combinational denotation to sequential state-transition semantics while identifying the additional temporal obligations required for a complete sequential theory.
9. It derives from the formal model an architecture for a future *CMOS Silicon Compiler* in which representation changes become compiler passes and semantic preservation becomes a checkable pass contract.

The contribution is therefore not a claim that Boolean logic, CMOS switching, logic synthesis, hardware verification, or compiler lowering are individually new. Rather, the aim is to organize the descent from formal computation to physical CMOS realization within one compositional semantic architecture.

1.5 Scope

The exact core of the present theory concerns finite combinational computation and complementary static CMOS under an ideal switch abstraction.

Within that domain, the logical, gate, conduction, and rail-level correctness claims are exact relative to their stated semantics.

The physical layer serves a different purpose. This paper does not attempt to derive MOSFET current–voltage equations, solve continuous transistor dynamics, perform parasitic extraction, replace circuit simulation, or provide a complete physical-design methodology. Instead, it specifies the semantic contract that a more detailed electrical realization must satisfy in order to implement the ideal CMOS model.

Similarly, sequential computation is treated as an extension of the semantic architecture rather than as a complete theory of latches, flip-flops, metastability, clocks, and feedback.

These boundaries are deliberate. They separate exact semantic preservation from technology-dependent physical claims and prevent equivalence at one abstraction level from being mistaken for equivalence at another.

1.6 Organization

Section 2 introduces the semantic layers and the relationships between them. Section 3 defines the formal source logic and its denotational semantics. Section 4 develops the gate-network representation and the logic-to-gate lowering.

Section 5 introduces the CMOS transistor-network algebra. Section 6 defines gate-to-CMOS lowering and proves its local and compositional correctness. Section 7 composes the lowering contracts into an end-to-end preservation theorem. Section 8 develops certified circuit transformations and separates semantic admissibility from optimization cost.

Section 9 introduces the physical interpretation and its conditional refinement contract. Section 10 describes the extension to sequential state-transition semantics. Section 11 derives the architecture of the future CMOS Silicon Compiler. Section 12 provides worked examples. Section 13 situates the framework relative to prior work, Section 14 discusses its scope and limitations, and Section 15 concludes.

The appendices collect the notation and the principal proof obligations.

The resulting framework is organized around a single invariant:

meaning is preserved across every correct representation boundary.

2 Semantic Layers

CMOS Silicon Algebra distinguishes four principal representation layers:

$$\mathsf{Logic}, \qquad \mathsf{Gate}, \qquad \mathsf{CMOS}, \qquad \mathsf{Phys}.$$

These layers are related by lowering maps,

$$\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys},$$

but they do not share a single representation or semantic domain.

The distinction is essential. Each downward step introduces implementation structure that is intentionally absent from the layer above it. The role of an abstraction map is to forget those additional distinctions when they are irrelevant to the meaning represented at the preceding layer.

Accordingly, the framework separates three notions:

1. the *representation* carried by a layer;
2. the *semantics* assigned to objects in that representation; and
3. the *abstraction* used to compare a lower-level behavior with the meaning of a higher-level object.

This section fixes that architecture before the individual layers are developed formally.

2.1 Logical Layer

The logical layer contains finite formal computations over Boolean interfaces. Its objects are written

$$P, Q \in \mathsf{Logic}.$$

For finite input and output interfaces I and O, a logical computation has denotation of the form

$$[\![P]\!]_{\mathsf{Logic}} : \mathbb{B}^I \to \mathbb{B}^O.$$

The logical semantics records the externally observable Boolean computation. It does not prescribe a gate basis, signal-flow graph, transistor topology, or physical implementation.

Two logical objects may therefore differ syntactically while denoting the same Boolean function. Their semantic equivalence is determined by denotation:

$$P \equiv_{\mathsf{Logic}} Q \quad \Longleftrightarrow \quad \forall \rho : I \to \mathbb{B}, \qquad [\![P]\!]_{\mathsf{Logic}}(\rho) = [\![Q]\!]_{\mathsf{Logic}}(\rho).$$

The formal syntax and compositional semantics of this layer are developed in Section 3.

2.2 Gate Layer

The gate layer makes the internal Boolean signal-flow structure of a computation explicit.

An object

$$G \in \mathsf{Gate}$$

contains gate nodes, wires, primary inputs, designated outputs, and the dependencies connecting them. In the combinational fragment considered in the exact core of the paper, networks are finite and acyclic.

For compatible interfaces I and O, gate semantics has the form

$$[\![G]\!]_{\mathsf{Gate}} : \mathbb{B}^I \to \mathbb{B}^O.$$

Although the logical and gate layers may share the same Boolean semantic codomain, they retain different representation structure. A source expression may be lowered into many distinct gate networks having the same denotation.

The logic-to-gate correctness requirement is therefore

$$\boxed{[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}}(\rho) = [\![P]\!]_{\mathsf{Logic}}(\rho)}$$

for every supported source computation P and input valuation ρ.

Gate-network structure and the construction of $\mathcal{L}_G$ are developed in Section 4.

2.3 CMOS Layer

The CMOS layer represents computation using idealized transistor conduction structure.

An object

$$C \in \mathsf{CMOS}$$

contains transistor networks, control signals, supply rails, output nodes, and their connectivity. Its semantics is intentionally richer than Boolean function semantics because an ideal transistor network may distinguish valid high and low outputs from floating or conflicting states.

For a CMOS network with designated outputs, the semantics

$$[\![C]\!]_{\mathsf{CMOS}}(\rho)$$

returns the corresponding ideal rail-state observation under valuation ρ.

For a single output, the possible rail observations are represented by subsets of the high and low supply rails:

$$\varnothing, \qquad \{H\}, \qquad \{L\}, \qquad \{H, L\}.$$

These correspond respectively to an undriven output, a valid high output, a valid low output, and a rail conflict.

The Boolean abstraction

$$\alpha_C$$

is therefore partial. On valid rail states,

$$\alpha_C(\{H\}) = 1, \qquad \alpha_C(\{L\}) = 0,$$

while invalid or non-Boolean rail states remain outside its Boolean domain.

For multi-output networks, the componentwise extension is written

$$\alpha_C^*.$$

The characteristic gate-to-CMOS correctness condition is

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho).}$$

The transistor-network algebra supporting this semantics is developed in Section 5, and the gate-to-CMOS lowering is developed in Section 6.

2.4 Physical Layer

The physical layer contains electrical realizations of CMOS structure.

An object

$$P_{\text{phys}} \in \mathsf{Phys}$$

may exhibit continuous voltages, propagation delays, loading effects, technology dependence, and time-varying behavior. These distinctions are deliberately absent from the ideal CMOS semantics.

Consequently, physical semantics cannot in general be identified directly with a Boolean function or a static rail state. In Section 9 it is modeled as a set-valued trace semantics of the form

$$[\![P_{\text{phys}}]\!]_{\mathsf{Phys}}(\rho, \lambda),$$

where ρ is the input valuation and λ collects physical and technology-dependent parameters.

A settling abstraction

$$\alpha_P^{[\Delta]}$$

observes the behavior over an interval of duration Δ and extracts the corresponding ideal CMOS rail behavior when the required physical conditions hold.

Physical correctness therefore has the conditional form

$$\boxed{\alpha_P^{[\Delta]}\,([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho, \lambda)) = [\![C]\!]_{\mathsf{CMOS}}(\rho),}$$

subject to an explicit admissibility predicate.

This final boundary differs from the discrete boundaries above it. The logic-to-gate and gate-to-CMOS contracts are exact within their stated mathematical models. The CMOS-to-physical contract is conditional on the electrical assumptions under which the physical implementation realizes the ideal model.

2.5 Lowering and Abstraction

The semantic architecture can be expressed generically.

Let

$$A \xrightarrow{L_{A\to B}} B$$

be a lowering between two representation spaces, with semantic maps

$$[\![-]\!]_A : A \to S_A$$

and

$$[\![-]\!]_B : B \to S_B.$$

Suppose further that

$$\alpha_{B\to A} : S_B \rightharpoonup S_A$$

forgets the distinctions introduced by the target representation.

The lowering is semantically correct on its supported domain when

$$\boxed{\alpha_{B\to A}\left([\![L_{A\to B}(x)]\!]_B\right) = [\![x]\!]_A.}$$

The abstraction may be the identity when source and target share the same semantic codomain, as at the logic-to-gate boundary. It may be partial when the target admits behaviors that have no valid source-level interpretation, as at the CMOS boundary. It may also depend on admissibility and observation conditions, as at the physical boundary.

Thus lowerings introduce structure while abstractions discard structure:

$$\begin{array}{ccc} A & \xrightarrow{L_{A\to B}} & B \\ \downarrow [\![\cdot]\!]_A & & \downarrow [\![\cdot]\!]_B \\ S_A & \xleftarrow{\alpha_{B\to A}} & S_B. \end{array}$$

Correctness means that this semantic comparison commutes on the supported domain.

2.6 Composition of Semantic Boundaries

The central advantage of explicit boundary contracts is compositionality.

Suppose

$$A \xrightarrow{L_{A\to B}} B \xrightarrow{L_{B\to C}} C$$

are correct lowerings with abstractions

$$\alpha_{B\to A} \qquad \text{and} \qquad \alpha_{C\to B}.$$

Then

$$\alpha_{B\to A}\left(\alpha_{C\to B}\left([\![L_{B\to C}(L_{A\to B}(x))]\!]_C\right)\right) = [\![x]\!]_A.$$

This principle is proved formally in Section 7. It allows end-to-end correctness to be decomposed into local proof obligations rather than requiring one monolithic proof spanning every implementation level simultaneously.

For the principal descent of this paper, the resulting architecture is

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys}}$$

together with abstraction in the opposite semantic direction.

The representation chain and the proof chain therefore align:

$$\boxed{\text{representation boundary} \longleftrightarrow \text{proof boundary.}}$$

2.7 Abstraction-Relative Equivalence

Because each layer forgets different information, equivalence is relative to the semantic level at which objects are observed.

Logical equivalence preserves Boolean denotation. Gate equivalence preserves gate-network Boolean behavior. CMOS rail equivalence preserves ideal rail observations. Abstract CMOS equivalence preserves only the Boolean behavior obtained after rail abstraction. Physical equivalence would require still stronger agreement over electrical behavior.

Consequently,

$$\text{Boolean equivalence} \not\Rightarrow \text{structural identity},$$

and, more specifically,

$$\boxed{\text{Boolean equivalence} \not\Rightarrow \text{physical equivalence.}}$$

Two implementations may compute the same Boolean function while differing in transistor topology, capacitance, delay, loading, switching activity, energy, or other implementation-dependent properties.

This hierarchy is not a defect of the framework. It is what permits multiple representations and implementations of the same computation while keeping their correctness conditions precise.

The remaining sections instantiate this architecture layer by layer.

3 Formal Source Logic

The logical layer provides the semantic reference point for the lowering pipeline. Its purpose is deliberately modest: to represent finite combinational Boolean computations without committing to a gate basis, signal-flow graph, transistor topology, or physical realization.

The source language therefore distinguishes the *meaning* of a Boolean computation from the structures subsequently introduced to implement it.

3.1 Boolean Interfaces

Let

$$\mathbb{B} = \{0, 1\}.$$

For a finite set I of input names, write

$$\mathbb{B}^I$$

for the set of Boolean valuations

$$\rho : I \to \mathbb{B}.$$

Likewise, for a finite output interface O, an output valuation is an element of

$$\mathbb{B}^O.$$

A source computation with input interface I and output interface O will therefore denote a function

$$\mathbb{B}^I \to \mathbb{B}^O.$$

The use of named finite interfaces rather than positional bit vectors makes renaming, substitution, and composition explicit without imposing any particular implementation structure.

3.2 Syntax

For a finite input interface I, Boolean expressions are generated by

$$e ::= 0 \mid 1 \mid x \mid \neg e \mid (e \wedge e) \mid (e \vee e), \qquad x \in I.$$

The constants 0 and 1 denote Boolean false and true, respectively.

Other Boolean operations are treated as derived syntax. For example,

$$e_1 \text{ NAND } e_2 = \neg(e_1 \wedge e_2),$$

and

$$e_1 \text{ NOR } e_2 = \neg(e_1 \vee e_2).$$

The chosen basis is not intended to prescribe the eventual gate basis. Its purpose is only to provide a simple functionally complete source language.

Definition 3.1 (Source computation)**.** *Let I and O be finite interfaces. A source computation*

$$P : I \Rightarrow O$$

is a family of Boolean expressions

$$P = (e_o)_{o \in O},$$

where every e_o has free variables drawn from I.

A single-output Boolean expression is the special case in which O contains one element.

For example, the expression used later in the worked example,

$$P_F = (a \wedge b) \vee c,$$

is a source computation with input interface

$$I = \{a, b, c\}$$

and a single Boolean output.

The syntax does not assign implementation significance to repeated subexpressions. If the same expression occurs twice, the source language does not determine whether a later gate realization duplicates that computation or shares it through fan-out. Such distinctions belong to the gate layer.

3.3 Denotational Semantics

The interpretation of a source expression under

$$\rho : I \to \mathbb{B}$$

is defined recursively by

$$[\![0]\!]_\rho = 0, \qquad [\![1]\!]_\rho = 1,$$

$$[\![x]\!]_\rho = \rho(x),$$

$$[\![\neg e]\!]_\rho = 1 - [\![e]\!]_\rho,$$

$$[\![e_1 \wedge e_2]\!]_\rho = [\![e_1]\!]_\rho \wedge [\![e_2]\!]_\rho,$$

and

$$[\![e_1 \vee e_2]\!]_\rho = [\![e_1]\!]_\rho \vee [\![e_2]\!]_\rho.$$

The semantics is therefore compositional: the denotation of a compound expression is determined entirely by the denotations of its immediate subexpressions.

For a source computation

$$P = (e_o)_{o \in O} : I \Rightarrow O,$$

define

$$[\![P]\!]_{\mathsf{Logic}} : \mathbb{B}^I \to \mathbb{B}^O$$

by

$$\boxed{[\![P]\!]_{\mathsf{Logic}}(\rho)(o) = [\![e_o]\!]_\rho.}$$

Thus the logical semantics records exactly the Boolean behavior visible at the source interface.

For the composite example,

$$P_F = (a \wedge b) \vee c,$$

the denotation is

$$[\![P_F]\!]_{\mathsf{Logic}}(a, b, c) = (a \wedge b) \vee c.$$

No gate decomposition or transistor realization is part of this definition.

3.4 Substitution

Substitution supplies the basic compositional operation of the source language.

Let I and J be finite interfaces, and let

$$\sigma : I \to \mathrm{Expr}(J)$$

assign to each variable $x \in I$ an expression over J.

For an expression e over I, write

$$e[\sigma]$$

for the expression obtained by simultaneously replacing each occurrence of $x \in I$ by $\sigma(x)$.

Given a valuation

$$\rho : J \to \mathbb{B},$$

the substitution σ induces a valuation

$$\widehat{\sigma}(\rho) : I \to \mathbb{B}$$

defined by

$$\widehat{\sigma}(\rho)(x) = [\![\sigma(x)]\!]_\rho.$$

Lemma 3.2 (Substitution)**.** *For every Boolean expression e over I, substitution $\sigma : I \to \mathrm{Expr}(J)$, and valuation $\rho : J \to \mathbb{B}$,*

$$\boxed{[\![e[\sigma]]\!]_\rho = [\![e]\!]_{\widehat{\sigma}(\rho)}.}$$

Proof. By structural induction on e.

The constant cases are immediate. If $e = x$, then

$$[\![x[\sigma]]\!]_\rho = [\![\sigma(x)]\!]_\rho = \widehat{\sigma}(\rho)(x).$$

For negation, conjunction, and disjunction, apply the induction hypothesis to the immediate subexpressions and then use the corresponding clause of the denotational semantics. □

The lemma states formally that syntactic substitution agrees with semantic substitution of Boolean values.

3.5 Composition of Source Computations

Let

$$P = (e_o)_{o \in O} : I \Rightarrow O$$

and

$$Q = (q_k)_{k \in K} : O \Rightarrow K.$$

The composite

$$Q \circ P : I \Rightarrow K$$

is obtained by substituting the output expression e_o of P for every input variable o occurring in Q.

Equivalently, if

$$\sigma_P(o) = e_o,$$

then

$$Q \circ P = \big(q_k[\sigma_P]\big)_{k \in K}.$$

Proposition 3.3 (Compositionality of source semantics)**.** *For source computations*

$$P : I \Rightarrow O \qquad \textit{and} \qquad Q : O \Rightarrow K,$$

$$\boxed{[\![Q \circ P]\!]_{\mathsf{Logic}} = [\![Q]\!]_{\mathsf{Logic}} \circ [\![P]\!]_{\mathsf{Logic}}.}$$

Proof. Let $\rho : I \to \mathbb{B}$. For each $k \in K$, the substitution lemma gives

$$[\![Q \circ P]\!]_{\mathsf{Logic}}(\rho)(k) = [\![q_k[\sigma_P]]\!]_\rho = [\![q_k]\!]_{[\![P]\!]_{\mathsf{Logic}}(\rho)}.$$

The right-hand side is exactly

$$[\![Q]\!]_{\mathsf{Logic}}\big([\![P]\!]_{\mathsf{Logic}}(\rho)\big)(k).$$

Since this holds for every k, the output valuations are equal. □

The identity source computation on an interface I is

$$\mathrm{id}_I = (x)_{x \in I} : I \Rightarrow I.$$

Its semantics is the identity function:

$$[\![\mathrm{id}_I]\!]_{\mathsf{Logic}} = \mathrm{id}_{\mathbb{B}^I}.$$

Thus source composition behaves exactly as composition of Boolean functions.

3.6 Semantic Equivalence

The framework identifies source computations by observable Boolean behavior, not by syntactic identity.

For compatible source computations

$$P, Q : I \Rightarrow O,$$

write

$$P \equiv_{\mathsf{Logic}} Q$$

when

$$\boxed{\forall \rho : I \to \mathbb{B}, \qquad [\![P]\!]_{\mathsf{Logic}}(\rho) = [\![Q]\!]_{\mathsf{Logic}}(\rho).}$$

For example,

$$\neg(a \wedge b) \equiv_{\mathsf{Logic}} (\neg a) \vee (\neg b),$$

even though the two expressions have different syntax.

Likewise,

$$(a \wedge b) \vee (a \wedge c) \equiv_{\mathsf{Logic}} a \wedge (b \vee c).$$

Semantic equivalence therefore forgets expression structure while preserving the computation represented by that structure.

Proposition 3.4 (Source equivalence is compositional)**.** *Suppose*

$$P \equiv_{\mathsf{Logic}} P'$$

and

$$Q \equiv_{\mathsf{Logic}} Q'$$

for compatible interfaces. Then

$$\boxed{Q \circ P \equiv_{\mathsf{Logic}} Q' \circ P'.}$$

Proof. For every input valuation ρ, compositionality gives

$$[\![Q \circ P]\!]_{\mathsf{Logic}}(\rho) = [\![Q]\!]_{\mathsf{Logic}}\left([\![P]\!]_{\mathsf{Logic}}(\rho)\right).$$

Replace the inner denotation using $P \equiv_{\mathsf{Logic}} P'$ and the outer denotation using $Q \equiv_{\mathsf{Logic}} Q'$. The result is

$$[\![Q \circ P]\!]_{\mathsf{Logic}}(\rho) = [\![Q' \circ P']\!]_{\mathsf{Logic}}(\rho).$$

□

This congruence is the source-level form of a principle used throughout the paper: a semantically equivalent component may be substituted into a larger computation without changing its observable meaning.

3.7 Source Semantics as the Reference Meaning

The source language is intentionally more abstract than the representations that follow it.

A term such as

$$(a \wedge b) \vee c$$

specifies a Boolean function. It does not specify whether the implementation uses AND and OR gates, NAND gates, shared subexpressions, duplicated logic, particular CMOS cells, or any physical transistor geometry.

Those choices are introduced by lowering.

Consequently, the correctness criterion for the first representation boundary is not structural similarity between the source term and its gate network. It is equality of denotation:

$$\boxed{[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}}(\rho) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

Section 4 defines the gate representation and constructs a logic-to-gate lowering satisfying this contract.

The source layer therefore fixes what must remain invariant throughout the subsequent semantic descent:

$$\boxed{\text{source meaning} = \text{Boolean denotation.}}$$

4 Gate-Network Algebra

The source language of Section 3 specifies Boolean computations without committing to an implementation structure. The gate layer introduces the first explicit computational network.

A gate network makes dependencies, fan-out, sharing, and a particular decomposition into primitive Boolean operations explicit while remaining independent of transistor realization.

The purpose of this section is to define that representation, give it a compositional Boolean semantics, construct the logic-to-gate lowering

$$\mathcal{L}_G : \mathsf{Logic} \to \mathsf{Gate},$$

and prove that the lowering preserves the source denotation.

4.1 Acyclic Gate Networks

Let I and O be finite input and output interfaces.

Definition 4.1 (Gate network)**.** *A finite acyclic gate network*

$$G : I \Rightarrow O$$

consists of:

(i) a finite sequence of internal gate nodes

$$v_1, \ldots, v_N$$

listed in topological order;

(ii) for each v_i, an arity $k_i \geq 0$ and an ordered predecessor tuple

$$\mathrm{pred}(v_i) \in (I \cup \{v_1, \ldots, v_{i-1}\})^{k_i};$$

(iii) for each v_i, a local Boolean function

$$f_{v_i} : \mathbb{B}^{k_i} \to \mathbb{B};$$

(iv) an output map

$$\mathrm{out}_G : O \to I \cup \{v_1, \ldots, v_N\}.$$

The topological-order condition guarantees that every internal node depends only on primary inputs or previously evaluated nodes.

The same wire may occur more than once in predecessor tuples. Consequently, fan-out is represented directly. Likewise, one internal node may feed several later nodes or several designated outputs, so sharing is represented without duplicating the corresponding computation.

A network with no internal gates is permitted. Such a network can represent identity wiring, renaming, permutation, duplication of an input at the output interface, or omission of unused inputs.

4.2 Primitive Gate Basis

The exact Boolean semantics of a gate network does not require a unique primitive basis. For the lowering developed here, however, it is useful to choose a basis that is directly compatible with the complementary-CMOS construction of Section 6.

We use the primitive functions

$$\mathsf{CONST}_0, \qquad \mathsf{CONST}_1, \qquad \mathsf{NOT}, \qquad \mathsf{NAND}.$$

The constant gates have arity zero and satisfy

$$\mathsf{CONST}_0() = 0, \qquad \mathsf{CONST}_1() = 1.$$

The unary inverter satisfies

$$\mathsf{NOT}(x) = 1 - x,$$

and the two-input NAND gate satisfies

$$\mathsf{NAND}(x, y) = 1 - (x \wedge y).$$

This basis is functionally complete. In particular,

$$x \wedge y = \mathsf{NOT}\big(\mathsf{NAND}(x, y)\big),$$

and, by De Morgan duality,

$$x \vee y = \mathsf{NAND}\big(\mathsf{NOT}(x), \mathsf{NOT}(y)\big).$$

The choice is also compatible with the transistor lowering developed later. Each primitive is the complement of a monotone Boolean kernel: constants are complements of constant kernels, NOT is the complement of a single variable, and NAND is the complement of a conjunction. Thus every gate introduced by the canonical logic-to-gate lowering lies in the local gate family supported by Section 6.

4.3 Gate-Network Semantics

Let

$$G : I \Rightarrow O$$

be an acyclic gate network and let

$$\rho : I \to \mathbb{B}$$

be a primary-input valuation.

The valuation is extended uniquely to the internal gate nodes by topological evaluation.

For a primary input $x \in I$, define

$$\widehat{\rho}_G(x) = \rho(x).$$

For each internal node v_i with predecessor tuple

$$\mathrm{pred}(v_i) = (w_{i,1}, \ldots, w_{i,k_i}),$$

define

$$\widehat{\rho}_G(v_i) = f_{v_i}\left(\widehat{\rho}_G(w_{i,1}), \ldots, \widehat{\rho}_G(w_{i,k_i})\right).$$

Because every predecessor of v_i occurs earlier in the topological order, this recursion is well defined.

Definition 4.2 (Gate-network denotation)**.** *The denotation of*

$$G : I \Rightarrow O$$

is the function

$$[\![G]\!]_{\mathsf{Gate}} : \mathbb{B}^I \to \mathbb{B}^O$$

defined by

$$\boxed{[\![G]\!]_{\mathsf{Gate}}(\rho)(o) = \widehat{\rho}_G\left(\mathrm{out}_G(o)\right)}$$

for every $o \in O$*.*

Thus the semantics of a gate network is determined by its externally observable Boolean behavior, while its internal graph retains the structural information required by later compiler stages.

4.4 Network Composition

Gate networks compose by connecting output wires of one network to the input interface of another.

Let

$$G : I \Rightarrow O$$

and

$$H : O \Rightarrow K.$$

After renaming internal nodes so that the two internal node sets are disjoint, form

$$H \circ G : I \Rightarrow K$$

by replacing each occurrence of a primary input $o \in O$ in H by the wire

$$\mathrm{out}_G(o).$$

The internal nodes of G are placed before the internal nodes of H in the resulting topological order.

Proposition 4.3 (Compositionality of gate semantics)**.** *For compatible finite acyclic gate networks*

$$G : I \Rightarrow O \qquad \textit{and} \qquad H : O \Rightarrow K,$$

$$\boxed{[\![H \circ G]\!]_{\mathsf{Gate}} = [\![H]\!]_{\mathsf{Gate}} \circ [\![G]\!]_{\mathsf{Gate}}.}$$

Proof. Let $\rho : I \to \mathbb{B}$.

The internal nodes of G are evaluated first, producing the output valuation

$$[\![G]\!]_{\mathsf{Gate}}(\rho) : O \to \mathbb{B}.$$

Every occurrence of an input o of H has been connected to the corresponding output wire of G. Hence the internal nodes of H receive exactly the valuation

$$[\![G]\!]_{\mathsf{Gate}}(\rho).$$

Topological evaluation of the remaining nodes therefore produces

$$[\![H]\!]_{\mathsf{Gate}}\left([\![G]\!]_{\mathsf{Gate}}(\rho)\right).$$

This is the denotation of the composed network. □

Parallel composition is obtained by taking disjoint unions of internal nodes while preserving a common or disjoint primary-input interface as appropriate. Because each component is evaluated independently except where wires are explicitly shared, its denotation is the corresponding product of component denotations.

The same formalism therefore supports serial composition, parallel composition, fan-out, and sharing without identifying any of them with transistor-path composition. The distinction becomes important in Section 6.

4.5 Syntax-Directed Logic-to-Gate Lowering

We now define a canonical lowering from the source expressions of Section 3 into the primitive gate basis above.

For an expression e over an input interface I, let

$$\Gamma_I(e)$$

denote a finite acyclic gate network with input interface I and one designated output.

The construction is recursive.

Variables. For

$$e = x, \qquad x \in I,$$

no internal gate is introduced. The designated output is simply the primary input wire x.

Constants. For

$$e = 0,$$

introduce one zero-input node carrying CONST_0.

For

$$e = 1,$$

introduce one zero-input node carrying CONST_1.

Negation. For

$$e = \neg e_1,$$

first construct

$$\Gamma_I(e_1)$$

and append one NOT node whose predecessor is the designated output of $\Gamma_I(e_1)$.

Conjunction. For

$$e = e_1 \wedge e_2,$$

construct

$$\Gamma_I(e_1) \qquad \text{and} \qquad \Gamma_I(e_2),$$

renaming internal nodes apart while identifying the common primary-input interface.

Append a NAND node with the two subnetwork outputs as predecessors, followed by a NOT node.

Thus the final two gates compute

$$\mathsf{NOT}\,(\mathsf{NAND}(u, v)) = u \wedge v.$$

Disjunction. For

$$e = e_1 \vee e_2,$$

construct the two subnetworks as above.

Append a NOT node to each subnetwork output and feed those two inverted values into a NAND node.

The resulting output is

$$\mathsf{NAND}\,(\mathsf{NOT}(u), \mathsf{NOT}(v)) = u \vee v.$$

This construction intentionally defines one canonical correct lowering rather than an optimized one. Repeated source subexpressions may initially be duplicated. Sharing, common-subexpression elimination, basis conversion, and other restructuring are compiler transformations whose admissibility is treated separately in Section 8.

4.6 Correctness of Expression Lowering

Theorem 4.4 (Expression-to-gate correctness)**.** *For every source expression e over a finite input interface I and every valuation*

$$\rho : I \to \mathbb{B},$$

the designated output of $\Gamma_I(e)$ satisfies

$$\boxed{[\![\Gamma_I(e)]\!]_{\mathsf{Gate}}(\rho) = [\![e]\!]_\rho.}$$

Here the single-output valuation on the left is identified with its Boolean component.

Proof. By structural induction on e.

For a variable x, the designated output of $\Gamma_I(x)$ is the primary input wire itself, so

$$[\![\Gamma_I(x)]\!]_{\mathsf{Gate}}(\rho) = \rho(x) = [\![x]\!]_\rho.$$

For the constants, the unique gate node produces the corresponding constant, hence

$$[\![\Gamma_I(0)]\!]_{\mathsf{Gate}}(\rho) = 0$$

and

$$[\![\Gamma_I(1)]\!]_{\mathsf{Gate}}(\rho) = 1.$$

Suppose the result holds for e_1. For negation, the appended inverter gives

$$\begin{aligned} [\![\Gamma_I(\neg e_1)]\!]_{\mathsf{Gate}}(\rho) &= 1 - [\![\Gamma_I(e_1)]\!]_{\mathsf{Gate}}(\rho) \\ &= 1 - [\![e_1]\!]_\rho \\ &= [\![\neg e_1]\!]_\rho. \end{aligned}$$

Suppose the result holds for e_1 and e_2. Let

$$u = [\![e_1]\!]_\rho, \qquad v = [\![e_2]\!]_\rho.$$

For conjunction, the final NAND and inverter compute

$$\begin{aligned} \mathsf{NOT}(\mathsf{NAND}(u, v)) &= 1 - \big(1 - (u \wedge v)\big) \\ &= u \wedge v \\ &= [\![e_1 \wedge e_2]\!]_\rho. \end{aligned}$$

For disjunction, the final gates compute

$$\begin{aligned}\mathsf{NAND}\,(\mathsf{NOT}(u), \mathsf{NOT}(v)) &= 1 - \big((1-u) \wedge (1-v)\big) \\ &= u \vee v \\ &= [\![e_1 \vee e_2]\!]_\rho.\end{aligned}$$

Thus every constructor preserves the source denotation. □

4.7 Lowering Complete Source Computations

Let

$$P = (e_o)_{o \in O} : I \Rightarrow O$$

be a source computation.

For each output $o \in O$, construct

$$\Gamma_I(e_o).$$

Rename their internal nodes apart, identify their common primary-input interface I, and designate the output of $\Gamma_I(e_o)$ as the gate-network output corresponding to o.

This defines

$$\boxed{\mathcal{L}_G : \mathsf{Logic} \to \mathsf{Gate}.}$$

The construction may subsequently be transformed to introduce sharing or a different gate decomposition, but the canonical lowering itself is completely syntax directed.

Theorem 4.5 (Logic-to-Gate Semantic Preservation)**.** *Let*

$$P : I \Rightarrow O$$

be a source computation. Then for every valuation

$$\rho : I \to \mathbb{B},$$

$$\boxed{[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}}(\rho) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

Proof. Write

$$P = (e_o)_{o \in O}.$$

By construction, the gate-network output associated with o is the designated output of

$$\Gamma_I(e_o).$$

By Theorem 4.4,

$$[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}}(\rho)(o) = [\![e_o]\!]_\rho.$$

By the definition of source semantics,

$$[\![e_o]\!]_\rho = [\![P]\!]_{\mathsf{Logic}}(\rho)(o).$$

The equality holds for every $o \in O$, so the two output valuations are equal. □

This theorem discharges the first lowering obligation required by the end-to-end semantic descent.

4.8 Compatibility with Gate-to-CMOS Lowering

The canonical lowering above produces only four kinds of local gates:

$$\mathsf{CONST}_0, \qquad \mathsf{CONST}_1, \qquad \mathsf{NOT}, \qquad \mathsf{NAND}.$$

Each is the negation of a monotone Boolean kernel.

In the notation introduced formally in Section 6,

$$\mathsf{CONST}_1 = g_{\bot}, \qquad \mathsf{CONST}_0 = g_{\top},$$

$$\mathsf{NOT}(x) = g_x,$$

and

$$\mathsf{NAND}(x, y) = g_{x \wedge y}.$$

Consequently, every internal node generated by $\mathcal{L}_G$ belongs to the supported local domain of the complementary gate-to-CMOS lowering developed in Section 6.

Corollary 4.6 (Compatibility of the discrete lowering domains)**.** *For every finite source computation P, the canonical gate network*

$$\mathcal{L}_G(P)$$

is finite and acyclic, and every one of its internal nodes admits the complementary static-CMOS lowering defined in Section 6.

Proof. Finiteness and acyclicity follow by structural induction from the construction of Γ_I. Every newly introduced node is one of the four primitive gates listed above, each of which is a negative-monotone gate supported by the gate-to-CMOS construction. □

Thus the two discrete lowering stages are not merely independently correct; their domains are compatible:

$$\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS}.$$

Combining Theorem 4.5 with the gate-to-CMOS preservation theorem developed in Section 6 yields the exact logic-to-CMOS result established in Section 7.

The semantic role of the gate layer can therefore be summarized as

$$\boxed{\text{Boolean meaning } + \text{ explicit signal-flow structure,}}$$

with the Boolean meaning preserved exactly by the logic-to-gate lowering.

5 CMOS Transistor-Network Algebra

This section introduces the first exact semantic layer of the framework. The key distinction is between a transistor network and the Boolean function that may ultimately be abstracted from it. At switch level, a transistor network is first a controlled conduction structure. Boolean meaning appears only after that structure satisfies an appropriate static-CMOS validity condition.

5.1 Series–Parallel Conduction Terms

6 Gate-to-CMOS Lowering

The transistor algebra of the preceding section gives a correctness criterion for individual complementary CMOS cells. The next problem is compilation: given a Boolean gate or a network of gates, construct CMOS cells whose rail semantics realizes the same computation, and show that local correctness is preserved when those cells are connected by signal-flow wiring.

A crucial distinction is maintained throughout this section. The operations $\otimes$ and $\oplus$ compose transistor conduction paths *inside* a cell. Gate-network composition connects the output signal of one valid cell to control terminals of other cells. These are different forms of composition and are not identified.

6.1 Monotone Kernels and NMOS Realization

The pull-down network of a complementary static-CMOS cell is naturally a positive, and therefore monotone, Boolean object. This motivates the following source language for cell synthesis.

Definition 6.1 (Monotone kernel)**.** *Fix a finite control set X. The language $\mathcal{M}_X$ of monotone Boolean kernels is generated by*

$$M ::= \bot \mid \top \mid x \mid (M \wedge M) \mid (M \vee M), \qquad x \in X.$$

Its denotation under a valuation $\rho : X \to \mathbb{B}$ is the usual Boolean interpretation, written

$$[\![M]\!]_{\mathcal{M}}(\rho) \in \mathbb{B}.$$

Definition 6.2 (NMOS realization)**.** *Define*

$$\nu_X : \mathcal{M}_X \to \mathcal{N}_X$$

recursively by

$$\nu_X(\bot) = \mathbf{0}, \qquad \nu_X(\top) = \mathbf{1}, \qquad \nu_X(x) = n_x,$$

and

$$\nu_X(M \wedge N) = \nu_X(M) \otimes \nu_X(N),$$
$$\nu_X(M \vee N) = \nu_X(M) \oplus \nu_X(N).$$

Lemma 6.3 (NMOS realization correctness)**.** *For every $M \in \mathcal{M}_X$ and valuation $\rho : X \to \mathbb{B}$,*

$$\boxed{\chi_\rho(\nu_X(M)) = [\![M]\!]_{\mathcal{M}}(\rho).}$$

Proof. By structural induction on M. The constant and variable cases follow from the definition of χ_ρ. For conjunction,

$$\begin{aligned}\chi_\rho(\nu_X(M \wedge N)) &= \chi_\rho(\nu_X(M) \otimes \nu_X(N)) \\ &= \chi_\rho(\nu_X(M)) \wedge \chi_\rho(\nu_X(N)) \\ &= [\![M]\!]_{\mathcal{M}}(\rho) \wedge [\![N]\!]_{\mathcal{M}}(\rho) \\ &= [\![M \wedge N]\!]_{\mathcal{M}}(\rho).\end{aligned}$$

The disjunction case is identical with $\oplus$ and $\vee$ in place of $\otimes$ and $\wedge$. □

6.2 Cell-Level Lowering

A single complementary static-CMOS cell obtained from the preceding algebra computes the negation of its NMOS conduction function. Accordingly, the natural primitive gate class is the class of negated monotone kernels.

Definition 6.4 (Negative-monotone gate)**.** *For $M \in \mathcal{M}_X$, let g_M denote the Boolean gate*

$$g_M : \mathbb{B}^X \to \mathbb{B}, \qquad g_M(\rho) = 1 - [\![M]\!]_{\mathcal{M}}(\rho).$$

Define its gate-to-CMOS lowering by

$$\boxed{\mathcal{L}_C(g_M) = C[\nu_X(M)] = \big(\nu_X(M)^\star, \nu_X(M)\big).}$$

Theorem 6.5 (Local Gate-to-CMOS Correctness)**.** *For every $M \in \mathcal{M}_X$, the lowered cell $\mathcal{L}_C(g_M)$ is valid under every valuation. Moreover, for every $\rho : X \to \mathbb{B}$,*

$$\boxed{\alpha_C\big(R_{\mathcal{L}_C(g_M)}(\rho)\big) = g_M(\rho).}$$

Thus the Boolean abstraction of the CMOS rail semantics is exactly the gate denotation.

Proof. By **??**, the cell $C[\nu_X(M)]$ is valid and satisfies

$$\alpha_C\big(R_{C[\nu_X(M)]}(\rho)\big) = 1 - \chi_\rho(\nu_X(M)).$$

Applying Lemma 6.3 yields

$$1 - \chi_\rho(\nu_X(M)) = 1 - [\![M]\!]_{\mathcal{M}}(\rho) = g_M(\rho).$$

□

Corollary 6.6 (Primitive CMOS lowering rules)**.** *For Boolean controls $x, y \in X$,*

$$\mathcal{L}_C(\neg x) = (p_x, n_x),$$

$$\mathcal{L}_C(\mathrm{NAND}(x, y)) = (p_x \oplus p_y,\ n_x \otimes n_y),$$

and

$$\mathcal{L}_C(\mathrm{NOR}(x, y)) = (p_x \otimes p_y,\ n_x \oplus n_y).$$

Each lowered cell is globally valid in the ideal switch semantics and computes the indicated Boolean function after application of α_C.

Proof. Use the kernels x, $x \wedge y$, and $x \vee y$ respectively in Theorem 6.5. □

More generally, the same construction lowers every gate whose Boolean function is the negation of a monotone kernel. This includes arbitrary fan-in NAND and NOR cells and the usual AOI/OAI families expressible in that form. The restriction belongs to a *single complementary cell*; it does not restrict the Boolean expressivity of networks of such cells.

6.3 Boolean Semantics of Valid Cells

For purposes of cell composition, it is convenient to name the Boolean function exposed by a globally valid CMOS cell.

Definition 6.7 (Abstract Boolean behavior)**.** *Let C be a static CMOS cell over controls X that is valid under every valuation. Its abstract Boolean behavior is*

$$\beta_C : \mathbb{B}^X \to \mathbb{B}, \qquad \beta_C(\rho) = \alpha_C(R_C(\rho)).$$

Because C is globally valid, β_C is total.

The map $C \mapsto \beta_C$ deliberately forgets transistor topology while preserving digital behavior. Consequently, distinct transistor realizations may have the same β_C even though they differ physically.

6.4 Signal-Flow Composition

CMOS cells interact digitally by driving transistor *control* terminals of downstream cells. At the present abstraction level, gate terminals are assumed electrically isolating: connecting a valid upstream output to a downstream control changes the downstream control valuation but does not create a conduction path through the upstream output node. Loading and nonideal gate behavior are deferred to the physical interpretation layer developed later.

Definition 6.8 (Ideal cascade)**.** *Let C_f be a globally valid cell with control set X, and let C_g be a globally valid cell with control set $Y \cup \{z\}$, where $z \notin Y$. The ideal cascade obtained by connecting the output of C_f to the control z of C_g is written*

$$C_g \circ_z C_f.$$

Its Boolean behavior is defined by signal substitution:

$$\beta_{C_g \circ_z C_f}(\rho_X, \rho_Y) = \beta_{C_g}\big(\rho_Y[z \mapsto \beta_{C_f}(\rho_X)]\big).$$

Proposition 6.9 (Cascade preservation)**.** *Suppose globally valid cells C_f and C_g implement Boolean functions $f : \mathbb{B}^X \to \mathbb{B}$ and $g : \mathbb{B}^{Y \cup \{z\}} \to \mathbb{B}$, respectively. Then their ideal cascade implements the Boolean substitution of f into the z-input of g:*

$$\boxed{\beta_{C_g \circ_z C_f}(\rho_X, \rho_Y) = g\big(\rho_Y[z \mapsto f(\rho_X)]\big).}$$

Proof. Substitute $\beta_{C_f} = f$ and $\beta_{C_g} = g$ into the defining equation for ideal cascade. □

This proposition is simple by design: once local cells are known to be valid, digital composition should reduce to ordinary substitution. Its importance is that the difficult transistor-level obligation has already been discharged locally by complementary-CMOS correctness.

6.5 Acyclic Cell Networks

The cascade rule extends from two cells to finite feed-forward networks.

Definition 6.10 (Acyclic gate network)**.** *An acyclic gate network consists of a finite set I of primary input wires, a finite sequence of gate nodes*

$$v_1, \ldots, v_N$$

in topological order, and an ordered predecessor tuple

$$\mathrm{pred}(v_i) \subseteq I \cup \{v_1, \ldots, v_{i-1}\}$$

for each v_i. If k_i is the length of this tuple, node v_i carries a local Boolean function

$$f_{v_i} : \mathbb{B}^{k_i} \to \mathbb{B}.$$

A finite ordered list of wires in $I \cup \{v_1, \ldots, v_N\}$ is designated as the network output interface.

For a primary-input valuation $\rho : I \to \mathbb{B}$, the gate semantics $[\![G]\!]_{\mathsf{Gate}}(\rho)$ is obtained by evaluating the nodes in topological order and then reading the designated output wires.

Definition 6.11 (Cellwise CMOS lowering)**.** *Let G be an acyclic gate network. Suppose that for every node v_i a globally valid CMOS cell C_{v_i} has been chosen such that*

$$\beta_{C_{v_i}} = f_{v_i}.$$

The cellwise lowering $\mathcal{L}_C(G)$ is obtained by replacing each gate node by its chosen CMOS cell and preserving the signal-flow wiring and output interface.

Theorem 6.12 (Local-to-Global Gate-to-CMOS Preservation)**.** *Let G be a finite acyclic gate network and let $\mathcal{L}_C(G)$ be a cellwise lowering satisfying*

$$\beta_{C_{v_i}} = f_{v_i}$$

for every node v_i. Then every internal cell output is valid under every primary-input valuation, and the abstract Boolean output of the lowered CMOS network agrees with the gate-network semantics:

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho).}$$

Here α_C^* denotes componentwise application of α_C to the designated output rail states.

Proof. Proceed by induction over the topological order $v_1, \ldots, v_N$. Every input to v_1 is a primary input, so the chosen cell C_{v_1} receives exactly the same Boolean valuation as the gate v_1. Since C_{v_1} is globally valid and $\beta_{C_{v_1}} = f_{v_1}$, its output is valid and abstracts to the gate output.

Assume the claim holds for $v_1, \ldots, v_{i-1}$. Each predecessor of v_i is either a primary input or the output of an earlier node. By the induction hypothesis, every earlier cell output is valid and its Boolean abstraction is identical to the corresponding gate-wire value. Hence C_{v_i} receives the same Boolean input tuple as v_i. Global validity of C_{v_i} gives a valid rail output, and $\beta_{C_{v_i}} = f_{v_i}$ makes its abstract value equal to the gate output. The invariant therefore holds at v_i.

After N steps, every designated output wire has the same Boolean value in the gate network and in the abstracted CMOS network. Componentwise application of α_C gives the displayed equality. □

Corollary 6.13 (Correctness of synthesized negative-monotone networks)**.** *If every node of an acyclic gate network is a negative-monotone gate g_M and each node is lowered by*

$$\mathcal{L}_C(g_M) = C[\nu_X(M)],$$

then the resulting CMOS cell network preserves the Boolean denotation of the original gate network.

Proof. Local correctness for every node follows from Theorem 6.5; apply Theorem 6.12. □

6.6 Functional Completeness of Cell Networks

The cell-level restriction to negated monotone kernels does not impose a restriction on complete network expressivity.

Corollary 6.14 (Functional completeness through NAND)**.** *Every Boolean function with finitely many inputs has an acyclic network of valid complementary static-CMOS cells whose abstract Boolean behavior realizes that function.*

Proof. The two-input NAND gate is functionally complete, so every finite Boolean function has a finite acyclic NAND-gate realization. By Corollary 6.6, every NAND node has a globally valid complementary CMOS implementation

$$(p_x \oplus p_y,\ n_x \otimes n_y).$$

Applying Theorem 6.12 to the resulting cellwise lowering preserves the Boolean denotation of the entire NAND network. □

This result separates two levels of expressivity that are easy to conflate. Within a single complementary cell, the pull-down conduction function is monotone and the cell computes its complement. Across a network of valid cells, signal-flow composition recovers arbitrary finite Boolean computation.

6.7 The Gate-to-CMOS Semantic Contract

The preceding results justify the gate-to-CMOS contract anticipated earlier in the paper. For every acyclic gate network in the supported lowering domain,

$$\boxed{\alpha_C^* \circ [\![\mathcal{L}_C(-)]\!]_{\mathsf{CMOS}} = [\![-]\!]_{\mathsf{Gate}}.}$$

Equivalently, the following diagram commutes at the ideal digital abstraction level:

$$\begin{array}{ccc} \mathsf{Gate} & \xrightarrow{\mathcal{L}_C} & \mathsf{CMOS} \\ \downarrow [\![\cdot]\!]_{\mathsf{Gate}} & & \downarrow [\![\cdot]\!]_{\mathsf{CMOS}} \\ \text{Boolean behavior} & \xleftarrow{\alpha_C^*} & \text{rail behavior.} \end{array}$$

The theorem is intentionally independent of transistor count, delay, loading, area, power, and process technology. Those quantities can distinguish multiple correct implementations without changing the exact Boolean semantic contract. This separation permits subsequent compiler passes to optimize among semantically equivalent CMOS realizations while preserving the proof obligation established here.

7 End-to-End Semantic Preservation

The preceding section establishes correctness at the gate-to-CMOS boundary. The purpose of this section is to show that such local contracts compose with an independently correct logic-to-gate lowering. The resulting statement is an end-to-end preservation theorem: a source computation and its lowered CMOS realization have the same Boolean meaning after the CMOS rail semantics is abstracted back to the logical level.

The argument is deliberately modular. No proof about transistor structure is repeated here. Instead, correctness is propagated through representation boundaries by composing the semantic contracts already associated with the individual lowering steps.

7.1 Composable Semantic Contracts

We first isolate the general principle used by the lowering pipeline.

Definition 7.1 (Semantic lowering contract)**.** *Let A and B be representation spaces with semantic domains S_A and S_B, denotational maps*

$$[\![-]\!]_A : A \to S_A, \qquad [\![-]\!]_B : B \to S_B,$$

a lowering map

$$L_{A\to B} : A \to B,$$

and an abstraction map

$$\alpha_B : S_B \rightharpoonup S_A.$$

The lowering $L_{A\to B}$ satisfies the semantic lowering contract when α_B is defined on the semantics of every supported lowered object and

$$\boxed{\alpha_B([\![L_{A\to B}(a)]\!]_B) = [\![a]\!]_A}$$

for every supported $a \in A$.

The definition permits the target semantics to retain distinctions that are absent from the source. Correctness requires only that those distinctions be forgotten by the designated abstraction map without changing the source meaning.

Theorem 7.2 (Composition of Semantic Lowering Contracts)**.** *Let*

$$A \xrightarrow{L_{A\to B}} B \xrightarrow{L_{B\to C}} C$$

be two lowering steps with semantic domains S_A, S_B, S_C and abstraction maps

$$\alpha_B : S_B \rightharpoonup S_A, \qquad \alpha_C : S_C \rightharpoonup S_B.$$

Suppose

$$\alpha_B([\![L_{A\to B}(a)]\!]_B) = [\![a]\!]_A$$

and

$$\alpha_C([\![L_{B\to C}(b)]\!]_C) = [\![b]\!]_B$$

for all supported $a \in A$ and $b \in B$. Then the composite lowering satisfies

$$\boxed{(\alpha_B \circ \alpha_C)([\![L_{B\to C}(L_{A\to B}(a))]\!]_C) = [\![a]\!]_A.}$$

Proof. For supported $a \in A$, apply the $B \to C$ contract to $b = L_{A\to B}(a)$ and then apply the $A \to B$ contract:

$$\begin{aligned}(\alpha_B \circ \alpha_C)([\![L_{B\to C}(L_{A\to B}(a))]\!]_C) &= \alpha_B([\![L_{A\to B}(a)]\!]_B) \\ &= [\![a]\!]_A.\end{aligned}$$

□

Corollary 7.3 (Finite semantic descent)**.** *Any finite sequence of lowering steps satisfying compatible semantic lowering contracts preserves the semantics of its initial representation after composition of the corresponding abstraction maps.*

Proof. Induction on the number of lowering steps using Theorem 7.2. □

This is the structural reason that compiler correctness can be established one representation boundary at a time. Once the contracts are compatible, an end-to-end theorem is a consequence of composition rather than a new proof about every implementation detail simultaneously.

7.2 The Logic-to-Gate Contract

Let $P \in \mathsf{Logic}$ have primary-input interface I and designated Boolean output interface O. The logic-to-gate lowering

$$\mathcal{L}_G : \mathsf{Logic} \to \mathsf{Gate}$$

is required to preserve that interface and satisfy, for every input valuation $\rho : I \to \mathbb{B}$,

$$\boxed{[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}}(\rho) = [\![P]\!]_{\mathsf{Logic}}(\rho).} \tag{LG}$$

The exact proof of this contract depends on the source syntax and gate-network construction fixed earlier in the paper. The present section uses only the contract itself: the internal form of the source term is irrelevant once its gate realization has been shown to preserve denotation.

For the second boundary, Theorem 6.12 supplies the already established contract

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho)} \tag{GC}$$

for every supported finite acyclic gate network G. Because every designated CMOS output is valid under the hypotheses of that theorem, α_C^* is total on the outputs produced by the lowering.

7.3 Logic-to-CMOS Preservation

We can now state the principal exact correctness theorem for the discrete portion of the semantic descent.

Theorem 7.4 (End-to-End Logic-to-CMOS Preservation)**.** *Let $P \in \mathsf{Logic}$ be a source computation with input interface I. Suppose:*

(i) the logic-to-gate lowering satisfies the contract (LG)*;*

(ii) $G = \mathcal{L}_G(P)$ is a finite acyclic gate network in the supported domain of the cellwise CMOS lowering; and

(iii) every gate node of G is lowered to a globally valid CMOS cell with the same local Boolean behavior.

Then for every primary-input valuation $\rho : I \to \mathbb{B}$, every internal output of the lowered CMOS network is valid and

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

Proof. Let $G = \mathcal{L}_G(P)$. By hypotheses (ii) and (iii), Theorem 6.12 gives validity of every internal cell output and

$$\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho).$$

By hypothesis (i),

$$[\![G]\!]_{\mathsf{Gate}}(\rho) = [\![P]\!]_{\mathsf{Logic}}(\rho).$$

Substitution yields the claimed equality. □

Thus the exact digital lowering pipeline satisfies

$$\boxed{\alpha_C^* \circ [\![\mathcal{L}_C(\mathcal{L}_G(-))]\!]_{\mathsf{CMOS}} = [\![-]\!]_{\mathsf{Logic}}}$$

on its supported domain. Pointwise in each input valuation, the following semantic diagram commutes:

$$\begin{array}{ccccc} \mathsf{Logic} & \xrightarrow{\mathcal{L}_G} & \mathsf{Gate} & \xrightarrow{\mathcal{L}_C} & \mathsf{CMOS} \\ \downarrow[\![\cdot]\!]_{\mathsf{Logic}} & & \downarrow[\![\cdot]\!]_{\mathsf{Gate}} & & \downarrow[\![\cdot]\!]_{\mathsf{CMOS}} \\ \text{Boolean behavior} & = & \text{Boolean behavior} & \xleftarrow{\alpha_C^*} & \text{rail behavior.} \end{array}$$

The significance of the theorem is not the final equality alone. Its proof factorizes the obligation into independently checkable compiler boundaries. A logic compiler need not reason about transistor topology, and a CMOS cell lowering need not reconstruct the formal source syntax. They interact through an explicit semantic contract.

7.4 No Semantic Drift Under Correct Lowering

It is useful to state the preceding result as an invariant of the lowering pipeline.

Corollary 7.5 (No semantic drift)**.** *Under the hypotheses of Theorem 7.4, the Boolean value of every designated output is invariant under the descent*

$$P \longmapsto \mathcal{L}_G(P) \longmapsto \mathcal{L}_C(\mathcal{L}_G(P)) \longmapsto \alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}})\,.$$

In particular, representational change alone cannot alter the denotation of a supported computation.

Proof. Immediate from Theorem 7.4. □

This statement is intentionally semantic rather than syntactic. The gate network may introduce auxiliary nodes, duplicate or share subcomputations, and the CMOS realization may replace each logical operation by a structurally very different transistor network. None of those changes matters to exact logical correctness provided the contracts remain satisfied.

7.5 Representation Independence

Correctness should not depend on selecting one privileged gate network or one privileged transistor realization. We therefore distinguish implementation identity from semantic identity.

Definition 7.6 (Source semantic equivalence)**.** *For source computations $P, Q \in \mathsf{Logic}$ with compatible interfaces, define*

$$P \equiv_{\mathsf{Logic}} Q$$

when

$$[\![P]\!]_{\mathsf{Logic}}(\rho) = [\![Q]\!]_{\mathsf{Logic}}(\rho)$$

for every input valuation ρ.

Definition 7.7 (Abstract CMOS equivalence)**.** *Let C and D be CMOS cell networks with compatible input and output interfaces whose designated outputs are valid under every valuation. Define*

$$C \equiv_{\alpha_C} D$$

when

$$\alpha_C^*([\![C]\!]_{\mathsf{CMOS}}(\rho)) = \alpha_C^*([\![D]\!]_{\mathsf{CMOS}}(\rho))$$

for every primary-input valuation ρ.

The relation $\equiv_{\alpha_C}$ is intentionally coarser than transistor-level identity. It classifies networks by their observable Boolean behavior after rail abstraction and therefore permits different transistor topologies to represent the same logical computation.

Theorem 7.8 (Representation Independence of Correct CMOS Realizations)**.** *Let $P, Q \in \mathsf{Logic}$ satisfy*

$$P \equiv_{\mathsf{Logic}} Q.$$

Suppose both are lowered through pipelines satisfying the hypotheses of Theorem 7.4, yielding CMOS networks

$$C_P = \mathcal{L}_C(\mathcal{L}_G(P)), \qquad C_Q = \mathcal{L}_C(\mathcal{L}_G(Q)).$$

Then

$$\boxed{C_P \equiv_{\alpha_C} C_Q.}$$

Proof. For every input valuation ρ, end-to-end correctness gives

$$\alpha_C^*([\![C_P]\!]_{\mathsf{CMOS}}(\rho)) = [\![P]\!]_{\mathsf{Logic}}(\rho)$$

and

$$\alpha_C^*([\![C_Q]\!]_{\mathsf{CMOS}}(\rho)) = [\![Q]\!]_{\mathsf{Logic}}(\rho).$$

The right-hand sides are equal because $P \equiv_{\mathsf{Logic}} Q$. Hence the two CMOS networks are abstractly equivalent. □

Corollary 7.9 (Implementation independence)**.** *Let two possibly different correct lowering pipelines compile the same source computation P to valid CMOS networks C_1 and C_2. If both pipelines satisfy the logic-to-gate and gate-to-CMOS semantic contracts, then*

$$\boxed{C_1 \equiv_{\alpha_C} C_2.}$$

Thus exact logical correctness does not determine a unique gate structure or a unique transistor structure.

Proof. Apply Theorem 7.8 with $Q = P$. □

This is the semantic freedom required by optimization. A compiler may choose among multiple gate decompositions and multiple CMOS realizations so long as it remains within the same abstract-equivalence class. Section 8 develops that observation into explicit semantics-preserving circuit transformations and separates correctness from implementation cost.

7.6 Extension Toward Physical Realization

The contract-composition theorem also explains how the physical layer can be added without changing the discrete correctness proof. If a future physical realization map

$$\mathcal{L}_P : \mathsf{CMOS} \to \mathsf{Phys}$$

satisfies an electrical abstraction contract

$$\alpha_P([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}) = [\![C]\!]_{\mathsf{CMOS}}$$

under explicit physical assumptions, then Theorem 7.2 composes that contract with the exact logic-to-CMOS theorem. For a supported source computation P, the resulting obligation has the form

$$\boxed{\alpha_C^*(\alpha_P([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}})) = [\![P]\!]_{\mathsf{Logic}}.}$$

Unlike the ideal digital contracts proved above, the physical contract will require explicit assumptions concerning admissible voltage regions, timing, loading, switching, and device behavior. Keeping that contract separate is what allows exact logical preservation and nonideal physical realization to be reasoned about without conflating their semantic regimes.

8 Circuit Transformations

Correct lowering establishes that a source computation may be represented by many different gate and CMOS structures without changing its abstract Boolean meaning. A compiler must therefore do more than lower representations: it must also transform them while preserving the semantic contracts already proved. This section isolates that transformation principle and separates two questions that must not be conflated:

(i) whether a rewrite preserves meaning; and

(ii) whether the rewritten realization is preferable according to a cost model.

The first question is exact and semantic. The second may depend on transistor count, depth, delay, area, power, loading, switching activity, or technology.

8.1 Semantic Equivalence at the Transformation Layers

Section 7 defined source equivalence and abstract CMOS equivalence. We now add the corresponding gate-level relation and a stronger rail-level relation for CMOS networks.

Definition 8.1 (Gate semantic equivalence)**.** *Let G and H be gate networks with compatible primary-input and designated output interfaces. Define*

$$G \equiv_{\mathsf{Gate}} H$$

when, for every primary-input valuation ρ,

$$[\![G]\!]_{\mathsf{Gate}}(\rho) = [\![H]\!]_{\mathsf{Gate}}(\rho).$$

Definition 8.2 (Rail equivalence)**.** *Let C and D be CMOS networks with compatible interfaces. Write*

$$C \equiv_R D$$

when their designated rail-state outputs agree exactly under every primary-input valuation:

$$[\![C]\!]_{\mathsf{CMOS}}(\rho) = [\![D]\!]_{\mathsf{CMOS}}(\rho).$$

No Boolean abstraction is applied in this definition.

Rail equivalence is stronger than the abstract CMOS equivalence $\equiv_{\alpha_C}$ introduced in Section 7. It preserves the exact ideal rail-state observation, whereas $\equiv_{\alpha_C}$ preserves only the Boolean meaning exposed after abstraction.

Proposition 8.3 (Rail equivalence implies abstract equivalence)**.** *Let C and D be CMOS networks whose designated outputs are valid under every valuation. If*

$$C \equiv_R D,$$

then

$$\boxed{C \equiv_{\alpha_C} D.}$$

Proof. For each valuation ρ, rail equivalence gives identical designated rail-state tuples. Since the outputs are valid, componentwise application of α_C is defined on both tuples and yields identical Boolean outputs. □

The converse need not hold in a richer CMOS semantics: two realizations may have the same Boolean abstraction while differing in internal topology or in non-output electrical structure. The distinction between exact rail identity and abstract Boolean identity therefore gives the compiler more than one useful equivalence relation.

8.2 Certified Rewrite Relations

Definition 8.4 (Semantics-preserving gate rewrite)**.** *A gate rewrite relation*

$$G \rightsquigarrow_{\mathsf{Gate}} H$$

is semantics preserving *when every admissible rewrite instance has compatible interfaces and satisfies*

$$G \equiv_{\mathsf{Gate}} H.$$

Definition 8.5 (Semantics-preserving CMOS rewrite)**.** *A CMOS rewrite relation*

$$C \rightsquigarrow_{\mathsf{CMOS}} D$$

is abstractly semantics preserving *when every admissible rewrite instance preserves interface compatibility and satisfies*

$$C \equiv_{\alpha_C} D.$$

It is rail preserving *when the stronger relation*

$$C \equiv_R D$$

holds.

These definitions deliberately describe correctness independently of the mechanism that discovers or applies the rewrite. A compiler pass may use a pattern rule, canonical form, theorem prover, equality saturation procedure, or technology-specific optimization search. Certification is determined by the semantic relation, not by the search strategy.

Theorem 8.6 (Finite rewrite preservation)**.** *Let*

$$X_0 \rightsquigarrow X_1 \rightsquigarrow \cdots \rightsquigarrow X_n$$

be a finite sequence of rewrites at one representation layer, and suppose each rewrite preserves the designated semantic equivalence relation at that layer. Then

$$X_0 \equiv X_n.$$

Proof. Each semantic equivalence relation used here is transitive because it is defined by equality of denotations under all valuations. Repeated transitivity across the finite rewrite sequence gives the result. □

Thus a compiler may compose individually certified transformations without reproving the entire optimization pipeline from first principles.

8.3 Congruence of Local Cell Replacement

A central compiler operation is replacement of one cell implementation by another implementation of the same local Boolean function. The following result shows that local equivalence lifts to global network equivalence in the acyclic signal-flow semantics of Section 6.

Theorem 8.7 (Local cell replacement congruence)**.** *Let N be a finite acyclic CMOS cell network obtained by cellwise lowering of an acyclic gate network. Let a node v of N contain a globally valid cell C with abstract Boolean behavior β_C. Form N' by replacing C with a globally valid cell D having the same input interface and satisfying*

$$\beta_D = \beta_C,$$

while leaving all other cells and all signal-flow wiring unchanged. Then

$$\boxed{N \equiv_{\alpha_C} N'.}$$

Moreover every internal output of N' remains valid.

Proof. Evaluate the network in a topological order. All nodes preceding v are unchanged, so they produce the same valid Boolean abstractions in N and N'. Hence C and D receive the same Boolean input tuple at v. Since $\beta_C = \beta_D$, their valid outputs have the same Boolean abstraction. Every downstream cell therefore receives exactly the same Boolean control valuation in the two networks. Induction over the remaining topological order shows that every downstream abstract output is identical and remains valid. Thus the designated outputs of N and N' are abstractly equivalent. □

Corollary 8.8 (Repeated local replacement)**.** *Any finite sequence of globally valid cell replacements preserving local abstract Boolean behavior preserves the abstract Boolean behavior of the entire acyclic cell network.*

Proof. Apply Theorem 8.7 repeatedly and use transitivity of $\equiv_{\alpha_C}$. □

This is the basic contextuality result required for library substitution and technology mapping at the ideal digital layer.

8.4 Certified Rewrites of Conduction Networks

The conduction algebra of Section 5 supplies a stronger source of certified CMOS transformations. Because conduction equivalence is defined pointwise, it is a congruence for the series and parallel constructors.

Proposition 8.9 (Conduction congruence)**.** *If*

$$K_1 \equiv_c L_1 \qquad \text{and} \qquad K_2 \equiv_c L_2,$$

then

$$K_1 \otimes K_2 \equiv_c L_1 \otimes L_2$$

and

$$K_1 \oplus K_2 \equiv_c L_1 \oplus L_2.$$

Furthermore,

$$K_1^\star \equiv_c L_1^\star.$$

Proof. For every valuation ρ, conduction equivalence gives $\chi_\rho(K_i) = \chi_\rho(L_i)$ for $i = 1, 2$. Applying conjunction and disjunction preserves equality, proving the first two claims. For duality, **??** gives

$$\chi_\rho(K_1^\star) = 1 - \chi_\rho(K_1) = 1 - \chi_\rho(L_1) = \chi_\rho(L_1^\star).$$

□

The complementary-cell construction therefore transports conduction rewrites into exact rail-preserving CMOS rewrites.

Theorem 8.10 (Conduction-equivalent cells are rail equivalent)**.** *Let $K, L \in \mathcal{N}_X$ satisfy*

$$K \equiv_c L.$$

Then their complementary cells

$$C[K] = (K^\star, K), \qquad C[L] = (L^\star, L)$$

are globally valid and satisfy

$$\boxed{C[K] \equiv_R C[L].}$$

Consequently,

$$C[K] \equiv_{\alpha_C} C[L].$$

Proof. Global validity of both cells follows from **??**. Since $K \equiv_c L$, their pull-down networks have identical conduction values under every valuation. By Proposition 8.9, $K^\star \equiv_c L^\star$, so their pull-up networks also have identical conduction values. The definition of rail semantics therefore gives

$$R_{C[K]}(\rho) = R_{C[L]}(\rho)$$

for every ρ. The abstract-equivalence conclusion follows from Proposition 8.3. □

This theorem turns every equation of the conduction lattice into a certified ideal CMOS rewrite. For example,

$$K \otimes (K \oplus L) \equiv_c K$$

by absorption, and

$$K \otimes (L \oplus M) \equiv_c (K \otimes L) \oplus (K \otimes M)$$

by distributivity. Either side may therefore be used as the pull-down network of a complementary cell without changing its exact ideal rail behavior.

8.5 Canonical Conduction Normalization

The minimal-support theorem of Section 5 gives a canonical representative for an NMOS conduction equivalence class.

Definition 8.11 (Minimal-support normalizer)**.** *For $K \in \mathcal{N}_X$, define*

$$\mathsf{NF}(K) = \bigoplus_{S \in \mathrm{Min}(K)} \bigotimes_{x \in S} n_x.$$

The conventions for empty series and parallel combinations are those of **??**.

Corollary 8.12 (Certified canonicalization)**.** *For every $K \in \mathcal{N}_X$,*

$$K \equiv_c \mathsf{NF}(K),$$

and therefore

$$\boxed{C[K] \equiv_R C[\mathsf{NF}(K)].}$$

Moreover,

$$K \equiv_c L \quad \Longleftrightarrow \quad \mathsf{NF}(K) = \mathsf{NF}(L)$$

up to the fixed ordering convention used to print the finite supports and their members.

Proof. The first claim is **??**; the cell-level claim then follows from Theorem 8.10. The final equivalence is **??** together with the definition of NF. □

Canonicalization is useful for equivalence checking and proof production, but it is not automatically an optimization. A canonical disjunctive form may increase transistor count or depth. The normal form is therefore a semantic representative, not a claim about physical optimality.

8.6 Gate-Level Rewrites and Lowering Independence

The gate layer admits its own semantics-preserving transformations, including Boolean identities, basis changes, decomposition, reassociation, and common subexpression restructuring when the relevant wiring semantics is respected. Correct CMOS lowering makes such rewrites safe across the representation boundary.

Theorem 8.13 (Gate rewrite lifting)**.** *Let G and H be compatible finite acyclic gate networks such that*

$$G \equiv_{\mathsf{Gate}} H.$$

Suppose both networks are lowered by correct cellwise CMOS lowerings satisfying Theorem 6.12. Then

$$\boxed{\mathcal{L}_C(G) \equiv_{\alpha_C} \mathcal{L}_C(H).}$$

Proof. For every primary-input valuation ρ, Theorem 6.12 gives

$$\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho)$$

and

$$\alpha_C^*([\![\mathcal{L}_C(H)]\!]_{\mathsf{CMOS}}(\rho)) = [\![H]\!]_{\mathsf{Gate}}(\rho).$$

The right-hand sides are equal because $G \equiv_{\mathsf{Gate}} H$. □

Thus an optimizer may transform the gate network before CMOS lowering without changing the eventual logical meaning, provided the gate rewrite itself is certified and the subsequent CMOS lowering remains correct.

Corollary 8.14 (Basis conversion is semantically admissible)**.** *Any finite gate-basis conversion that preserves the Boolean denotation of a finite acyclic network preserves the abstract Boolean denotation of its correct CMOS lowering.*

Proof. A denotation-preserving basis conversion produces a gate-equivalent network; apply Theorem 8.13. □

For example, an AND node may be represented by a NAND followed by an inverter, and an OR node may be represented by a NAND whose inputs are first inverted. The resulting structures differ syntactically and physically, but their correct lowerings remain members of the same abstract semantic class.

8.7 An Abstract Optimization Diamond

Gate-level and CMOS-level optimizations can occur at different points in a compiler pipeline. The following theorem records the semantic relation between those choices.

Theorem 8.15 (Abstract optimization diamond)**.** *Let T_G be a semantics-preserving transformation on a supported acyclic gate network G, so that*

$$T_G(G) \equiv_{\mathsf{Gate}} G.$$

Let T_C be an abstractly semantics-preserving CMOS transformation applicable to $\mathcal{L}_C(G)$, so that

$$T_C(\mathcal{L}_C(G)) \equiv_{\alpha_C} \mathcal{L}_C(G).$$

Assume G and $T_G(G)$ are both lowered correctly. Then

$$\boxed{\mathcal{L}_C(T_G(G)) \equiv_{\alpha_C} T_C(\mathcal{L}_C(G)).}$$

Proof. By Theorem 8.13,

$$\mathcal{L}_C(T_G(G)) \equiv_{\alpha_C} \mathcal{L}_C(G).$$

By hypothesis,

$$T_C(\mathcal{L}_C(G)) \equiv_{\alpha_C} \mathcal{L}_C(G).$$

Symmetry and transitivity of $\equiv_{\alpha_C}$ give the result. □

The theorem does not state that the two optimized circuits are structurally identical or have equal physical cost. It states only that certified optimization before and after the gate-to-CMOS boundary remains inside the same abstract semantic equivalence class.

8.8 Correctness and Cost

Semantic equivalence specifies which implementations are admissible replacements. A separate cost model determines which admissible replacement a compiler should prefer.

Definition 8.16 (Cost model)**.** *Let $\mathcal{D}$ be a set equipped with a preorder $\preceq_{\mathcal{D}}$. A CMOS cost model is a map*

$$\kappa : \mathsf{CMOS} \to \mathcal{D}.$$

Possible components of κ include transistor count, network depth, area proxy, delay estimate, fan-out burden, switching activity, or power estimates. No particular choice is fixed at the exact semantic layer.

Definition 8.17 (Admissible optimization set)**.** *For a valid CMOS network C, let*

$$\mathcal{A}(C) \subseteq \{D \in \mathsf{CMOS} : D \equiv_{\alpha_C} C\}$$

be a chosen set of compiler-generated candidate implementations. An element $D^\star \in \mathcal{A}(C)$ is cost-minimal in $\mathcal{A}(C)$ when

$$\kappa(D^\star) \preceq_{\mathcal{D}} \kappa(D)$$

for every $D \in \mathcal{A}(C)$ whenever such a least element exists.

Theorem 8.18 (Optimization soundness)**.** *Let C be a valid CMOS network and let $D^\star$ be any implementation chosen from $\mathcal{A}(C)$, including a cost-minimal implementation when one exists. Then*

$$\boxed{D^\star \equiv_{\alpha_C} C.}$$

Consequently, optimization over $\mathcal{A}(C)$ cannot change the abstract Boolean behavior of C.

Proof. By definition, every member of $\mathcal{A}(C)$ belongs to the abstract semantic equivalence class of C. The criterion used to select one member of that set does not alter membership in the class. □

Although elementary, the theorem expresses an important compiler architecture rule:

semantic admissibility first, cost selection second.

A cost improvement cannot certify correctness, and semantic equivalence alone cannot establish that an implementation is physically preferable.

8.9 Ideal Equivalence Versus Physical Optimization

The algebraic identities of the present section hold at the ideal switch and Boolean abstraction levels. They must not be interpreted as claims of physical equivalence. As observed in Section 5,

$$K \oplus K \equiv_c K,$$

yet two parallel transistor branches can differ physically from one branch in resistance, capacitance, area, current capability, loading, and switching behavior. Similarly, distributivity may preserve conduction while duplicating transistor structure.

This produces a deliberate hierarchy of compiler reasoning:

physical realization $\longrightarrow$ switch/conduction behavior $\longrightarrow$ rail behavior $\longrightarrow$ Boolean behavior.

Moving to the right forgets distinctions. A rewrite valid at a coarser level must therefore be reconsidered before being promoted to a claim at a finer physical level.

The role of the physical semantics developed later is precisely to supply the additional contract under which cost-sensitive implementation decisions can be related to voltages, timing, loading, and switching behavior without weakening the exact semantic-preservation theorems proved here.

9 Physical Interpretation

The preceding sections treat CMOS networks at an ideal switch level. A transistor is either conducting or nonconducting, supply rails are abstract symbols, and valid complementary cells produce exact rail states. Physical CMOS devices do not operate in this idealized domain. Voltages vary continuously, switching requires time, capacitive loads affect propagation, and device behavior depends on electrical and technological parameters.

The purpose of this section is not to introduce a transistor-device model or to derive circuit equations. Instead, it specifies the semantic interface between the exact CMOS model developed above and a more detailed physical realization. The physical layer is therefore treated as a refinement of the ideal CMOS semantics, together with an explicit abstraction contract stating when physical behavior implements the discrete model.

9.1 Voltage Regions

Let

$$\mathbb{V} = [0, V_{\mathrm{DD}}]$$

denote the admissible voltage interval for a fixed supply voltage $V_{\mathrm{DD}} > 0$. Choose thresholds

$$0 \le V_L < V_H \le V_{\mathrm{DD}}.$$

These determine the low and high voltage regions

$$\mathbb{V}_L = [0, V_L], \qquad \mathbb{V}_H = [V_H, V_{\mathrm{DD}}],$$

and the intermediate region

$$\mathbb{V}_U = (V_L, V_H).$$

The voltage abstraction is the partial map

$$\delta_V : \mathbb{V} \rightharpoonup \mathbb{B}$$

defined by

$$\delta_V(v) = \begin{cases} 0, & v \in \mathbb{V}_L, \\ 1, & v \in \mathbb{V}_H, \end{cases}$$

and undefined for $v \in \mathbb{V}_U$.

Thus logical values are represented by admissible voltage regions rather than by exact physical voltages. In particular, logical 0 does not mean that a node voltage must equal exactly 0, and logical 1 does not mean that it must equal exactly V_{DD}.

This partial abstraction deliberately retains the possibility that a physical state has no valid Boolean interpretation. Such states arise naturally during switching and may also indicate failure of the assumptions required for correct digital operation.

9.2 Physical Observations and Rail Abstraction

To relate physical behavior to the rail semantics of Section 5, it is useful to distinguish voltage from effective pull-up and pull-down drive.

For a single observed node, define a physical observation to be a triple

$$o = (v, u, d) \in \mathbb{V} \times \mathbb{B} \times \mathbb{B},$$

where v is the observed node voltage, u records whether an effective conducting path to the high supply is present, and d records whether an effective conducting path to the low supply is present.

Define the physical-to-rail abstraction

$$\alpha_P^{\text{rail}} : \mathbb{V} \times \mathbb{B} \times \mathbb{B} \longrightarrow \mathcal{P}(\{H, L\})$$

by

$$\alpha_P^{\text{rail}}(v, u, d) = \{H \mid u = 1\} \cup \{L \mid d = 1\}.$$

Hence

$$\begin{aligned} (u, d) = (1, 0) \quad &\Longmapsto \quad \{H\}, \\ (u, d) = (0, 1) \quad &\Longmapsto \quad \{L\}, \\ (u, d) = (0, 0) \quad &\Longmapsto \quad \varnothing, \end{aligned}$$

and

$$(u, d) = (1, 1) \quad \Longmapsto \quad \{H, L\}.$$

The voltage coordinate is retained because physical correctness requires more than connectivity alone. A node may have the intended rail connectivity but fail to settle into the voltage region required for a valid Boolean interpretation.

For m observed nodes, define

$$\mathsf{Obs}_m = (\mathbb{V} \times \mathbb{B} \times \mathbb{B})^m.$$

The abstraction α_P^{rail} extends componentwise to Obs_m.

9.3 Physical Traces

Physical execution is temporal. Let

$$\mathsf{Tr}_m = \{\sigma : \mathbb{R}_{\geq 0} \to \mathsf{Obs}_m\}$$

denote the set of physical traces over m observed nodes.

A physical realization may depend on a collection of electrical and technological parameters. Let Λ denote a parameter space containing whatever quantities are required by the chosen physical model, such as supply conditions, device characteristics, capacitive loads, interconnect properties, and environmental assumptions.

A physical circuit $P \in \mathsf{Phys}$ is assigned a possibly set-valued semantics

$$[\![P]\!]_{\mathsf{Phys}}(\rho, \lambda) \subseteq \mathsf{Tr}_m,$$

where ρ is an input valuation and $\lambda \in \Lambda$ is a physical parameter assignment.

The use of a set-valued semantics permits the framework to represent nondeterminism, parameter uncertainty, model abstraction, or multiple physically admissible trajectories without requiring the present theory to choose a particular differential-equation or device-level semantics.

9.4 Settling and Stable Observation

The ideal CMOS semantics is static, whereas physical behavior evolves through time. The bridge between them is therefore defined by stable observation after a settling interval.

For a trace $\sigma \in \mathsf{Tr}_m$ and a time $T \geq 0$, say that output i is *Boolean-stable after* T when there exists $b_i \in \mathbb{B}$ such that

$$\delta_V(v_i(t)) = b_i$$

for every $t \geq T$ in the observation interval under consideration.

For a finite observation horizon $\Delta > 0$, define the settled physical abstraction

$$\alpha_P^{[\Delta]}$$

whenever every designated output remains in a valid voltage region throughout the interval

$$[T, T + \Delta]$$

and its associated drive state agrees with the corresponding ideal rail state throughout that interval.

The abstraction is undefined when an output remains in the intermediate voltage region, changes Boolean region during the required stable interval, floats when a driven value is required, or simultaneously exhibits effective high and low drive.

For a physical realization P, input valuation ρ, parameter assignment λ, and designated output set O, define a settling-time quantity

$$T_{\mathrm{set}}(P, \rho, \lambda, O)$$

to be any sound upper bound after which all outputs in O satisfy the required stable-observation condition.

No claim is made here that such a bound exists for every physical circuit or every parameter assignment. Existence is part of the physical admissibility conditions.

9.5 Physical Admissibility

The discrete semantic theorems of the preceding sections are unconditional within their formal domains. Physical correctness necessarily depends on additional assumptions.

Let

$$\mathrm{Adm}_P(C, \rho, \lambda, \Delta)$$

be a physical admissibility predicate for a CMOS network C, input valuation ρ, parameter assignment λ, and required stable interval Δ.

Its concrete definition belongs to the selected physical backend, but may include conditions such as:

1. the supply voltage lies within the supported operating range;
2. input voltages lie in valid low or high regions;
3. device and interconnect parameters lie within the modeled domain;
4. capacitive loading and fan-out remain within certified bounds;
5. no forbidden persistent contention occurs;
6. every designated output settles within a certified time bound;

7. settled output voltages remain within their required valid regions for at least Δ;

8. environmental and process assumptions required by the physical model are satisfied.

The predicate separates two fundamentally different questions:

What does the ideal CMOS network mean?

and

Under what physical conditions does a realization exhibit that meaning?

This separation prevents physical assumptions from being hidden inside the exact Boolean semantics.

9.6 CMOS-to-Physical Realization

Let

$$\mathcal{L}_P : \mathsf{CMOS} \to \mathsf{Phys}$$

be a physical realization map. The map may represent technology mapping, transistor sizing, layout-dependent realization, extraction, or another backend-specific transformation that produces an object with physical semantics.

The required correctness condition is not equality between ideal CMOS objects and physical trajectories. These inhabit different semantic domains. Correctness is instead expressed through abstraction.

Definition 9.1 (Physical realization contract)**.** *A physical realization map $\mathcal{L}_P$ satisfies the physical realization contract for a CMOS network C when, for every supported input valuation ρ, physical parameter assignment λ, and stable interval Δ satisfying*

$$\mathrm{Adm}_P(C, \rho, \lambda, \Delta),$$

the settled physical semantics is defined and

$$\boxed{\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho, \lambda)\right) = [\![C]\!]_{\mathsf{CMOS}}(\rho).}$$

When the physical semantics is set-valued, the displayed equality is understood uniformly: every physically admissible trace represented by $[\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho, \lambda)$ must abstract to the same ideal CMOS rail behavior.

This requirement is essential. Correctness must not depend on selecting one favorable trajectory while admitting another trajectory that violates the ideal semantics.

Theorem 9.2 (Uniform physical correctness)**.** *Suppose $\mathcal{L}_P$ satisfies the physical realization contract for C under Adm_P. Then every physical execution admitted under those conditions has the same settled rail abstraction as the ideal CMOS semantics:*

$$\boxed{\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho, \lambda)\right) = [\![C]\!]_{\mathsf{CMOS}}(\rho).}$$

Proof. This is the defining obligation of the physical realization contract. The uniform interpretation of the set-valued physical semantics ensures that the equality holds for every admitted physical trace rather than merely for an existentially selected execution. □

9.7 Propagation Delay as a Compositional Bound

The semantic framework does not require physical transistor delay to be literally additive. Actual propagation depends on device characteristics, slew, loading, topology, interconnect, and other physical effects. What is required for compositional compiler reasoning is a sound local bound.

Suppose each realized CMOS cell C_i is assigned a certified local delay bound

$$d_i \geq 0$$

under an associated admissibility condition. For a directed signal path

$$\pi = (C_{i_1}, \ldots, C_{i_k}),$$

define the compositional path bound

$$D(\pi) = \sum_{j=1}^{k} d_{i_j}.$$

For a designated output o, let $\Pi(o)$ be the set of directed input-to-output paths terminating at o. Define

$$D_o = \max_{\pi \in \Pi(o)} D(\pi),$$

and for a network G with designated output set O define

$$D_G = \max_{o \in O} D_o.$$

These quantities are semantic certification bounds, not claims that physical delay is intrinsically additive.

Proposition 9.3 (Compositional settling bound)**.** *Suppose every cell in an acyclic realized network satisfies its certified local settling bound whenever its inputs have stabilized, and suppose the corresponding physical admissibility conditions hold. Then every designated output is stable no later than the compositional network bound D_G after the primary inputs stabilize.*

Proof. Proceed in a topological order. A cell whose predecessors have stabilized receives stable inputs and therefore settles within its certified local bound. Induction along a path gives the sum of the local bounds on that path. Taking the maximum over all input-to-output paths yields D_o for each output, and taking the maximum over designated outputs yields D_G. □

The proposition illustrates the intended role of physical models in the compiler. Detailed electrical analysis may produce the local certified bounds, while the semantic layer composes those bounds without pretending that the underlying device physics has become an additive algebra.

9.8 Loading and Fan-Out

Fan-out is semantically free at the Boolean level: one logical value may be used by multiple downstream gates without changing its denotation. Physical fan-out is not free. Additional loads can alter capacitance, transition time, noise margin, and ultimately the validity of the digital abstraction.

Accordingly, a Boolean or gate-level transformation that increases fan-out remains logically correct but need not remain physically admissible.

Let

$$L_i(C, \lambda)$$

denote a backend-defined load measure at node i, and let

$$L_i^{\max}$$

denote a certified admissible bound. A physical backend may therefore include conditions of the form

$$L_i(C, \lambda) \leq L_i^{\max}$$

inside Adm_P.

The exact definition of load is intentionally left to the physical model. The semantic point is structural:

logical fan-out preservation does not imply physical realizability.

Thus an optimization pass may preserve Boolean semantics while forcing a later physical pass to insert buffers, resize devices, restructure the network, or reject the realization.

9.9 End-to-End Logic-to-Physical Preservation

The physical realization contract composes directly with the exact semantic contracts of Sections 6 and 7.

Let $P_0 \in \mathsf{Logic}$ be a supported source computation. Its complete lowering pipeline is

$$P_0 \overset{\mathcal{L}_G}{\longmapsto} \mathcal{L}_G(P_0) \overset{\mathcal{L}_C}{\longmapsto} \mathcal{L}_C(\mathcal{L}_G(P_0)) \overset{\mathcal{L}_P}{\longmapsto} \mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P_0))).$$

Theorem 9.4 (End-to-End Logic-to-Physical Preservation)**.** *Suppose the logic-to-gate and gate-to-CMOS lowerings satisfy their exact semantic contracts, and suppose the physical realization satisfies the physical realization contract under*

$$\mathrm{Adm}_P\left(\mathcal{L}_C(\mathcal{L}_G(P_0)), \rho, \lambda, \Delta\right).$$

Then

$$\boxed{\alpha_C^*\Big(\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P_0)))]\!]_{\mathsf{Phys}}(\rho, \lambda)\right)\Big) = [\![P_0]\!]_{\mathsf{Logic}}(\rho).}$$

Proof. By the physical realization contract,

$$\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P_0)))]\!]_{\mathsf{Phys}}(\rho, \lambda)\right) = [\![\mathcal{L}_C(\mathcal{L}_G(P_0))]\!]_{\mathsf{CMOS}}(\rho).$$

Applying α_C^* and then using the end-to-end logic-to-CMOS preservation theorem gives

$$\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P_0))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P_0]\!]_{\mathsf{Logic}}(\rho).$$

Composition yields the result. □

The theorem completes the semantic descent

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys.}}$$

Each movement to the right introduces representation distinctions absent from the preceding layer. Correctness is recovered by abstraction in the opposite direction. The resulting architecture permits exact semantic claims at the logical and ideal CMOS layers while making every physical assumption explicit at the boundary where it is actually required.

9.10 Semantic Equivalence Does Not Imply Physical Equivalence

Two physical realizations may implement exactly the same Boolean computation while differing in transistor dimensions, layout, capacitance, resistance, delay, energy consumption, leakage, noise behavior, area, and reliability. Likewise, two CMOS networks that are rail equivalent under the ideal semantics need not produce identical physical traces.

The hierarchy developed throughout the paper therefore culminates in the distinction

$$\boxed{\text{Boolean equivalence} \not\Rightarrow \text{physical equivalence.}}$$

This is not a weakness of the abstraction. It is precisely what makes the abstraction useful. The semantic layers forget implementation distinctions that are irrelevant to logical meaning while preserving them at the physical level for optimization and feasibility analysis.

Consequently, the future CMOS Silicon Compiler may first establish semantic admissibility using the exact algebraic contracts developed in this paper and then select among admissible physical realizations according to backend criteria such as delay, area, loading, power, or technology constraints. The correctness question and the physical optimization question remain related but formally distinct.

10 Sequential Logic and State

The semantic development of the preceding sections is intentionally centered on finite acyclic combinational networks. This restriction permits the meaning of a circuit to be defined directly as a function of its current inputs and permits correctness to be propagated through a topological ordering of the network.

Sequential circuits require an additional semantic ingredient: state. Feedback, latches, flip-flops, registers, and clocked networks cannot in general be interpreted solely as functions

$$\mathbb{B}^I \to \mathbb{B}^O,$$

because their outputs may depend on both the current input and the history of the computation.

The purpose of this section is to identify the extension required to incorporate such systems without developing a complete sequential-CMOS theory.

10.1 State-Transition Semantics

Let I, S, and O denote finite sets of input, state, and output wires. A deterministic sequential component may be represented abstractly by a pair of functions

$$\delta : \mathbb{B}^I \times \mathbb{B}^S \longrightarrow \mathbb{B}^S$$

and

$$\omega : \mathbb{B}^I \times \mathbb{B}^S \longrightarrow \mathbb{B}^O,$$

where δ is the state-transition function and ω is the output function.

For an input sequence

$$\rho_0, \rho_1, \rho_2, \dots$$

and initial state s_0, execution is determined recursively by

$$s_{t+1} = \delta(\rho_t, s_t), \qquad o_t = \omega(\rho_t, s_t).$$

Thus the denotation of a sequential circuit is naturally a transformation of input traces and initial states rather than a single Boolean function.

10.2 Sequential Semantic Preservation

The lowering architecture developed earlier extends to sequential systems by requiring preservation of state-transition behavior.

Suppose a source sequential object P has transition semantics (δ_P, ω_P) and a lowered representation $L(P)$ has transition semantics (δ_L, ω_L). Let

$$\alpha_S : S_L \to S_P$$

be an abstraction from target states to source states, together with the appropriate abstraction α_O on outputs.

A sequential lowering contract requires that corresponding states remain related after every transition:

$$\boxed{\alpha_S(\delta_L(\rho, s_L)) = \delta_P(\rho, \alpha_S(s_L))}$$

and that observations agree:

$$\boxed{\alpha_O(\omega_L(\rho, s_L)) = \omega_P(\rho, \alpha_S(s_L))\,.}$$

Given related initial states, induction on execution steps then yields preservation of the complete abstract output trace.

Proposition 10.1 (Finite-trace preservation)**.** *Suppose a sequential lowering satisfies the transition and output contracts above, and suppose the initial states satisfy*

$$\alpha_S(s_0^L) = s_0^P\,.$$

Then for every finite input trace

$$\rho_0, \ldots, \rho_n,$$

the abstracted lowered execution and the source execution have identical source-level state and output traces through step n.

Proof. The claim follows by induction on n. The initial states are related by hypothesis. If the states at step t are related by α_S, the transition contract establishes the relation at step $t+1$, while the output contract establishes equality of the abstract observations at step t. Repeated application proves the result for every finite prefix. □

10.3 Feedback and Physical Time

At the gate and CMOS levels, feedback introduces issues absent from the acyclic theory. A feedback loop cannot in general be evaluated by the topological argument used for combinational networks. Its meaning depends on an explicit temporal or state interpretation.

At the physical layer, further obligations arise. Clock periods, setup and hold requirements, propagation delay, metastability, initialization, and feedback stability may determine whether a physically realized sequential network correctly implements its abstract transition system.

Accordingly, Boolean equivalence of combinational subcircuits is not by itself a sufficient correctness criterion for arbitrary sequential replacement. A transformation must also preserve the relevant state-transition and temporal contracts.

The physical admissibility framework of Section 9 provides the appropriate place for such conditions. A sequential backend may strengthen Adm_P with clocking and timing requirements and prove that each physical transition realizes the corresponding abstract state transition.

10.4 Scope of the Present Paper

A complete treatment of sequential CMOS would require substantially more machinery than is needed for the present theory. In particular, it would require a precise model of storage elements, feedback, initialization, clocking, temporal composition, and the relationship between continuous physical evolution and discrete state transitions.

Those developments are therefore left to an extension of the framework.

The important point for the present paper is that sequential computation does not require abandoning the semantic architecture already established. It requires enriching the objects being preserved:

$$\boxed{\text{combinational denotation} \quad \longrightarrow \quad \text{state-transition denotation.}}$$

The same general correctness principle remains in force. A lowering is correct when the distinctions introduced by the target representation can be abstracted away without changing the observable meaning of the source computation.

11 Compiler Architecture

The formal development of the preceding sections determines the architecture of a future *CMOS Silicon Compiler*. The compiler is not merely a translator from Boolean expressions to transistor diagrams. It is a sequence of representation changes, each associated with an explicit semantic preservation obligation.

The principal compilation pipeline is

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys}.}$$

The first two lowering boundaries admit exact discrete correctness theorems. The final boundary is governed by the physical realization contract of Section 9 and therefore depends on explicit admissibility assumptions.

This separation suggests a compiler architecture in which semantic correctness is established independently of backend-specific optimization and physical feasibility.

11.1 Intermediate Representations

Each semantic layer induces a corresponding intermediate representation.

A compiler implementation may therefore distinguish at least four principal representations:

$$\mathsf{IR}_{\mathsf{Logic}}, \qquad \mathsf{IR}_{\mathsf{Gate}}, \qquad \mathsf{IR}_{\mathsf{CMOS}}, \qquad \mathsf{IR}_{\mathsf{Phys}}.$$

The logical representation contains the source computation together with its typed or otherwise well-formed interface. Its semantics is the source denotation

$$[\![-]\!]_{\mathsf{Logic}}.$$

The gate representation makes Boolean structure, signal-flow composition, fan-out, and sharing explicit. Its semantics is

$$[\![-]\!]_{\mathsf{Gate}}.$$

The CMOS representation replaces supported gate nodes by complementary pull-up and pull-down transistor networks. It records transistor polarity, control signals, series and parallel conduction structure, supply rails, cell interfaces, and inter-cell signal flow. Its ideal semantics is

$$[\![-]\!]_{\mathsf{CMOS}}.$$

The physical representation introduces distinctions required by a concrete realization backend, potentially including device parameters, loading, timing information, technology-library choices, layout-dependent quantities, or another electrical model. Its semantics is

$$[\![-]\!]_{\mathsf{Phys}}.$$

These representations should not be collapsed into a single universal data structure merely because they describe the same eventual computation. Each representation exposes distinctions relevant to one level of reasoning while deliberately hiding others.

The compiler architecture therefore mirrors the semantic architecture:

$$\boxed{\text{representation boundary} \quad \longleftrightarrow \quad \text{proof boundary.}}$$

11.2 Verified Lowering Passes

A lowering pass transforms an object in one representation into an object in a more concrete representation.

For a generic pass

$$L_{A\to B} : A \to B,$$

correctness is established by an abstraction map

$$\alpha_B : S_B \rightharpoonup S_A$$

satisfying

$$\boxed{\alpha_B([\![L_{A\to B}(a)]\!]_B) = [\![a]\!]_A.}$$

Accordingly, the compiler should associate every semantics-changing lowering pass with an explicit contract rather than relying on the correctness of the complete pipeline as an indivisible implementation claim.

For the architecture developed here, the principal obligations are

$$[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}} = [\![P]\!]_{\mathsf{Logic}},$$

$$\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}) = [\![G]\!]_{\mathsf{Gate}},$$

and, under physical admissibility conditions,

$$\alpha_P^{[\Delta]}([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}) = [\![C]\!]_{\mathsf{CMOS}}.$$

The composition theorem of Section 7 then converts local pass correctness into end-to-end correctness.

This architecture has an important engineering consequence: a new backend or intermediate lowering stage need not require reproving the complete compiler. It must instead satisfy the semantic contract at the representation boundary where it is introduced.

11.3 Normalization and Certified Transformation

Lowering is not the only operation performed by the compiler. Each intermediate representation may admit semantics-preserving transformations before the next lowering step.

At the logical and gate levels, these may include Boolean simplification, basis conversion, reassociation, decomposition, common-subexpression restructuring, or other transformations compatible with the relevant wiring semantics.

At the CMOS level, Section 8 supplies certified transformations based on conduction equivalence. In particular, the canonical normalizer

$$\mathsf{NF}(K) = \bigoplus_{S \in \mathrm{Min}(K)} \bigotimes_{x \in S} n_x$$

satisfies

$$K \equiv_c \mathsf{NF}(K)$$

and therefore

$$C[K] \equiv_R C[\mathsf{NF}(K)].$$

A compiler may consequently organize transformations into two logically distinct stages:

generate semantically admissible candidates

followed by

select among candidates according to a cost model.

The ordering is essential. As established in Section 8,

semantic admissibility first, cost selection second.

A transformation is not certified merely because it reduces transistor count, depth, estimated delay, or another implementation cost.

11.4 Proof-Producing Passes

The semantic contracts suggest a proof-producing compiler architecture.

A compiler pass may return not merely a transformed object y, but a pair

$$(y, \pi),$$

where π is evidence that the transformation satisfies the appropriate semantic relation.

Abstractly, a verified pass has the form

$$\mathsf{Pass}(x) = (y, \pi),$$

with

$$\pi : \alpha([\![y]\!]_B) = [\![x]\!]_A$$

for a lowering pass, or

$$\pi : [\![x]\!]_A = [\![y]\!]_A$$

for a semantics-preserving transformation within one representation.

The proof object need not have a single implementation form. Depending on the eventual compiler, it may be represented by a theorem-prover term, a proof certificate checked by a smaller trusted kernel, a verified decision procedure, or another independently checkable witness.

The architectural principle is that correctness evidence should be attached to the transformation that creates the proof obligation.

This permits the trusted computing base to be smaller than the complete optimization and search machinery. Candidate generation may be complex, heuristic, or backend-specific while a comparatively small checker validates the semantic certificate associated with the selected result.

11.5 Technology Mapping

The abstract CMOS algebra intentionally does not commit to a fabrication process, standard-cell library, transistor geometry, voltage specification, or device model. Technology mapping therefore belongs after exact CMOS semantics has been established.

Let

$$\Theta$$

denote a technology description containing the parameters and implementation choices required by a physical backend. A technology-mapping pass may be written schematically as

$$\mathsf{Map}_\Theta : \mathsf{IR}_{\mathsf{CMOS}} \longrightarrow \mathsf{IR}_{\mathsf{Phys}}.$$

Different technology descriptions may map the same ideal CMOS network to different physical realizations.

Correctness does not require those realizations to be physically identical. It requires each realization to satisfy the physical contract under its declared admissibility conditions.

Thus, for admissible λ,

$$\alpha_P^{[\Delta]}([\![\mathsf{Map}_\Theta(C)]\!]_{\mathsf{Phys}}(\rho,\lambda)) = [\![C]\!]_{\mathsf{CMOS}}(\rho).$$

Technology mapping can therefore optimize quantities invisible to the ideal Boolean semantics, including device sizing, loading, buffering, timing, energy, area, or other backend-defined costs, while remaining constrained by the semantic and physical correctness contracts.

11.6 Validation and Physical Certification

The physical backend requires a different form of validation from the exact discrete stages.

For a candidate physical realization P, the compiler must determine whether the relevant admissibility predicate

$$\mathrm{Adm}_P(C,\rho,\lambda,\Delta)$$

has been established over the intended operating domain.

This validation may rely on static bounds, characterized technology data, electrical analysis, simulation, external verification tools, or combinations of these methods. The present theory does not prescribe one physical solver.

The compiler should therefore distinguish

semantic verification

from

physical certification.

Semantic verification establishes that a transformation preserves the designated abstract meaning. Physical certification establishes that a particular realization remains inside the domain in which the abstraction contract is valid.

Failure of physical certification does not imply that the source computation is semantically ill-defined. It means that the proposed physical realization has not been shown to implement that computation under the required physical conditions.

11.7 Compiler Pass Structure

A concrete compiler derived from the framework may therefore use a pass structure of the following form:

$$
\begin{aligned}
P &\longmapsto \mathsf{Parse}(P) \\
&\longmapsto \mathsf{Check}_{\mathsf{Logic}}(P) \\
&\longmapsto \mathsf{Normalize}_{\mathsf{Logic}}(P) \\
&\xmapsto{\mathcal{L}_G} G \\
&\longmapsto \mathsf{Optimize}_{\mathsf{Gate}}(G) \\
&\xmapsto{\mathcal{L}_C} C \\
&\longmapsto \mathsf{Normalize}_{\mathsf{CMOS}}(C) \\
&\longmapsto \mathsf{Optimize}_{\mathsf{CMOS}}(C) \\
&\xmapsto{\mathcal{L}_P} P_{\mathrm{phys}} \\
&\longmapsto \mathsf{Certify}_{\mathsf{Phys}}(P_{\mathrm{phys}}).
\end{aligned}
$$

Each arrow is associated either with a well-formedness obligation, an exact semantic-preservation obligation, or a physical admissibility obligation.

The pipeline need not be implemented literally as this linear sequence. Practical compilers may use repeated optimization passes, feedback between physical analysis and earlier representations, alternative candidate generation, or multiple backend paths. What must remain invariant is the semantic contract associated with every accepted transformation.

11.8 Prototype Targets

The formal architecture does not require the first implementation to perform full physical design. A useful prototype can terminate at several different levels.

A first backend may emit an explicit CMOS transistor-network representation whose cells can be checked directly against the conduction algebra of Section 5. Such a representation would already permit testing of logic-to-gate lowering, gate-to-CMOS lowering, canonicalization, transistor count, network depth, and semantic equivalence.

A subsequent backend may emit a conventional transistor-level netlist suitable for external electrical simulation. This would permit the physical contracts of Section 9 to be tested against voltage and timing traces without requiring the compiler itself to implement a device simulator.

Additional backends could target hardware-description or circuit-exchange formats where doing so preserves the distinctions required by the formal model.

The essential requirement is therefore not a particular output syntax. It is that every backend make explicit the representation boundary at which the formal guarantees end and additional backend assumptions begin.

11.9 Compiler Correctness Architecture

The resulting compiler has two complementary correctness regimes.

The discrete portion

$$\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS}$$

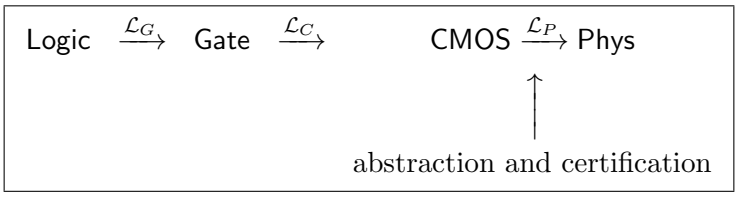
is governed by exact semantic preservation.

The physical portion

$$\mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys}$$

is governed by conditional refinement under explicit physical admissibility.

Together they yield the architecture



whose end-to-end correctness criterion is the preservation theorem established in Section 9.

The future CMOS Silicon Compiler is therefore specified not merely by the representations it can generate, but by the semantic evidence accompanying their generation. Its central invariant is:

every accepted compilation step preserves the meaning appropriate to its abstraction boundary.

This invariant converts the mathematical framework of the present paper into an executable compiler specification.

12 Worked Examples

The preceding sections develop the semantic framework abstractly. This section illustrates the complete construction on representative circuits. The first three examples recover the standard CMOS inverter, NAND, and NOR structures directly from the conduction algebra. The final example shows how a nontrivial Boolean computation is lowered compositionally through the gate and CMOS representations.

The examples are intentionally evaluated at the ideal CMOS level. Their physical realizations are governed by the additional admissibility conditions and abstraction contract of Section 9.

12.1 CMOS Inverter

Consider the Boolean function

$$f(x) = \neg x.$$

At the gate level, this is represented by a single inverter. In the negative-monotone formulation of Section 6, take the monotone kernel

$$M(x) = x.$$

Then

$$f(x) = 1 - M(x).$$

The NMOS realization of the kernel is

$$\nu_{\{x\}}(M) = n_x.$$

Its CMOS dual is

$$n_x^\star = p_x.$$

The complementary CMOS construction therefore gives

$$\boxed{C_{\mathrm{INV}} = (p_x, n_x).}$$

The pull-up network consists of one PMOS transistor controlled by x, and the pull-down network consists of one NMOS transistor controlled by x.

For $x = 0$,

$$\chi_0(p_x) = 1, \qquad \chi_0(n_x) = 0,$$

so

$$R_{C_{\mathrm{INV}}}(0) = \{H\}.$$

Hence

$$\alpha_C(R_{C_{\mathrm{INV}}}(0)) = 1.$$

For $x = 1$,

$$\chi_1(p_x) = 0, \qquad \chi_1(n_x) = 1,$$

so

$$R_{C_{\mathrm{INV}}}(1) = \{L\},$$

and therefore

$$\alpha_C(R_{C_{\mathrm{INV}}}(1)) = 0.$$

Thus

$$\boxed{\alpha_C(R_{C_{\mathrm{INV}}}(x)) = \neg x.}$$

The truth table is

x	PUN conducts	PDN conducts	output
0	1	0	1
1	0	1	0

This is the smallest instance of the complementary construction

$$C[K] = (K^\star, K)$$

proved correct in Section 5.

12.2 Two-Input NAND

Consider

$$f_{\mathrm{NAND}}(x, y) = \neg(x \wedge y).$$

Choose the monotone kernel

$$M(x, y) = x \wedge y.$$

Its NMOS conduction realization is

$$\nu_{\{x,y\}}(M) = n_x \otimes n_y.$$

Because series composition represents conjunction,

$$\chi_\rho(n_x \otimes n_y) = \rho(x) \wedge \rho(y).$$

Applying CMOS duality gives

$$(n_x \otimes n_y)^\star = p_x \oplus p_y.$$

The lowered complementary cell is therefore

$$\boxed{C_{\mathrm{NAND}} = (p_x \oplus p_y,\, n_x \otimes n_y)\,.}$$

The pull-down network contains two NMOS devices in series, while the pull-up network contains two PMOS devices in parallel.

Its rail behavior is summarized by

x	y	PUN conducts	PDN conducts	output
0	0	1	0	1
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Hence

$$\boxed{\alpha_C(R_{C_{\mathrm{NAND}}}(x, y)) = \neg(x \wedge y).}$$

This example also exhibits the direct relationship between Boolean conjunction and series NMOS conduction:

$$x \wedge y \quad \longleftrightarrow \quad n_x \otimes n_y.$$

Complementary dualization then constructs the pull-up network automatically.

12.3 Two-Input NOR

Now consider

$$f_{\mathrm{NOR}}(x, y) = \neg(x \vee y).$$

The monotone kernel is

$$M(x, y) = x \vee y.$$

Its NMOS realization is

$$\nu_{\{x,y\}}(M) = n_x \oplus n_y,$$

because parallel conduction represents disjunction:

$$\chi_\rho(n_x \oplus n_y) = \rho(x) \vee \rho(y).$$

Dualization gives

$$(n_x \oplus n_y)^\star = p_x \otimes p_y.$$

Thus

$$\boxed{C_{\mathrm{NOR}} = (p_x \otimes p_y,\, n_x \oplus n_y)\,.}$$

The pull-up network contains two PMOS devices in series, and the pull-down network contains two NMOS devices in parallel.

The resulting behavior is

x	y	PUN conducts	PDN conducts	output
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	1	0

Therefore

$$\boxed{\alpha_C(R_{C_{\mathrm{NOR}}}(x, y)) = \neg(x \vee y).}$$

The NAND and NOR examples display the fundamental series–parallel duality:

Boolean kernel	NMOS PDN	PMOS PUN
$x \wedge y$	$n_x \otimes n_y$	$p_x \oplus p_y$
$x \vee y$	$n_x \oplus n_y$	$p_x \otimes p_y$

The complementary topology is therefore derived algebraically rather than introduced as an independent circuit rule.

12.4 Composite Circuit

We now consider a computation that requires composition across multiple cells. Let

$$F(a, b, c) = (a \wedge b) \vee c.$$

This function is deliberately chosen because its direct Boolean expression is not itself a negative-monotone gate of the form used by the primitive single-cell lowering. It therefore illustrates the distinction between cell-level expressivity and network-level functional completeness.

Using NAND gates, define the intermediate signals

$$u = \mathrm{NAND}(a, b) = \neg(a \wedge b),$$

$$v = \mathrm{NAND}(c, c) = \neg c,$$

and

$$z = \mathrm{NAND}(u, v).$$

Then

$$\begin{aligned} z &= \neg(u \wedge v) \\ &= \neg\left(\neg(a \wedge b) \wedge \neg c\right) \\ &= (a \wedge b) \vee c \\ &= F(a, b, c), \end{aligned}$$

where the penultimate equality is De Morgan's law.

Thus the logic-to-gate lowering may choose the acyclic network

$$\boxed{\mathcal{L}_G(F) = \mathrm{NAND}\left(\mathrm{NAND}(a, b), \mathrm{NAND}(c, c)\right).}$$

The gate network has three nodes:

$$g_1 = \mathrm{NAND}(a, b), \qquad g_2 = \mathrm{NAND}(c, c), \qquad g_3 = \mathrm{NAND}(g_1, g_2).$$

Each NAND node is lowered independently by

$$\mathcal{L}_C(\mathrm{NAND}(x,y)) = (p_x \oplus p_y,\, n_x \otimes n_y)\,.$$

Consequently,

$$C_1 = (p_a \oplus p_b,\, n_a \otimes n_b)\,,$$
$$C_2 = (p_c \oplus p_c,\, n_c \otimes n_c)\,,$$

and, writing u and v for the output wires of C_1 and C_2,

$$C_3 = (p_u \oplus p_v,\, n_u \otimes n_v)\,.$$

The complete CMOS realization is the signal-flow composition

$$\boxed{C_F = C_3(C_1(a,b), C_2(c,c)).}$$

This notation denotes inter-cell signal-flow composition. It must not be confused with the internal transistor-network operations $\otimes$ and $\oplus$.

Local correctness gives

$$\alpha_C(R_{C_1}(a,b)) = \neg(a \wedge b) = u,$$

and

$$\alpha_C(R_{C_2}(c,c)) = \neg(c \wedge c) = \neg c = v.$$

Because the outputs of C_1 and C_2 are valid Boolean rails, they may serve as the controls of the transistors in C_3. Applying local correctness once more,

$$\alpha_C(R_{C_3}(u,v)) = \neg(u \wedge v).$$

Substitution yields

$$\alpha_C(R_{C_F}(a,b,c)) = \neg\left(\neg(a \wedge b) \wedge \neg c\right) = (a \wedge b) \vee c.$$

Hence

$$\boxed{\alpha_C(R_{C_F}(a,b,c)) = F(a,b,c).}$$

The complete truth table is

a	b	c	$u = \neg(a \wedge b)$	$v = \neg c$	$z = \neg(u \wedge v)$
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	1

The final column is exactly the truth table of

$$(a \wedge b) \vee c.$$

12.5 End-to-End Interpretation of the Composite Example

The composite example instantiates the complete discrete semantic descent.

At the source level,

$$P_F = (a \wedge b) \vee c.$$

Logic-to-gate lowering produces

$$G_F = \mathcal{L}_G(P_F),$$

with

$$[\![G_F]\!]_{\mathsf{Gate}} = [\![P_F]\!]_{\mathsf{Logic}}.$$

Gate-to-CMOS lowering produces

$$C_F = \mathcal{L}_C(G_F),$$

and the local-to-global theorem gives

$$\alpha_C^*([\![C_F]\!]_{\mathsf{CMOS}}(\rho)) = [\![G_F]\!]_{\mathsf{Gate}}(\rho).$$

Therefore,

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P_F))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P_F]\!]_{\mathsf{Logic}}(\rho).}$$

If a physical backend subsequently produces

$$P_F^{\mathrm{phys}} = \mathcal{L}_P(C_F)$$

and the admissibility conditions of Section 9 hold, then

$$\alpha_P^{[\Delta]}\left([\![P_F^{\mathrm{phys}}]\!]_{\mathsf{Phys}}(\rho,\lambda)\right) = [\![C_F]\!]_{\mathsf{CMOS}}(\rho).$$

Combining the contracts yields

$$\boxed{\alpha_C^*\left(\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P_F)))]\!]_{\mathsf{Phys}}(\rho,\lambda)\right)\right) = [\![P_F]\!]_{\mathsf{Logic}}(\rho).}$$

Thus the same computation can be followed through every representation:

$$\boxed{\text{Boolean expression} \longrightarrow \text{gate network} \longrightarrow \text{CMOS cell network} \longrightarrow \text{physical realization.}}$$

The intermediate objects differ substantially in structure, but their designated meaning is invariant under every lowering step satisfying the corresponding semantic contract.

12.6 Transformation of the Worked Realization

The examples also illustrate why semantic correctness and implementation form must remain distinct.

For example, the inverter used for

$$v = \neg c$$

was represented above as the degenerate NAND

$$\mathrm{NAND}(c, c).$$

Its pull-down network is

$$n_c \otimes n_c,$$

and its pull-up network is

$$p_c \oplus p_c.$$

At the conduction level, idempotence gives

$$n_c \otimes n_c \equiv_c n_c$$

and

$$p_c \oplus p_c \equiv_c p_c.$$

The cell may therefore be replaced by the simpler inverter

$$(p_c, n_c)$$

without changing its ideal rail semantics.

Consequently, the three-NAND gate decomposition provides one correct route from the source expression to CMOS, while a subsequent certified transformation may replace the degenerate NAND by a two-transistor inverter.

This small example exhibits the compiler discipline developed in Sections 8 and 11:

$$\boxed{\text{correct lowering} \longrightarrow \text{certified semantic transformation} \longrightarrow \text{implementation selection.}}$$

No unique transistor topology is identified with the meaning of the source computation. The semantics determines an equivalence class of correct realizations, and later compiler stages may choose among members of that class according to additional implementation criteria.

13 Related Work

The framework developed in this paper intersects several established research traditions: switch-level models of MOS circuits, logic synthesis and technology mapping, formal hardware verification, compiler intermediate representations, and verified compilation. Its contribution is not to replace these traditions, but to organize a particular path through them as a sequence of explicit semantic representations and correctness-preserving lowerings from formal logic to physical CMOS realization.

13.1 Switch-Level Models of MOS Circuits

Formal switch-level modeling of MOS circuits substantially predates the present work. Bryant developed a switch-level model in which MOS networks are represented by nodes connected through transistor switches, with node values including logical 0, logical 1, and an indeterminate state [1]. That framework was designed to capture behavior beyond the ordinary unidirectional Boolean-gate abstraction, including bidirectional transistor behavior, storage, charge-related effects, and other properties important to MOS circuits.

Subsequent symbolic work showed that switch-level MOS behavior could be analyzed through Boolean representations. In particular, Bryant's Boolean analysis of MOS circuits expresses switch-network behavior using systems of Boolean equations and derives symbolic Boolean descriptions of network behavior [2]. Related work on switch-level simulation and symbolic analysis further demonstrated that transistor networks can be given mathematically structured discrete semantics rather than treated only through numerical device simulation.

The present framework shares with this tradition the principle that a transistor network requires its own semantic level rather than being identified directly with a Boolean formula. Its scope, however, is deliberately narrower at the CMOS layer. Sections 5–8 concentrate on complementary static CMOS and isolate a conduction algebra in which series and parallel transistor structure can be related compositionally to Boolean behavior.

This restriction permits a particularly direct lowering theory:

$$\text{Boolean gate} \longrightarrow \text{complementary CMOS cell}$$

together with exact local and global semantic-preservation theorems. Richer switch-level phenomena such as dynamic storage, bidirectional pass-transistor behavior, and charge sharing are not modeled by the present conduction algebra and would require an extension of its semantic domain.

Mathematical models specifically for combinational static CMOS have also studied the structure and behavior of complementary CMOS networks [**?**]. The present work is distinguished primarily by the role assigned to the CMOS model inside a larger compiler-oriented semantic descent: the transistor algebra is used as an intermediate representation between gate semantics and an explicitly separate physical semantics.

13.2 Logic Synthesis and Technology Mapping

Logic synthesis transforms Boolean or register-transfer representations into implementation structures while attempting to improve cost measures such as area, delay, or power. Classical technology mapping further replaces technology-independent logic structures with components available in a target library.

A particularly influential formulation treats technology mapping as a graph- or DAG-covering problem. Keutzer's DAGON work related technology binding to the pattern-matching techniques used in compiler code generation [4]. In this view, a subject logic graph is covered by patterns corresponding to available implementation elements, with costs guiding the choice among valid coverings.

Modern synthesis systems continue this broad separation between logical transformation and implementation mapping. Yosys, for example, provides a pass-oriented synthesis framework for behavioral and RTL designs and uses specialized logic-synthesis machinery for gate-level optimization [5].

CMOS Silicon Algebra addresses a different level of the synthesis problem. It does not propose a new general-purpose Boolean optimization algorithm or a replacement for established technology-mapping procedures. Instead, it asks what semantic obligations a synthesis or lowering procedure must satisfy when moving explicitly from

$$\mathsf{Logic} \to \mathsf{Gate} \to \mathsf{CMOS}.$$

This distinction motivates the separation developed in Section 8 between semantic admissibility and optimization cost. A technology mapper may search among many implementations using arbitrary cost models, but a candidate transformation is accepted semantically only when it remains in the required equivalence class.

The resulting principle,

$$\boxed{\text{semantic admissibility first, cost selection second,}}$$

is compatible with conventional synthesis and technology mapping while making the correctness boundary explicit.

13.3 Formal Hardware Verification

Formal hardware verification encompasses theorem proving, model checking, equivalence checking, symbolic simulation, and related techniques for establishing properties of hardware designs.

One relevant line of work integrates hardware description directly with proof assistants. Kami is a Coq-based framework for high-level parametric hardware specification and modular verification [6]. Designs, specifications, and implementation relations can be expressed within Coq, and verified designs can be extracted into a hardware implementation flow. Kami demonstrates that hardware generators and parameterized hardware systems can be treated as proof-bearing formal objects rather than merely as external netlists subjected to post hoc checking.

The present framework shares the objective of compositional correctness but focuses on a different representation boundary. Its central object is the semantic descent from source logic through gate networks to complementary transistor networks and ultimately to physical behavior. The principal correctness statements therefore have the form

$$\alpha_B(\llbracket L_{A\to B}(a)\rrbracket_B) = \llbracket a\rrbracket_A,$$

where a lowering introduces a more concrete representation and an abstraction map recovers the source meaning.

Equivalence checking also plays an important role in practical synthesis flows. Synthesis transformations may substantially change network structure while preserving externally observable logical behavior. Work integrating synthesis and verification has shown the value of retaining information about the transformations performed during optimization so that equivalence can be checked efficiently.

Section 8 gives an algebraic instance of this general idea. A conduction rewrite

$$K \equiv_c L$$

acts as a local certificate from which rail and abstract Boolean equivalence can be derived for the corresponding complementary CMOS cells.

13.4 Hardware Intermediate Representations

Modern compiler infrastructure increasingly treats hardware compilation as a multi-representation problem.

MLIR provides extensible compiler infrastructure in which multiple intermediate representations, or dialects, may coexist and be progressively lowered while retaining domain-specific structure [7]. Its design is motivated in part by the need to support compilation across heterogeneous abstraction levels and specialized computational domains.

CIRCT applies this style of infrastructure specifically to hardware design [8]. It provides hardware-oriented intermediate representations and compiler passes within the MLIR ecosystem, with the broader goal of bringing reusable compiler infrastructure and explicit intermediate representations to hardware tooling.

CMOS Silicon Algebra is complementary to such infrastructure. It does not prescribe a concrete compiler framework, serialization format, or software implementation of its intermediate representations. Instead, it specifies semantic roles for the representations

$$\mathsf{IR}_{\mathsf{Logic}}, \qquad \mathsf{IR}_{\mathsf{Gate}}, \qquad \mathsf{IR}_{\mathsf{CMOS}}, \qquad \mathsf{IR}_{\mathsf{Phys}}$$

and identifies the correctness contracts required at their boundaries.

A future implementation could therefore encode these representations within an existing extensible compiler infrastructure or within a dedicated compiler. The mathematical theory is intended to remain independent of that engineering choice.

13.5 Verified Compilation

The organization of compiler correctness as semantic preservation across intermediate languages has a substantial precedent in verified software compilation.

CompCert provides a prominent example. Its compiler is accompanied by machine-checked proofs that compilation preserves the semantics of source programs through a sequence of intermediate representations and compiler passes [9, 10]. The general methodological lesson is that end-to-end compiler correctness can be decomposed into preservation arguments associated with individual transformations rather than proved as one monolithic correspondence between source and machine code.

The semantic contracts of Section 7 adopt this compositional perspective for the hardware lowering problem. If

$$L_{A\to B} : A \to B$$

and

$$L_{B\to C} : B \to C$$

satisfy compatible abstraction contracts, their composition inherits an end-to-end preservation theorem.

The target representations in the present work are fundamentally different from those of a conventional software compiler. The final stages expose transistor connectivity and physical behavior rather than instruction execution. Nevertheless, the proof architecture is analogous: correctness is transported across representation boundaries by explicit semantic relations.

This connection motivates the proof-producing compiler architecture proposed in Section 11. Optimization and candidate generation need not themselves constitute the trusted semantic argument if their outputs can be accompanied by independently checkable preservation evidence.

13.6 High-Level and Hardware Compilation

High-level synthesis and hardware compilation translate computational descriptions into progressively more concrete hardware structures. Recent work has also explored compiler infrastructures that combine high-level program transformation with hardware-oriented lowering, including MLIR-based flows for accelerator generation and hardware synthesis.

The present work occupies a lower and more explicitly transistor-oriented portion of this design space. Its principal lowering target is not merely RTL or a gate library but an algebra of complementary CMOS transistor networks.

At the same time, the framework deliberately stops short of claiming that ideal transistor connectivity constitutes a complete physical implementation. Section 9 introduces a separate physical representation and a conditional realization contract

$$\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}\right) = [\![C]\!]_{\mathsf{CMOS}}$$

under explicit admissibility assumptions.

This boundary separates exact discrete correctness from electrical realizability and permits physical models of different levels of fidelity to serve as backends to the same ideal CMOS semantics.

13.7 Position of the Present Framework

The closest conceptual neighbors of CMOS Silicon Algebra therefore address different portions of the same broad path from computation to hardware:

- switch-level MOS models formalize transistor-network behavior;
- logic synthesis transforms and optimizes Boolean networks;
- technology mapping selects implementations from target libraries;
- formal hardware verification establishes implementation properties and equivalences;
- hardware compiler infrastructures organize multiple hardware intermediate representations; and
- verified compilation supplies compositional methods for proving semantic preservation across representation changes.

The present framework combines these concerns around one explicit semantic spine:

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys.}}$$

Its specific contribution is the organization of these layers by paired lowering and abstraction maps, together with exact preservation results for the discrete portion of the descent, an algebra of certified CMOS transformations, and a conditional refinement contract for the transition from ideal CMOS semantics to physical behavior.

Accordingly, the framework should not be read as asserting that Boolean logic, transistor-level semantics, hardware verification, or compiler lowering are individually new. The proposed contribution lies in their composition into a single compiler-oriented semantic architecture in which the meaning preserved at each representation boundary is stated explicitly.

14 Discussion

The central claim of CMOS Silicon Algebra is not that Boolean logic, CMOS circuits, or physical transistor behavior can be reduced to one undifferentiated mathematical representation. The opposite distinction is essential. Each level exposes structure that is invisible at the level above it, and each abstraction deliberately forgets distinctions that are irrelevant to the meaning represented at that level.

The framework therefore treats compilation toward silicon as a sequence of representation changes:

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys.}}$$

Correctness is not identity of representations. It is preservation of designated meaning across their boundaries.

This perspective has consequences both for the scope of the present theory and for the architecture of a future compiler.

14.1 Scope

The exact mathematical core of the present framework concerns finite combinational computation and complementary static CMOS.

At the logical layer, computations are interpreted denotationally as Boolean functions with explicit interfaces. At the gate layer, those computations are represented as networks whose wiring, composition, fan-out, and sharing are structurally explicit. At the CMOS layer, supported gates are lowered to complementary pull-up and pull-down transistor networks and interpreted first through conduction and rail semantics.

Within this domain, the framework establishes several exact results.

First, the conduction algebra gives a compositional interpretation of series and parallel transistor structure:

$$\otimes \quad \longleftrightarrow \quad \wedge, \qquad \oplus \quad \longleftrightarrow \quad \vee.$$

Second, CMOS duality provides a constructive relationship between pull-down and pull-up networks. For a conduction expression K, the complementary construction

$$C[K] = (K^\star, K)$$

produces a valid static CMOS cell under the assumptions established in Section 5.

Third, gate-to-CMOS lowering preserves Boolean meaning under the rail abstraction. This local result composes across finite acyclic gate networks.

Fourth, the logic-to-gate and gate-to-CMOS contracts compose into an end-to-end preservation theorem:

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

Fifth, conduction equivalence supplies a basis for certified CMOS transformations. A circuit may therefore change structurally without changing its designated abstract meaning.

These results deliberately stop short of identifying ideal CMOS semantics with complete transistor physics.

The framework does not attempt to derive MOSFET current-voltage equations, model semiconductor fabrication, solve transistor differential equations, perform parasitic extraction, construct geometric layout, or replace electrical simulation. It also does not claim that conduction-equivalent networks have equal delay, area, energy, capacitance, leakage, noise margins, or reliability.

Those distinctions belong to the physical representation.

Section 9 therefore introduces physical behavior through a separate semantic domain and a conditional abstraction contract rather than weakening the exact discrete semantics with device-specific approximations.

The present treatment of sequential logic is likewise intentionally limited. Section 10 shows how the preservation architecture extends from combinational functions to state-transition systems, but a complete theory of latches, flip-flops, clocking, metastability, initialization, asynchronous behavior, and temporal composition remains outside the exact combinational core developed here.

The scope can therefore be summarized as

exact semantics for supported combinational logic, gates, and ideal complementary CMOS networks
+ an explicit contract for refinement toward physical realization.

This boundary is intentional. A formal model becomes less informative, not more, if exact theorems and engineering approximations are stated in the same semantic regime without distinguishing their assumptions.

14.2 Software Abstraction and Physical Computation

The broader purpose of the framework is to make explicit the relationship between software-level abstraction and physical computation.

A Boolean expression such as

$$(a \wedge b) \vee c$$

contains no transistors, voltages, capacitances, or propagation delays. Its meaning is specified independently of any physical realization.

A gate network implementing the same expression introduces structural distinctions. The computation may now have multiple nodes, explicit wires, fan-out, sharing, and a particular decomposition into primitive gates.

A CMOS realization introduces still more distinctions. Gates become complementary transistor networks; series and parallel conduction structure becomes explicit; and supply connectivity determines ideal rail behavior.

A physical realization introduces further distinctions again: continuous voltage, time, load, device characteristics, interconnect, environmental conditions, and technology-specific behavior.

Thus the descent

$$\mathsf{Logic} \longrightarrow \mathsf{Gate} \longrightarrow \mathsf{CMOS} \longrightarrow \mathsf{Phys}$$

does not merely translate notation. Each step adds implementation information.

Abstraction moves in the opposite direction. Physical trajectories are interpreted as settled rail behavior; rail behavior is interpreted as Boolean behavior; gate-network structure is interpreted as a logical function.

Schematically,

Logic	$\xrightarrow{\mathcal{L}_G}$	Gate	$\xrightarrow{\mathcal{L}_C}$	CMOS	$\xrightarrow{\mathcal{L}_P}$	Phys
meaning		structure		switching		physical behavior.

Lowering introduces distinctions. Abstraction forgets them.

This gives a precise interpretation to the claim that software abstractions are ultimately realized by physical computation. The relationship is not expressed by saying that a Boolean value *is* a voltage, that a gate *is* a transistor network, or that a program *is* its physical implementation. These objects inhabit different semantic domains.

Instead, a physical realization implements an abstract computation when the appropriate abstraction maps recover the same observable meaning.

For a supported source computation P, the complete correctness statement has the form

$$\boxed{\alpha_C^*\Big(\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}}(\rho,\lambda)\right)\Big) = [\![P]\!]_{\mathsf{Logic}}(\rho),}$$

under the physical admissibility assumptions of Section 9.

The equation states what survives the complete descent.

The source expression and its physical realization may differ in virtually every representational respect. One is symbolic and discrete; the other is electrical and temporal. Yet the designated computation remains invariant when every lowering satisfies its semantic contract.

This suggests a general principle underlying the framework:

correct lowering may change representation without changing meaning.

The principle also explains why semantic equivalence is necessarily abstraction-relative.

Two source expressions may be logically equivalent while having different syntax. Two gate networks may compute the same Boolean function while having different depth or fan-out. Two CMOS networks may have the same rail semantics while differing in transistor topology. Two physical realizations may implement the same Boolean computation while differing in delay, energy, layout, loading, or device parameters.

Consequently,

$$\text{Boolean equivalence} \not\Rightarrow \text{gate-structural equivalence} \not\Rightarrow \text{CMOS-topological equivalence} \not\Rightarrow \text{physical equivalence}.$$

The implications should not be read as failures of the semantics. They describe the information deliberately forgotten by successive abstractions.

This distinction is particularly important for optimization. If two realizations inhabit the same abstract semantic equivalence class, the compiler is free to choose between them according to properties that the abstraction does not preserve.

The semantic theory therefore creates optimization freedom precisely by separating meaning from representation.

14.3 From Paper to Compiler

The paper is intended to serve as the semantic specification against which the future *CMOS Silicon Compiler* can be implemented and experimentally tested.

The mathematical objects developed here correspond directly to compiler representations:

$$\mathsf{Logic}, \qquad \mathsf{Gate}, \qquad \mathsf{CMOS}, \qquad \mathsf{Phys}.$$

The lowering maps correspond to compiler passes:

$$\mathcal{L}_G, \qquad \mathcal{L}_C, \qquad \mathcal{L}_P.$$

Semantic equivalence relations determine which transformations are admissible, while cost functions and physical models determine which admissible implementation should be selected.

A prototype therefore need not begin by solving the complete physical-design problem. The first implementation can test the exact discrete theory.

A minimal compiler can:

1. parse a supported logical source representation;
2. construct and validate its logical intermediate representation;
3. lower the computation to a gate network;
4. evaluate the gate semantics independently;
5. lower supported gate nodes to complementary CMOS cells;
6. construct explicit transistor-network representations;
7. evaluate conduction and rail semantics;
8. compare the abstracted CMOS result with the original source denotation;
9. apply certified conduction-level transformations; and

10. verify that transformed networks remain in the same semantic equivalence class.

This prototype would already test the central claim of the paper:

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}) = [\![P]\!]_{\mathsf{Logic}}.}$$

The worked examples of Section 12 provide immediate regression cases for such an implementation. The inverter, NAND, NOR, and composite circuit can be generated automatically and checked against their source truth tables.

A later backend can extend the compiler boundary toward conventional transistor-level representations and external electrical tools. Such a backend could emit a transistor netlist, obtain physical voltage and timing traces from an electrical model or simulator, and test whether those traces satisfy the abstraction contract of Section 9.

The development path is therefore incremental:

formal semantics
↓
executable intermediate representations
↓
verified discrete lowering
↓
certified circuit transformation
↓
physical backend integration
↓
experimental validation.

The paper and compiler consequently have different roles.

The paper defines the representations, semantics, equivalence relations, lowering maps, and proof obligations. The compiler makes those objects executable.

This distinction also provides a practical testing methodology. Because the layers have independent semantics, compiler output can be checked at every boundary rather than only by observing the final circuit.

For a source computation P, one can independently compare

$$[\![P]\!]_{\mathsf{Logic}}, \qquad [\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}},$$

then compare the gate result with

$$\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}),$$

and, once a physical backend exists, compare the ideal CMOS result with

$$\alpha_P^{[\Delta]}([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}}).$$

A failure can therefore be localized to a particular representation boundary.

This is one of the principal engineering consequences of the compositional theory. Correctness becomes a collection of small, explicit, testable contracts rather than one opaque assertion that a source expression and a physical circuit somehow compute the same thing.

The future compiler can also exploit the distinction between correctness and optimization established in Section 8. Search procedures may generate multiple gate decompositions, transistor networks, or physical candidates. Provided that each accepted candidate carries or passes the required semantic certificate, optimization can proceed over the resulting admissible class.

The intended relationship between the theory and its implementation is therefore

$$\boxed{\text{paper} \;=\; \text{semantic specification}, \qquad \text{compiler} \;=\; \text{executable realization of that specification.}}$$

The transition from CMOS Silicon Algebra to the CMOS Silicon Compiler is thus not a change of subject. It is the next stage of the same construction: turning the semantic descent developed in this paper into an executable, testable, and eventually physically grounded compilation system.

15 Conclusion

This paper has developed CMOS Silicon Algebra as a compositional semantic framework for relating formal logic, gate networks, complementary CMOS transistor networks, and physical realization.

The central construction is the semantic descent

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys.}}$$

These representations are intentionally distinct. A logical expression is not a gate network, a gate network is not a transistor network, and an ideal transistor network is not its physical electrical realization. Each lowering introduces implementation structure absent from the representation above it.

Correctness is therefore expressed not by requiring representations to be identical, but by requiring their semantic relationships to commute under the appropriate abstraction maps.

At the CMOS level, the paper introduced a conduction algebra in which series and parallel transistor composition correspond to conjunction and disjunction of conduction conditions. CMOS duality then provides a constructive relationship between pull-down and pull-up networks, yielding valid complementary cells from monotone conduction kernels.

This structure supports a gate-to-CMOS lowering with an exact local correctness theorem. Because the semantics is compositional, local correctness extends to finite acyclic gate networks. When combined with a correct logic-to-gate lowering, the result is the end-to-end discrete preservation property

$$\boxed{\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

The paper further showed that correctness does not determine a unique implementation. Conduction equivalence, rail equivalence, gate equivalence, and logical equivalence define semantic classes within which circuit representations may be transformed.

This yields a disciplined basis for optimization:

$$\boxed{\text{semantic admissibility first, cost selection second.}}$$

A compiler may therefore search among multiple implementations without identifying any particular gate decomposition or transistor topology with the meaning of the computation itself.

The transition from ideal CMOS semantics to physical computation requires a different kind of contract. Physical voltages are continuous, switching requires time, loading affects behavior, and implementation properties depend on technological and environmental parameters. These effects should not be silently incorporated into an otherwise exact Boolean semantics.

Accordingly, the physical layer was introduced as an explicit refinement boundary. Under a physical admissibility predicate, a correct realization must satisfy

$$\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho,\lambda)\right) = [\![C]\!]_{\mathsf{CMOS}}(\rho).$$

Composing this contract with the discrete preservation theorems produces the complete logic-to-physical correctness condition

$$\boxed{\alpha_C^*\Big(\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}}(\rho,\lambda)\right)\Big) = [\![P]\!]_{\mathsf{Logic}}(\rho).}$$

This equation captures the principal invariant of the framework. The representations may change from symbolic logic to electrical behavior, while the designated computation remains recoverable through abstraction.

The distinction between semantic layers also clarifies the limits of equivalence. Boolean equivalence need not imply identical gate structure; gate equivalence need not imply identical transistor topology; and ideal CMOS equivalence need not imply physical equivalence. Each abstraction preserves the distinctions relevant to its semantic level while forgetting others.

Rather than obstructing implementation, this loss of information creates the freedom required for compilation and optimization. Multiple physical structures may realize the same abstract computation, allowing implementation choices to be governed by delay, area, loading, energy, technology, or other backend criteria once semantic correctness has been established.

The resulting theory also determines the architecture of the proposed *CMOS Silicon Compiler*. Logical, gate, CMOS, and physical representations become compiler intermediate representations; lowering maps become compiler passes; semantic equivalence becomes the criterion for certified transformation; and the physical admissibility contract determines the boundary between exact compilation and technology-dependent realization.

The paper therefore serves as a semantic specification for the compiler rather than as a description of a completed implementation. A prototype can first make the discrete layers executable, verify the lowering theorems experimentally against generated circuits, and implement certified conduction-level transformations. Physical backends can then be introduced without changing the semantic architecture established here.

The broader principle is that movement toward physical implementation should not require the meaning of a computation to become implicit.

Instead, every representation boundary should state explicitly what is added, what is forgotten, and what must remain invariant.

For CMOS Silicon Algebra, that invariant is:

> The meaning of a computation remains invariant under every correct lowering step from formal logic toward physical CMOS realization.

The path from abstraction to silicon can therefore be treated not merely as a sequence of implementation choices, but as a compositional semantic descent whose correctness can be stated, proved, transformed, and ultimately made executable.

A Notation

This appendix summarizes the principal notation used throughout the paper. The table is intended as a reference rather than as a substitute for the definitions in the main text. When a symbol has semantics specific to one representation layer, the definition given in that section is authoritative.

A.1 Semantic Layers and Lowering Maps

Symbol	**Meaning**
Logic	source logical representation
Gate	gate-network representation
CMOS	ideal CMOS transistor-network representation
Phys	physical/electrical representation
$\mathcal{L}_G : \mathsf{Logic} \to \mathsf{Gate}$	logic-to-gate lowering map
$\mathcal{L}_C : \mathsf{Gate} \to \mathsf{CMOS}$	gate-to-CMOS lowering map
$\mathcal{L}_P : \mathsf{CMOS} \to \mathsf{Phys}$	CMOS-to-physical realization map
α_C	abstraction from CMOS rail semantics toward Boolean/gate semantics
α_C^*	componentwise extension of α_C to multi-output networks
$\alpha_P^{[\Delta]}$	physical settling abstraction over observation interval Δ
$[\![P]\!]_A$	denotation of object P in semantic domain or layer A

The principal lowering chain is

$$\boxed{\mathsf{Logic} \xrightarrow{\mathcal{L}_G} \mathsf{Gate} \xrightarrow{\mathcal{L}_C} \mathsf{CMOS} \xrightarrow{\mathcal{L}_P} \mathsf{Phys}.}$$

A.2 Boolean and Interface Notation

Symbol	**Meaning**
$\mathbb{B} = \{0, 1\}$	Boolean value domain
X	finite set of input variables or input names
$\rho : X \to \mathbb{B}$	Boolean input valuation
$\mathbb{B}^X$	set of Boolean valuations over X
P, Q	source-level logical computations or expressions
$P \equiv_{\mathsf{Logic}} Q$	logical semantic equivalence
G, H	gate networks
$G \equiv_{\mathsf{Gate}} H$	gate-level semantic equivalence

Semantic equivalence is always relative to the representation layer under consideration. Equality of Boolean denotation does not imply identity of gate, transistor, or physical structure.

A.3 CMOS Connectivity and Rail Semantics

Symbol	**Meaning**
N	CMOS switch or transistor network
B	set of electrical boundary or network nodes
$\mathrm{Eq}(B)$	set of equivalence relations on B
$[\![N]\!]_{\mathrm{conn}}$	connectivity semantics of network N under an input valuation
H	high supply rail
L	low supply rail
$R_N(\rho, o)$	set of supply rails connected to output o of N under valuation ρ
$\varnothing$	floating or undriven rail state, denoted informally by Z
$\{H\}$	uniquely high-driven output rail state
$\{L\}$	uniquely low-driven output rail state
$\{H, L\}$	simultaneous high/low connection, denoted informally by X
α_C	partial Boolean abstraction of a valid rail state

For a valid static CMOS output,

$$\alpha_C(\{H\}) = 1, \qquad \alpha_C(\{L\}) = 0,$$

while floating and conflicting rail states are outside the Boolean abstraction domain.

A.4 Conduction Algebra

Symbol	**Meaning**
K, L	conduction expressions
$\mathbf{0}$	never-conducting expression
$\mathbf{1}$	always-conducting expression
n_x	NMOS conduction literal controlled by input x
p_x	PMOS conduction literal controlled by input x
$K \otimes L$	series composition of conduction structures
$K \oplus L$	parallel composition of conduction structures
$\chi_\rho(K)$	Boolean conduction value of K under valuation ρ
$K \equiv_c L$	conduction equivalence: $\chi_\rho(K) = \chi_\rho(L)$ for every valuation ρ
$K^\star$	complementary CMOS dual of conduction expression K
$\mathrm{Min}(K)$	minimal support sets associated with the conduction function of K
$\mathsf{NF}(K)$	canonical minimal-support conduction normal form

Series and parallel composition satisfy

$$\chi_\rho(K \otimes L) = \chi_\rho(K) \wedge \chi_\rho(L),$$

and

$$\chi_\rho(K \oplus L) = \chi_\rho(K) \vee \chi_\rho(L).$$

The CMOS dual satisfies

$$\chi_\rho(K^\star) = 1 - \chi_\rho(K), \qquad (K^\star)^\star = K.$$

A.5 Static CMOS Cells and Equivalence

Symbol	**Meaning**
$C = (U, D)$	static CMOS cell with pull-up network U and pull-down network D
$C[K] = (K^\star, K)$	complementary CMOS cell constructed from conduction kernel K
U	pull-up network (PUN)
D	pull-down network (PDN)
$C_1 \equiv_R C_2$	rail-semantic equivalence of CMOS cells or networks
$C_1 \equiv_{\alpha_C} C_2$	equivalence after Boolean abstraction through α_C
$\mathcal{A}(C)$	semantic admissibility class of implementations equivalent to C

The distinctions among these relations are intentional. In particular,

$$\boxed{\text{conduction equivalence} \neq \text{physical equivalence.}}$$

Two networks may agree at the conduction or Boolean level while differing in transistor count, resistance, capacitance, delay, loading, switching activity, or other physical properties.

A.6 Physical Interpretation

Symbol	**Meaning**
$\mathbb{V} = [0, V_{\mathrm{DD}}]$	continuous voltage domain
V_{DD}	positive supply voltage
δ_V	partial abstraction from admissible voltage regions to Boolean values
(v, u, d)	physical observation consisting of voltage and high/low drive indicators
α_P^{rail}	abstraction from a physical observation to its driven-rail set
Obs_m	observation space for m outputs
Tr_m	time-indexed physical trace space for m outputs
λ	collection of physical, technological, loading, and environmental parameters
$[\![P]\!]_{\mathsf{Phys}}(\rho, \lambda)$	set of physical traces admitted by physical realization P
$\alpha_P^{[\Delta]}$	abstraction of sufficiently settled physical behavior over interval Δ
T_{set}	settling-time bound
Adm_P	physical admissibility predicate
D_i	local delay bound associated with component or stage i
D_G	compositional delay bound for a gate network or path

Unlike the exact Boolean and ideal-switch semantics, physical correctness is conditional on explicit admissibility assumptions. Its characteristic contract is

$$\alpha_P^{[\Delta]}\left([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho, \lambda)\right) = [\![C]\!]_{\mathsf{CMOS}}(\rho).$$

A.7 Sequential Extension

Symbol	**Meaning**
I	input interface of a sequential component
S	state interface or state space
O	output interface
δ	state-transition function
ω	output function
s_t	state at discrete step t
ρ_t	input valuation at discrete step t
o_t	output at discrete step t
α_S	abstraction between lower- and higher-level state representations
α_O	abstraction between lower- and higher-level output representations

The sequential extension uses

$$s_{t+1} = \delta(\rho_t, s_t), \qquad o_t = \omega(\rho_t, s_t),$$

replacing purely combinational denotation by state-transition denotation.

A.8 Compiler Notation

Symbol	**Meaning**
$\mathsf{IR}_{\mathsf{Logic}}$	logical intermediate representation
$\mathsf{IR}_{\mathsf{Gate}}$	gate-network intermediate representation
$\mathsf{IR}_{\mathsf{CMOS}}$	CMOS transistor-network intermediate representation
$\mathsf{IR}_{\mathsf{Phys}}$	technology or physical intermediate representation
$L_{A\to B}$	generic compiler lowering from representation A to representation B
Pass	compiler transformation or proof-producing pass
Θ	technology description or technology-specific parameter environment
Map_Θ	technology-dependent mapping from CMOS representation toward physical realization

The compiler interpretation of the formal framework can therefore be summarized by

$$\boxed{\text{representation boundary} \longleftrightarrow \text{proof boundary.}}$$

Each lowering pass may change representation, but acceptance of the pass requires preservation of the meaning designated at that boundary.

B Core Proof Obligations

This appendix collects the principal proof obligations underlying the semantic architecture developed in the paper. Most of these obligations are discharged by the constructions and results of the preceding sections; they are collected here to make explicit the dependency structure that an implementation of the CMOS Silicon Compiler must preserve.

1. **Source semantics.** The source language must have a well-defined denotational semantics

$$[\![-]\!]_{\mathsf{Logic}}$$

such that source-level semantic equivalence is determined by equality of denotation.

2. **Gate semantics.** Gate networks must have a compositional semantics

$$[\![-]\!]_{\mathsf{Gate}}$$

defined independently of their eventual CMOS realization.

3. **Logic-to-gate preservation.** The lowering

$$\mathcal{L}_G : \mathsf{Logic} \to \mathsf{Gate}$$

must satisfy

$$[\![\mathcal{L}_G(P)]\!]_{\mathsf{Gate}} = [\![P]\!]_{\mathsf{Logic}}$$

for every well-formed source computation P.

4. **Transistor conduction semantics.** Every CMOS conduction term must determine a conduction function

$$\chi_\rho(K),$$

with series composition interpreted conjunctively and parallel composition interpreted disjunctively.

5. **CMOS duality.** The CMOS dual operation must satisfy

$$\chi_\rho(K^\star) = 1 - \chi_\rho(K)$$

and

$$(K^\star)^\star = K.$$

6. **Complementary-cell validity.** For every admissible conduction kernel K, the complementary construction

$$C[K] = (K^\star, K)$$

must define a valid static CMOS cell whose pull-up and pull-down networks are complementary under every Boolean valuation.

7. **Local gate-to-CMOS correctness.** For every gate in the supported lowering domain, the CMOS realization produced by

$$\mathcal{L}_C : \mathsf{Gate} \to \mathsf{CMOS}$$

must have rail semantics whose Boolean abstraction agrees with the semantics of the source gate.

8. **Compositional CMOS correctness.** Correctness of individual CMOS cells must be preserved under well-formed signal-flow composition. Consequently, for a finite acyclic gate network G,

$$\alpha_C^*([\![\mathcal{L}_C(G)]\!]_{\mathsf{CMOS}}(\rho)) = [\![G]\!]_{\mathsf{Gate}}(\rho).$$

9. **End-to-end discrete preservation.** Logic-to-gate and gate-to-CMOS correctness must compose, yielding
$$\alpha_C^*([\![\mathcal{L}_C(\mathcal{L}_G(P))]\!]_{\mathsf{CMOS}}(\rho)) = [\![P]\!]_{\mathsf{Logic}}(\rho).$$

10. **Transformation preservation.** Every compiler rewrite admitted as semantics preserving must remain within the appropriate semantic equivalence class. In particular, a conduction rewrite
$$K \equiv_c L$$
must preserve the ideal rail behavior of the corresponding complementary CMOS cells.

11. **Physical refinement.** A physical realization
$$\mathcal{L}_P : \mathsf{CMOS} \to \mathsf{Phys}$$
must be accompanied by explicit admissibility assumptions. Under those assumptions and an adequate settling interval Δ, physical behavior must abstract to the ideal CMOS semantics:
$$\alpha_P^{[\Delta]}([\![\mathcal{L}_P(C)]\!]_{\mathsf{Phys}}(\rho,\lambda)) = [\![C]\!]_{\mathsf{CMOS}}(\rho).$$

12. **Logic-to-physical preservation.** Whenever the physical admissibility conditions hold, the complete lowering chain must satisfy
$$\alpha_C^*\Big(\alpha_P^{[\Delta]}([\![\mathcal{L}_P(\mathcal{L}_C(\mathcal{L}_G(P)))]\!]_{\mathsf{Phys}}(\rho,\lambda))\Big) = [\![P]\!]_{\mathsf{Logic}}(\rho).$$

13. **Separation of semantic and physical equivalence.** No correctness theorem at a coarser abstraction level may be promoted automatically to a claim of physical equivalence. In particular, conduction-equivalent or Boolean-equivalent implementations may differ in delay, area, capacitance, loading, energy, leakage, noise behavior, and other physical properties.

14. **Compiler preservation.** Every compiler pass that crosses or transforms a representation boundary must either construct evidence of the corresponding preservation obligation or produce an object whose correctness can be independently checked against that obligation.

Together these obligations express the proof architecture of the framework:

representation boundary $\longleftrightarrow$ semantic preservation obligation.

They also separate the exact mathematical core of CMOS Silicon Algebra from technology-dependent physical certification. The discrete lowering theorems are exact within their stated models; physical correctness is conditional on the explicit admissibility assumptions associated with the chosen realization.

References

[1] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, vol. C-33, no. 2, pp. 160–177, 1984.

[2] R. E. Bryant, "Boolean Analysis of MOS Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-6, no. 4, pp. 634–649, 1987.

[3] J. A. Brzozowski and M. Yoeli, "Combinational Static CMOS Networks," *Integration*, vol. 5, no. 2, pp. 103–122, 1987.

[4] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," in *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 341–347, 1987.

[5] C. Wolf and J. Glaser, "Yosys—A Free Verilog Synthesis Suite," in *Proceedings of Austrochip 2013*, 2013.

[6] J. Choi, M. Vijayaraghavan, B. Sherman, A. Chlipala, and Arvind, "Kami: A Platform for High-Level Parametric Hardware Specification and Its Modular Verification," *Proceedings of the ACM on Programming Languages*, vol. 1, ICFP, Article 24, 2017.

[7] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation," in *Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, 2021.

[8] CIRCT Project, "Circuit IR Compilers and Tools," LLVM Project. Available: `https://circt.llvm.org/`.

[9] X. Leroy, "Formal Verification of a Realistic Compiler," *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, 2009.

[10] X. Leroy, "A Formally Verified Compiler Back-End," *Journal of Automated Reasoning*, vol. 43, no. 4, pp. 363–446, 2009.