Mechanical Approach to Linking Operational Semantics and Algebraic Semantics for Verilog using Maude

Huibiao Zhu[1] Peng Liu[1] Jifeng He[1] Shengchao Qin[2]

[1]Shanghai Key Laboratory of Trustworthy Computing
Software Engineering Institute, East China Normal University

[2]School of Computing, University of Teesside

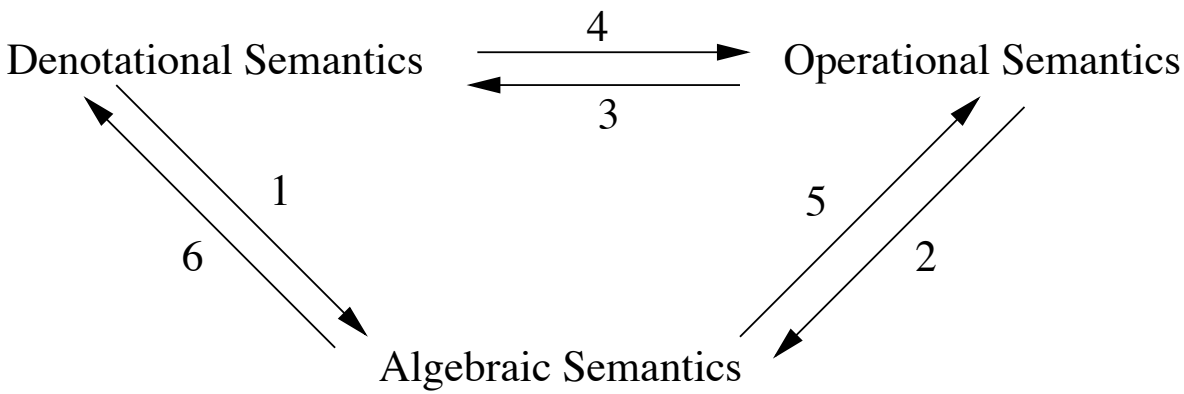
UTP 2012, 27-28 August, 2012

Outline

Outline

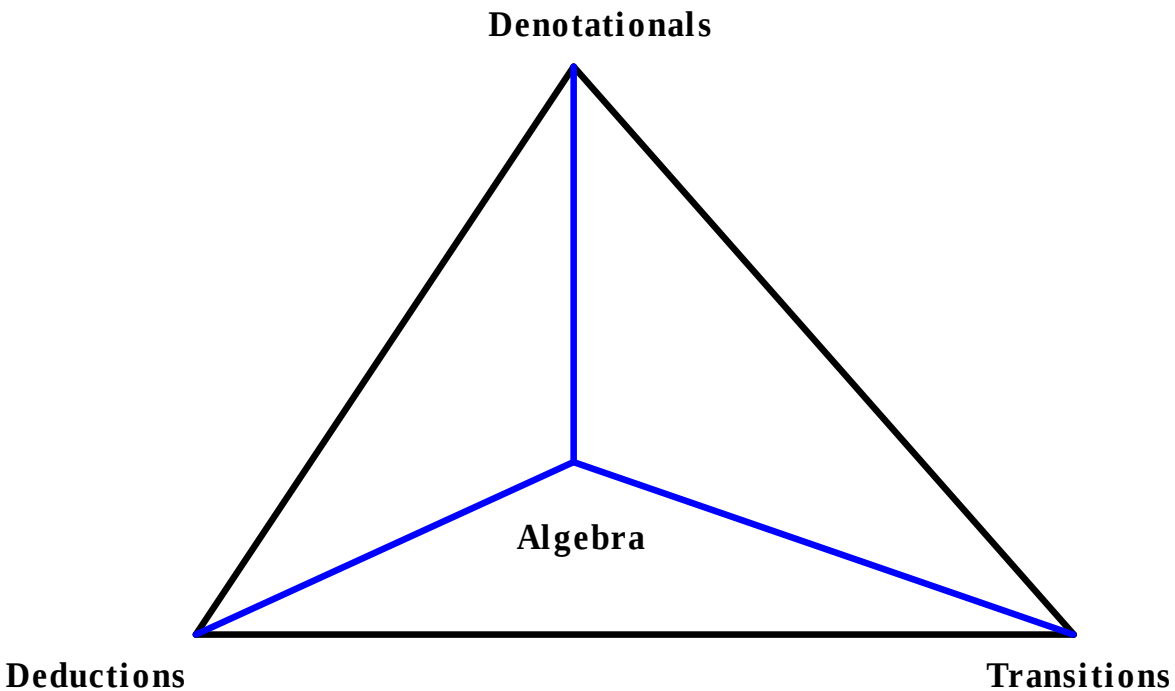
Background

1. **Verilog:** (a) event-driven computation
   (b) shared-variable concurrency
   (c) simulator-based interpretation
2. **Semantic Study:**



Related Work

- Recently, Hoare proposed the challenging research for the semantic linking between **algebra**, **denotations**, **transitions** and **deductions** (in Meeting 52 of WG 2.3).



Contribution

How can we guarantee the consistency between operational semantics and algebraic semantics for Verilog?

(1) **Approach:**

- Deriving operational semantics from algebraic semantics mechanically
- Using equational and rewriting logic system Maude to support the mechanical approach

(2) **Methodology:**

- Further Algebraic Laws with Location Status
- Head Normal Form
- Deriving Operational Semantics from Algebraic Semantics
- Also Mechanizing the Derived Operational Semantics

The Syntax of Verilog

The Syntax of Verilog:

$$P ::= PC \mid P\ ;\ P \mid \textbf{if}\ b\ \textbf{then}\ P\ \textbf{else}\ P \mid \textbf{while}\ b\ \textbf{do}\ P \mid c\ P \mid P \parallel P$$

- PC ranges over primitive commands.

  $$PC ::= x := e \mid \textbf{Skip} \mid @(x := e)$$

- $P\ ;\ Q$ is the sequential composition.
- $P \parallel Q$ is the parallel composition, where its mechanism is an interleaving shared-variable concurrency model.
- $c\ P$ denotes a timing control statement, and c is a time control used for scheduling.

  $$c ::= \#n \mid @(g)$$

  where, $g ::= \eta \mid g\ or\ g \mid g\ and\ g \mid g\ and\ \neg g$

  $\eta ::= v \mid \uparrow v \mid \downarrow v, \quad n \geq 1$

Equational and Rewriting Logic System Maude

1. Rewriting logic has been introduced as a general semantic and logical framework.
2. In Maude, the fundamental unit can be a functional module or a system module, containing the declarations of importing options, sorts, subsorts, operations, equations and rules (only in system modules).

```
fmod PEANO-NATURAL is including BOOL .
   sorts NzNat Nat .
   subsort NzNat < Nat .
   op 0 :  -> Nat [ctor] .
   op s(_) :  Nat -> NzNat [ctor] .
   op _+_ :  Nat Nat -> Nat .
   vars N M : Nat .
   eq 0 + N = N .
   eq s(M) + N = s(M + N) .
   op _>_ :  Nat Nat -> Bool .
   eq s(N) > 0 = true .
   ceq s(N) > s(M) = true if N > M .
   eq N > M = false [owise] .
endfm
```

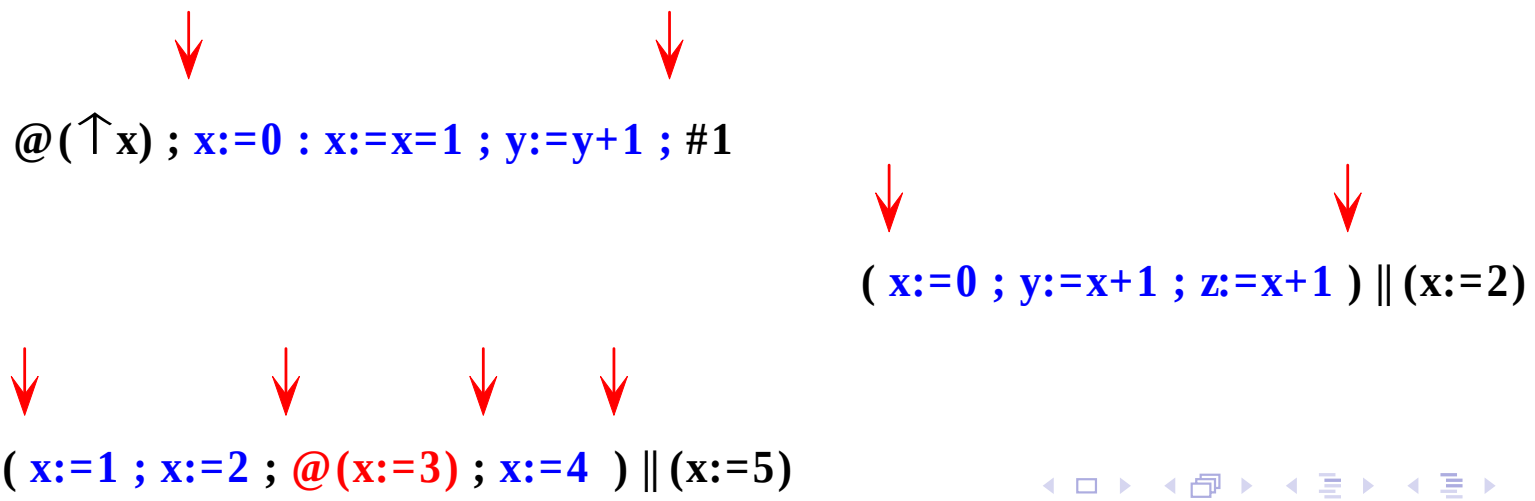
```
mod MY-LIST is including PEANO-NATURAL .
   sorts Elt List .
   subsort Nat < Elt < List .
   op null :  -> List [ctor] .
   op _ _ :  List List -> List [ctor assoc id:  null] .
   vars A B : Elt .
   crl [swap] :  A B => B A if A > B .
endm
```

Outline

Uninterrupted Piece and Atomic Action

Examples:

1. For $@(\uparrow x)\ ;\ x := 0\ ;\ x := x+1\ ;\ y := x+1\ ;\ \#1$. The environment can be scheduled at the $\downarrow$ places shown below.
2. Let $P =_{df} (x := 0\ ;\ y := x+1\ ;\ z := x+1)$ and $Q =_{df} x := 2$. Consider $P \parallel Q$. For process P, process Q can only be scheduled at the $\downarrow$ places.
3. The assignment guard $@(x := e)$ is an atomic action.
4. For $(x := 1\ ;\ x := 2\ ;\ @(x := 3)\ ;\ x := 4)||(x := 5)$, $x := 5$ can only be scheduled at the $\downarrow$ places.

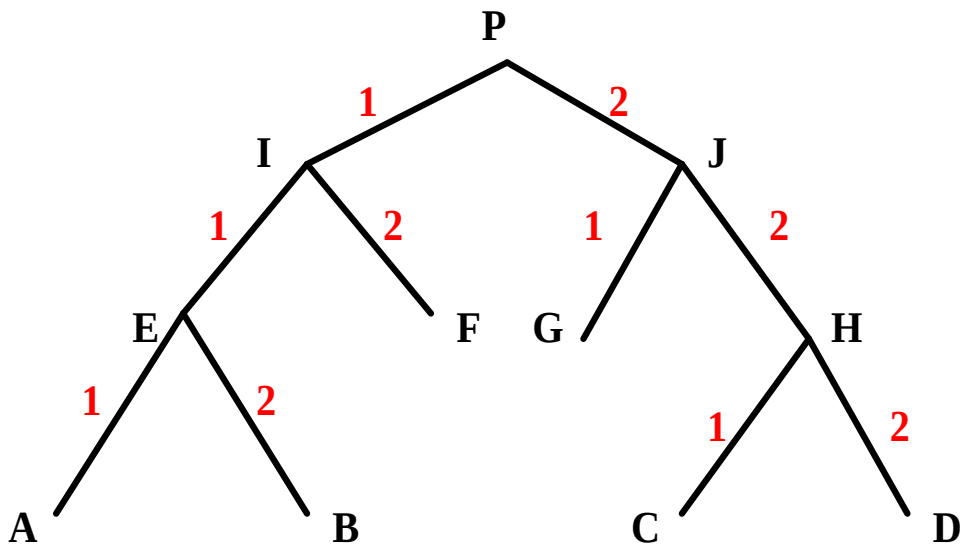


Locality of Instantaneous Action

Example 1: Let $P =_{df} x := 0 \ ; \ \#1 \ ; \ x := 1 \ ; \ @(\uparrow x)$. The instantaneous actions $x := 0$ and $x := 1$ are both due to process P itself. Therefore, the sequence for indexing their contribution in P is $\langle\rangle$.

Example 2: Let $P = I \parallel J, \quad I = E \parallel F, \quad J = G \parallel H,$
$E = A \parallel B, \quad H = C \parallel D.$

where, the outside structure of processes A, B, F, G, C and D is not the parallel composition. Below is the graph that illustrates the index sequence of the instantaneous actions.



Locality of Instantaneous Action

Now we define concept of location status for a program, which is one of the following three forms:

1. *index*: it can be $\langle\rangle$ or a non-empty thread sequence. It indicates an instantaneous action is due to which exact component of a parallel process.
2. 0: it indicates the termination of an atomic action.
3. *null*: it indicates a process is at a state where the environment has a chance to perform its instantaneous action.

Example 2 (continued):

Let $A = x := 1 \; ; \; x := x + 1 \; ; \; x := x + 2$
$B = y := 1 \; ; \; y := y + 1 \; ; \; y := y + 2$
$C = z := 1 \; ; \; z := z + 1 \; ; \; z := z + 2$
$D = u := 1 \; ; \; u := u + 1 \; ; \; u := u + 2$
$F = v := 1 \; ; \; v := v + 1 \; ; \; v := v + 2$
$G = w := 1 \; ; \; w := w + 1 \; ; \; w := w + 2$

Consider the location status for P.

Guarded Choice: Definition

Guarded choice can be formalized with location status (i.e., *tag*), as defined below.

Definition:

(1) $h\ (P,\ tag)$ is a guarded component if it can be one of the forms below. Here, b is a Boolean condition and *index* can be $\langle\rangle$ or a non-empty thread sequence.

$$b\&(x := e)\ (P,\ index), \qquad b\&(x := e)\ (P,\ 0),$$

$$@(g)\ (P,\ null), \qquad \#1\ (P,\ null)$$

(2) $[\!]\{h_1\ (P_1,\ tag_1),\ \ldots,\ h_n\ (P_n,\ tag_n)\}$ is a guarded choice if every element $h_i\ (P_i,\ tag_i)$ is a guarded component.

Guarded Choice: Five Types

The first type of guarded choice is composed of a set of assignment components.

(1) $[\!]_{i \in I}\{b_i \& (x_i := e_i)\ (P_i,\ tag_i)\}$

The second type of guarded choice is only composed of a set of event guard components.

(2) $[\!]_{i \in I}\{@(\eta_i)\ (P_i,\ null)\}$

The third type is composed of one time-delay component.

(3) $[\!]\{\#1\ (P,\ null)\}$

The fourth type of guarded choice is composed of a set of assignment components and a set of event guard components.

(4) $[\!]_{i \in I}\{b_i \& (x_i := e_i)\ (P_i,\ tag_i)\}\ [\!]\ [\!]_{j \in J}\{@(\eta_j)\ (Q_j,\ null)\}$

The fifth type of guarded choice is composed of a set of event guard components and a time delay component.

(5) $[\!]_{i \in I}\{@(\eta_i)\ (P_i,\ null)\}\ [\!]\ \{\#1\ (Q,\ null)\}$

Guarded Choice: Mechanizing in Maude

*GComponent*1: $b\&(x := e)\ (P, \mathit{index})$ and $b\&(x := e)\ (P, 0)$

*GComponent*2: $@(g)\ (P, \mathit{null})$

*GComponent*3: $\#1\ (P, \mathit{null})$

```
fmod GUARDED-COMPONENT is pr VERILOG-PROGRAM .
     pr CONFIG .

     sorts GComponent1 GComponent2 GComponent3 GComponent .
     subsort GComponent1 GComponent2 GComponent3 < GComponent .

     sorts AssignmentGuard GuardPostfix GuardPostfix1 GuardPostfix2
           GuardPostfix3 .
     subsort GuardPostfix1 GuardPostfix2 GuardPostfix3 < GuardPostfix .

     op _&(_) :  BoolExp Assignment -> AssignmentGuard [ctor] .
     op `(_,_`) :  Program Index -> GuardPostfix1 [ctor] .
     op `(_,_`) :  Program EndPoint -> GuardPostfix2 [ctor] .
     op `(_,_`) :  Program Null -> GuardPostfix3 [ctor] .

     op __ :  AssignmentGuard GuardPostfix1 -> GComponent1 [ctor] .
     op __ :  AssignmentGuard GuardPostfix2 -> GComponent1 [ctor] .
     op __ :  EventGuard GuardPostfix3 -> GComponent2 [ctor] .
     op __ :  TimeControl GuardPostfix3 -> GComponent3 [ctor] .
endfm
```

Generating Algebraic Laws (1)

The Expansion Laws for Parallel Composition:

1. We summarize that there are five typical parallel expansion forms, described as $comp1$, $comp2$, $\cdots$, and $comp5$.
2. We use the notation $P =_{tag} Q$ to stand for $(P, tag) = (Q, tag)$, indicating that process P and Q are equivalent at location status tag.

Case 1: One parallel component is in the form of assignment guarded choice.This case can be expressed in "**comp1**".

For example, assume $P =_{null} [\!]_{i \in I}\{b_i \& (x_i := e_i)\ (P_i,\ tag_i)\}$ and

Q be any process.

Then, $[\!]_{i \in I}\{b_i \& (x_i := e_i)\ \mathbf{par1}(P_i, Q, 1, tag_i))\}$ is one part of the parallel expansion of $P \parallel Q$.

```
op comp1(_,_,_) :  HGCType1 Program Index -> HGCType1 .
eq comp1({b &(x := e)(P1,tag1)},Q,<1>) = {b &(x := e)par1(P1,Q,<1>,tag1)} .
eq comp1({b &(x := e)(Q1,tag1)},P,<2>) = {b &(x := e)par1(P,Q1,<2>,tag1)} .
eq comp1({h1 Post1} [] hgc',P,i) = comp1({h1 Post1},P,i) [] comp1(hgc',P,i) .
```

Generating Algebraic Laws (2)

Case 2: One parallel component is in the form of event-guarded choice and another parallel component does not have event-guard initially. This case is expressed using "**comp2**".

For example, assume $Q =_{null} [\!]_{j \in J}\{@(\eta_j)\ (Q_j, null)\}$ and
P does not have event-guard initially.

Then, $[\!]_{j \in J}\{@(\eta_j)\ \mathbf{par}(P, Q_j)\}$ is one part of the parallel expansion.

Case 3: Both of the two parallel components are in the form of event-guarded choice. There are three types of the triggered cases, defining in **comp3**, **comp4** and **comp5**.

For example, Assume $P =_{null} [\!]_{i \in I}\{@(\eta_i)\ (P_i, null)\}$ and
$Q =_{null} [\!]_{j \in J}\{@(\xi_j)\ (Q_j, null)\}$

Then, $[\!]_{i \in I}\{@(\eta_i\ and\ \neg\xi)\ \mathbf{par}(P_i, Q)\}$ (1)

and $[\!]_{j \in J}\{@(\xi_j\ and\ \neg\eta)\ \mathbf{par}(P, Q_j)\}$ (2)

and $[\!]_{i \in I \wedge j \in J}\{@(\eta_i\ and\ \xi_j)\ \mathbf{par}(P_i, Q_j)\}$ (3)

are the three firing cases for $P \parallel Q$. Here $\eta = or_{i \in I}\{\eta_i\}$ and $\xi = or_{j \in J}\{\xi_i\}$.

Outline

Generating Head Normal Form (1)

Aim: For support the derivation of operational semantics from algebraic semantics, we introduce the concept of head normal form.

Form: We use the notation $HF((P, tag))$ to stand for the head normal form of process P at the location status tag.

1. Sequential Constructs

```
eq   HF(x := e,tag) = ({t &(x := e)(nil,<>)},tag) .
eq    HF(Skip,tag) = ({t &(Skip)(nil,<>)},tag) .
eq   HF(@(x := e),tag) = ({t &(x := e)(nil,0)},tag) .
eq   HF(@(g),tag) = ({@(g)(nil,null)},tag) .
eq   HF(# 1,tag) = ({# 1(nil,null)},tag) .
eq   HF(# n,tag) = ({# 1(# (n - 1),null)},tag) .
ceq  HF(P ; Q,tag) = (seq(T,Q),tag) if (T,tag) := HF(P,tag) .
ceq  HF(P ; Q,tag) = (seq(T,Q),tag) if (T,tag) := HF(P,tag) .
ceq  HF(if b then P else Q,tag) = ({b &(Skip)(P,<>)} []
                                   {~ b &(Skip)(Q,<>}},tag) .
ceq  HF(while b do P,tag) = ({b &(Skip)(P ; while b do P,<>)} []
                             {~ b &(Skip)(nil,<>)},tag) .
```

Generating Head Normal Form (2)

2. Parallel Composition:

For parallel process $P \parallel Q$, its location status can be *null*, 0 and *seq*.

Case 1: Firstly we consider the head normal form of $P \parallel Q$ at the location status *null*.

(a) We first consider the case that two parallel components of a parallel process are of the first three types.

```
ceq    HF(P || Q,null)
       = (comp1(hgct11,Q,<1>) [] comp1(hgct12,P,<2>) , null)
      if (hgct11,null) := HF(P,null) /\ (hgct12,null) := HF(Q,null) .

ceq    HF(P || Q,null)
       = (comp1(hgct1,Q,<1>) [] comp2(hgct2,P,<2>) , null)
      if (hgct1,null) := HF(P,null) /\ (hgct2,null) := HF(Q,null) .

          ......                          ......
```

Generating Head Normal Form (3)

(b) If a process is in the form of the fourth type of guarded choice (or the fifth type of guarded choice), it can be composed in parallel with any process (i.e., in the form of any type of guarded choice).

Case 2: Now we consider other cases for the location status of a parallel process.

```
ceq   HF(P || Q,<1> ^ index)
      = (comp1({b &(x := e)(P1,index)},Q,<1>), <1> ^ index)
      if ({b &(x := e)(P1,index)},index) := HF(P,index) .

ceq   HF(Q || P,<2> ^ index) =
      (comp1({b &(x := e)(P1,index)},Q,<2>), <2> ^ index)
      if ({b &(x := e)(P1,index)},index) := HF(P,index) .

            ......                          ......
```

Outline

Transition Types (1)

1. **Three Types of Configurations**

   $\langle P, \sigma, \emptyset \rangle$, $\langle P, \sigma, \sigma', 1, seq \rangle$, $\langle P, \sigma, \sigma', 0 \rangle$

2. **Three Transitions Types:**

   (1) Instantaneous transition $C \longrightarrow C'$

   (2) Event transition $C \overset{\langle \sigma, \sigma' \rangle}{\longrightarrow} C'$

   (3) Time advance transition $C \overset{1}{\longrightarrow} C'$.

3. **Rules:**

1. Instantaneous transition

   $\mathbf{T}_1$ A process can perform its first instantaneous action of an atomic action.

   $$\langle P, \sigma, \emptyset \rangle \longrightarrow \langle P', \sigma, \sigma', 1, seq \rangle$$

   $\mathbf{T}_2$ A process can continue its following instantaneous action in an atomic action.

   $$\langle P, \sigma, \sigma', 1, seq \rangle \longrightarrow \langle P', \sigma, \sigma'', 1, seq \rangle$$

   $\mathbf{T}_3$ A process completes an instantaneous section.

   $$\langle P, \sigma, \sigma', 1, seq \rangle \longrightarrow \langle P, \sigma, \sigma', 0 \rangle$$

   $\mathbf{T}_4$ A process executes an assignment guard.

   $$\langle P, \sigma, \emptyset \rangle \longrightarrow \langle P', \sigma, \sigma', 0 \rangle$$

Transition Types (2)

1. Event transition

   $\mathbf{T}_5$ (1) A transition can be fired by the atomic action that has just completed.

   $$\langle P,\ \sigma,\ \sigma',\ 0\rangle \xrightarrow{\langle\sigma,\sigma'\rangle} \langle P',\ \sigma',\ \emptyset\rangle$$

   (2) A transition can be fired by the action of its environment.

   $$\langle P,\ \sigma,\ \emptyset\rangle \xrightarrow{\langle\sigma,\sigma'\rangle} \langle P',\ \sigma',\ \emptyset\rangle$$

2. Time advance transition

   $\mathbf{T}_6$ A process that cannot do anything else will allow time to advance. Time advances in unit steps.

   $$\langle P,\ \sigma,\ \emptyset\rangle \xrightarrow{1} \langle P',\ \sigma,\ \emptyset\rangle$$

Deriving Oper Seman from Algebraic Semantics (1)

Methodology:

We give the derivation strategy, which is based on the head normal of each program. For every program, its location status can be *null*, $\langle\rangle$ or *seq*.

(1.a) If $HF((P, null)) = (\ [\!]_{i\in I}\{b_i \& @(x_i := e_i)\ (P_i, tag_i)\},\ \ null\ )$, then P can perform transitions at states $\langle P, \sigma, \emptyset\rangle$ and $\langle P, \sigma, \sigma', 0\rangle$.

```
crl [1.a1] :  .  < P,env ,# > => < Pi ,env ,env <- (x , e),1,tag >
          if (hgct1,null) := HF(P,null) /\
             (hgc [] {b &(x := e)(Pi,tag)} [] hgc', null)
               := (hgct1,null) /\ b[env] /\ tag =/= 0 .

crl [1.a2] :  .  < P , env , # > => < Pi , env , env <- (x , e) , 0 >
          if (hgct1,null) := HF(P,null) /\
             (hgc [] {b &(x := e)(Pi,tag)} [] hgc', null)
               := (hgct1,null) /\ b[env] /\ tag == 0 .

crl [1.a3] :  .  < P , env , env' , 0 > => < P , env' , # >
          if (hgct1,null) := HF(P,null) .
```

Deriving Oper Seman from Algebraic Semantics (2)

(1.b) If $HF((P,\ null)) = (\ [\!]_{i\in I}\{@(\eta_i)\ (P_i,\ null)\},\ \ null\ )$,
then P can perform transitions at states $\langle P, \sigma, \emptyset\rangle$ and $\langle P, \sigma, \sigma', 0\rangle$.

```
crl [1.b1] :  .  < P , env , env' , 0 > => < P , env' , # >
           if (hgct2,null) := HF(P,null) /\ not fire(guard(hgct2))(env,env') .
crl [1.b2] :  .  < P , env , env' , 0 > => < Pi , env' , # >
           if (hgct2,null) := HF(P,null) /\ (hgc [] {@(g)(Pi,null)} [] hgc' , null) :=
           (hgct2,null) /\ fire(g)(env,env') .
crl [1.b3] :  .  < P , env , # > => < P , env , # >
           if (hgct2,null) := HF(P,null) .
```

(1.c) If $HF((P, null)) = (\ [\!]\{\#1\ (R, null)\},\ \ null)$,
then P can perform transitions at states $\langle P, \sigma, \emptyset\rangle$ and $\langle P, \sigma, \sigma', 0\rangle$.

```
crl [1.c1] :  < P , env , env' , 0 > => < P , env' , # >
           if ({# 1(R,null)} , null) := HF(P,null) .
crl [1.c2] :  < P , env , # > => < R , env , # >
           if ({# 1(R,null)} , null) := HF(P,null) .
```

Deriving Oper Seman from Algebraic Semantics (3)

(1.d) If $HF((P, null)) = (\ [\!]_{i \in I}\{b_i \& @(x_i := e_i)\ (P_i,\ tag_i)\}\ [\!]\ [\!]_{j \in J}\{@(\eta_j)\ (R_j, null)\},$ $null\ )$,
then P can perform transitions at states $\langle P, \sigma, \emptyset \rangle$ and $\langle P, \sigma, \sigma', 0 \rangle$.

```
crl [1.d1] : . < P , env , # > => < Pi , env , env <- (x , e) , 1 , tag >
        if (hgct1 [] hgct2,null) := HF(P,null) /\ (hgc [] {b &(x := e)(Pi,tag)} [] hgc',
        null):= (hgct1,null) /\ b[env] /\ tag =/= 0 .
crl [1.d1'] : . ···
crl [1.d2] : . ···
crl [1.d3] : . ···
crl [1.d4] : . ···
```

(1.e) If $HF((P,\ null)) = (\ [\!]_{i \in I}\{@(\eta_i)\ (P_i,\ null)\}\ [\!]\ \{\#1\ (R,\ null)\},\ \ null\ )$,
then P can perform transitions at states $\langle P, \sigma, \emptyset \rangle$ and $\langle P, \sigma, \sigma', 0 \rangle$.

```
crl [1.e1] : . ···
crl [1.e2] : . ···
crl [1.e3] : . ···
```

Deriving Oper Seman from Algebraic Semantics (4)

(2.a) If $HF((P, seq)) = (\ [\!]_{i \in I}\{b\&(x_i := e_i)\ (P_i, seq)\},\ \ seq\ )$,
then P can perform transitions at state $\langle P, \sigma, \sigma', 1, seq \rangle$.

crl [2.a]:
```
. < P ,env ,env',1,index > => < Pi,env,env <- (x,e),1,index >
    if (hgct1,index) := HF(P,index) /\
       (hgc [] {b &(x := e)(Pi,index)} [] hgc', index)
         := (hgct1,index) /\ b[env] .
```

(2.b) If $HF((P, seq)) = (\ gc,\ \ seq\ )\}$, and hc has component
$b\&(x := e\ (P', 0)$
then P can perform transitions at state $\langle P, \sigma, \sigma', 1, seq \rangle$.

```
.....
.....
```

(3.a) If $HF((P, \langle\rangle)) = (\ [\!]_{i \in I}\{g_i\ (P_i, tag_i)\},\ \ \langle\rangle\ )$
and $\forall i \in I \bullet tag_i \neq \langle\rangle$,
then P can perform transitions at state $\langle P, \sigma, \sigma', 1, \langle\rangle \rangle$.

```
.....
.....
```

Outline

Mechanizing Derived Operational Semantics (1)

1. Aim:

1. Based on the derivation strategy, we can derive the full set of operational semantics for Verilog as theorems by strict proof.
2. Now we apply Maude in mechanizing the derived operational semantics.
3. We can prove the equivalence between the derivation strategy and the derived operational semantics theoretically. Our mechanical approach can support this equivalence.

2. Mechanizing Derived Operational Semantics

(a) Assignment

```
rl : . < x := e , env , # > => < nil , env , env <- (x , e) , 1 , <> > .
rl : . < x := e , env , env' , 1 , <> > => < nil , env , env <- (x , e) , 1 , <> > .
rl : . < x := e , env , env' , 0 > => < x := e , env' , # > .
rl : . < x := e , env , # > => < x := e , env' , # > [nonexec] .
```

Mechanizing Derived Operational Semantics (2)

(b) Parallel Composition

If one of the two parallel parts of a Verilog program can perform the first instantaneous action of an atomic action, then the whole process can also make this transition.

```
crl :  .  < P || Q , env , # > => < par(nil,Q) , env , env' , 0 >
    if .  < P , env , # > => < nil , env , env' , 1 , seq > .
              ......                    ......

crl :  .  < P || Q , env , # > => < par(P',Q) , env , env' , 1 , <1> ^ seq >
    if .  < P , env , # > => < P' , env , env' , 1 , seq > /\ P' =/= nil .
              ......                    ......
```

If one of the two parallel parts of a Verilog program continues to perform the instantaneous action of an atomic action, then the whole process can also make this transition.

```
crl :  .  < P || Q , env , env' , 1 , <1> ^ seq > => < par(P',Q) , env , env'',1,<1> ^ seq >
    if .  < P , env , env' , 1 , seq > => < P' , env , env'' , 1 , seq > /\ P' =/= nil .
              ......                    ......
```

and other cases

Outline

Conclusion and Future Work

1. Conclusion:

- We studied the derivation of deriving the operational semantics from the algebraic semantics for Verilog. The equational and rewriting logic system Maude ia applied.
- We have given the algebraic laws. We introduced the concept of head normal form for every program.
- We have given the definition of the derivation strategy for deriving operational semantics from algebraic semantics. A transition system is derived via the derivation strategy.

2. Future Work:

- Exploring further linking theories for Verilog semantics
- Studying the mechanical approach to the derivation of denotational semantics from algebraic semantics

Thank you very much!