What are “semantics”? 

- Theoretically: the meaning of some program / expression / term
- Pragmatically: a reference implementation of a language
- i.e. a reference interpreter
- This serves as a gold standard of truth for diagnosing compiler bugs, developing high performance optimizing compilers, etc.
- Ideally: very small interpreter; mathematically “nice”
Check Out My Cool New HW Language!

- Usual “reference” implementation (translate to verilog)
- So then what are Verilog’s semantics?
Figure 4-1—Event scheduling regions
Two Weird Features of the Verilog Spec

1) Recursive Event Propagation

Continue sending messages in loops until process quiesces

4. termination?
Two Weird Features of the Verilog Spec

1) Recursive Event Propagation
   - Keep sending messages in loops until process quiesces
   - $\leq$ termination?

2) "4-Valued Logic"
   - 0, 1, X, Z
     - Unknown
     - Don't care
     - "High-impedance"
Aside: VHDL / IEEE1164

9-valued logic

→ “don’t care” vs. “uninitialized” vs. “unknown”
→ “weak” vs. “strong” signals
Reduction to “common tools” or standards makes semantics less clear
Back to Square 1
Simple Example with Hierarchy
Hierarchical Counter

Counter(1)

Counter(2)

Reg init to 0
Simple Example with Hierarchy

Simple Encoding as a Netlist

def Counter1Bit():
c = Circuit('Counter(1)',
['cin'],
['cnt', 'cout'])
c.add_inst(XOR(), ['cin', 'cnt'],
['sum'])
c.add_inst(Reg(), ['sum'],
['cnt'])
c.add_inst(AND(), ['cin', 'cnt'],
['cout'])
return c
def Counter2Bit():
c = Circuit('Counter(2)',
['cin'],
['cnt0', 'cnt1', 'cout'])
c.add_inst( Counter1Bit(), ['cin'],
['cnt0', 'cnt1', 'cout0'])
c.add_inst( Counter1Bit(), ['cout0'],
['cnt1', 'cout'])
return c
Simulate As Software

Lexical Ordering

def Counter1Bit():
    c = Circuit('Counter(1)',
                ['cin'],
                ['cnt', 'cout'])
    c.add_inst(XOR(), ['cin', 'cnt'],
                ['sum'])
    c.add_inst(Reg(), ['sum'],
                ['cnt'])
    c.add_inst(AND(), ['cin', 'cnt'],
                ['cout'])
    return c

def Counter2Bit():
    c = Circuit('Counter(2)',
                ['cin'],
                ['cnt0', 'cnt1', 'cout'])
    c.add_inst( Counter1Bit(), ['cin'],
                ['cnt0', 'cout0'])
    c.add_inst( Counter1Bit(), ['cout0'],
                ['cnt1', 'cout'])
    return c
Simulate As Software
Walkthrough

Counter(1)

Counter(2)
Simulate As Software

Walkthrough

1 1 ? ? ?
Simulate As Software

Walkthrough

\begin{align*}
\text{Cin} & \quad \text{cin} & \quad \text{sum} & \quad \text{cnt} & \quad \text{cout} \\
1 & \quad 1 & \quad ? & \quad ? & \quad ?
\end{align*}

Counter(1)

Counter(2)
Simulate As Software

Walkthrough

\[
\begin{align*}
&\text{Cin} \quad \text{cin} \quad \text{sum} \quad \text{cnt} \quad \text{cout} \\
&1 \quad 1 \quad ? \quad ? \quad ?
\end{align*}
\]

\[
\begin{align*}
&? \quad 0
\end{align*}
\]
Simulate As Software

Walkthrough

Cin  cin  sum  cnt  cout
1    1    ?    ?    ?

0    0
Simulate As Software

Uh Oh!

We don't have a well-defined value to save in the Reg.
Simulate Via Fix-Point

Cin  Cin  sum  cnt  cout
1    1    ?    ?    ?
?    0    ?    0

1
0

Counter(1)

Counter(2)
Simulate Via Fix-Point

Q1: Does this process terminate?
Q2: Does the result depend on execution order?
Q3: What is “?”
Abstract Interpretation
Abstract Interpretation

History

• Approach to static analysis of programs proposed by Cousot & Cousot at POPL in 1977

• Generalizes data flow analyses of programs
Abstract Interpretation

Basic Concepts

• Treat a program as a data flow / control flow graph (with values of variables defined at program points between statements)

• For HW: each gate/component = a statement; wires = variables

• Statements/Gates have “concrete semantics” as functions of “concrete values” e.g. \{0,1\} or the integers. This set of values is called a *domain*.

• We are going to *abstract* these domains.
Abstract Interpretation
Abstracting Domains

Concrete Domain

\( \mathbb{Z} \)
Abstract Interpretation
Abstracting Domains

Concrete Domain
$\mathbb{Z}$

Concrete "Lattice"
$P(\mathbb{Z})$

$\{3, \{0, 1\}, \mathbb{Z}, \{0, 1\}\}$

$P(\mathbb{Z})$
Abstract Interpretation

Abstracting Domains

Concrete Domain

\[ \mathcal{Z} \]

Concrete "Lattice"

\[ \mathcal{P}(\mathcal{Z}) \]

\[ \{ \mathcal{Z}, \mathcal{Z}_0, \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3, \mathcal{Z}_4 \} \]

\[ \mathcal{P}(\mathcal{Z}) \rightarrow \text{uncountable} \]

in 1-1 correspondence with infinite seq. of 0s & 1s
Abstract Interpretation
Abstracting Domains

Concrete Domain
\{0, 1, \top\}
\mathcal{P}(\{0, 1, \top\})

Concrete "Lattice"
\mathbb{Z}
\mathcal{P}(\mathbb{Z})

Abstract "Lattice"
e.g. \{0, -, +, \top, \bot\}
A Lattice?

A set $L$ is a lattice when

1. it has a partial order $x \leq y$
2. it has meet "\( \wedge \)" and join "\( \vee \)" operations defined

$$x \wedge y \equiv \min(x, y) \quad x \vee y \equiv \max(x, y)$$
A Lattice?

An example

\[
L = \{ \emptyset, \{0\}, \{0, 3\}, \{0, 13\} \}
\]

for \( x, y \in L \), \( x \leq y \) iff. \( x \subseteq y \)
A Lattice?
An example

\[ L = \{ \emptyset, \{0\}, \{1\}, \emptyset \} \]

For \( x, y \in L \) \( x \leq y \) iff. \( x \subseteq y \)

\[ x \wedge y = x \cap y \quad x \vee y = x \cup y \]

All lattices are Boolean Algebras.
The Concrete Lattice

Given any set $X$, $P(X)$ is a lattice where $\leq$ is $\subseteq$, $\land$ is $\cap$, and $\lor$ is $\cup$. 
Abstract Lattices

An example

\[ \{0, +, -, T, \bot\} \]
Abstract Lattices

An example

\{0, +, -, T, \perp\}

Not every partial order is a lattice

e.g.
The Abstraction Relationship

These maps must preserve \( \leq, \land, \text{ and } \lor \)
The Abstraction Relationship

\[ x \leq y \implies \alpha(x) \leq \alpha(y) \]

\[ \alpha(x \land y) = \alpha(x) \land \alpha(y) \]

(\text{ditto for } \lor)

\[ \alpha : \text{Conc} \rightarrow \text{Abs} \]

“abstraction”

\[ \chi : \text{Abs} \rightarrow \text{Conc} \]

“concretization”

These maps must preserve \( \leq, \land, \text{ and } \lor \)
The Abstraction Relationship

\[ \alpha'(\gamma(\alpha(x))) = x \]
\[ \delta(\alpha'(x)) \geq x \]

\( \alpha' : \text{Conc} \rightarrow \text{Abs} \)
“abstraction”

\( \delta : \text{Abs} \rightarrow \text{Conc} \)
“concretization”

These maps must preserve \( \leq, \wedge, \text{and} \vee \)
A nontrivial Example

\[ \gamma(0) = \{0^3\} \quad \gamma(-) = \{x \in \mathbb{Z} \mid x < 0^3\} \quad \gamma(+) = \{x \in \mathbb{Z} \mid x > 0^3\} \]

\[ \sigma(T) = \mathbb{Z} \quad \sigma(\perp) = \emptyset \]
# Abstracting Functions

**An example**

Let $B = \{0, 1\}$ and $L = \{0, 1, T, F\}$

We know $\text{AND}: B \times B \rightarrow B$.

What is $\text{AND}: L \times L \rightarrow L$?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$\text{AND}(x, y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$0$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
<tr>
<td>$1$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
<tr>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
</tbody>
</table>
Abstracting Functions

An example

Let $B = \{0, 1\}$ and $L = \{0, 1, T, F\}$.

We know $\text{AND} : B \times B \rightarrow B$.

What is $\text{AND} : L \times L \rightarrow L$?

**General Rule:**

1. Concretize
2. Use basic defn.
3. Abstract again

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>AND(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Abstracting Functions

An example

\[ \text{AND}(T, 1) \begin{cases} \{0, 1\} \\ \{1\} \end{cases} \begin{cases} 0 \\ T \end{cases} \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>AND(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

General Rule:
1. Concretize
2. Use basic defn.
3. Abstract again
Abstracting Functions

An example

\[ \text{AND}(T, 1) = \{ (0,1) \} \{ 0 \} T \]
\[ \text{AND}(T, 0) = \{ (0,0) \} \{ 0 \} 0 \]

General Rule:
1. Concretize
2. Use basic defn.
3. Abstract again
Given a choice of Abstraction, (lattice and abstraction function) all other details are determined. They can be mechanically derived.
The Simulation/Interpretation Algorithm Specification

1. Replace All concrete values with abstract values.
2. Use abstracted defns. of functions/statements/gates.
3. Initialize all unknown values to $T$
4. Run all functions to fix-point, merging values with $\Lambda$
The Simulation/Interpretation Algorithm
Guarantees

1. This algorithm will always terminate for finite lattices.
2. There is a unique fix-point solution regardless of which order execution proceeds in.
Switch to Ross for example set 1
What About Verilog’s “Z”?
Our Method for Untangling Semantics

Be More Concrete!
Our Method for Untangling Semantics

Be More Concrete!

Q: In which specific situations is the Z value necessary? & What is special about those situations?
Tri-State Buffers & Buses
Tri-State Buffers & Buses

"As a function"

<table>
<thead>
<tr>
<th>en</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*</td>
<td>&quot;Z&quot;</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Tri-State Buffers & Buses
Be More Concrete!

If we can explain transistors then we can explain the Tri-State
A Transistor
Logical idealization

if $G = 1$, then $S = D$
if $G = 0$, then $S$ and $D$ are disconnected
A Transistor

Functional specification?

If $G = 1$, then $S = D$

If $G = 0$, then $S$ and $D$ are disconnected

Is $D$ a function of $S$ and $G$?

Is $S$ a function of $G$ and $D$?
A Transistor
Relational Specification

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c} G & S & D \\ \hline 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \]
There is no inherent direction of "data flow" in some real circuits
Modification of Abstract Interpretation to Handle Relations

Instead of abstracting functions, we can abstract relations. Applying relations looks like "filtering" values rather than applying a function.
Abstracting a Transistor

\[ \text{Trans} (1, 0, T) \mapsto \{(1, 0, 0)\} \]
\[ \downarrow \]
\[ \lessdot (1, 0, 0) \]

\[ \text{Trans} (T, 0, T) \sim (T, 0, T) \]
Ross Examples 2