Gordon J. Pace

The Semantics of Verilog Using Transition System Combinators

Verimag, Grenoble, France

(Chalmers University of Technology, Gothenburg, Sweden)

Motivation

- Verilog *synthesis semantics* (relatively) well formally modeled
- The *simulation semantics* are not

...but ...

- testing through simulation is widely used
- and thus synthesis semantics should correspond to simulation semantics

Why are the simulation semantics so complex?

Resolution of concurrency is based on a complex simulation algorithm

Interleaving

```
initial w=1;w=0;
initial v=w;
initial wait (v) P
```

Synchronous composition

```
initial w=1; #1 w=0;
initial while (b) v=!v; #1 v=w;
```

More on why the simulation semantics are so complex

Order of execution

```
initial w=1; w=0;

initial #0 v=w;
```

```
initial w<=1; w=0;

initial v=w;
```

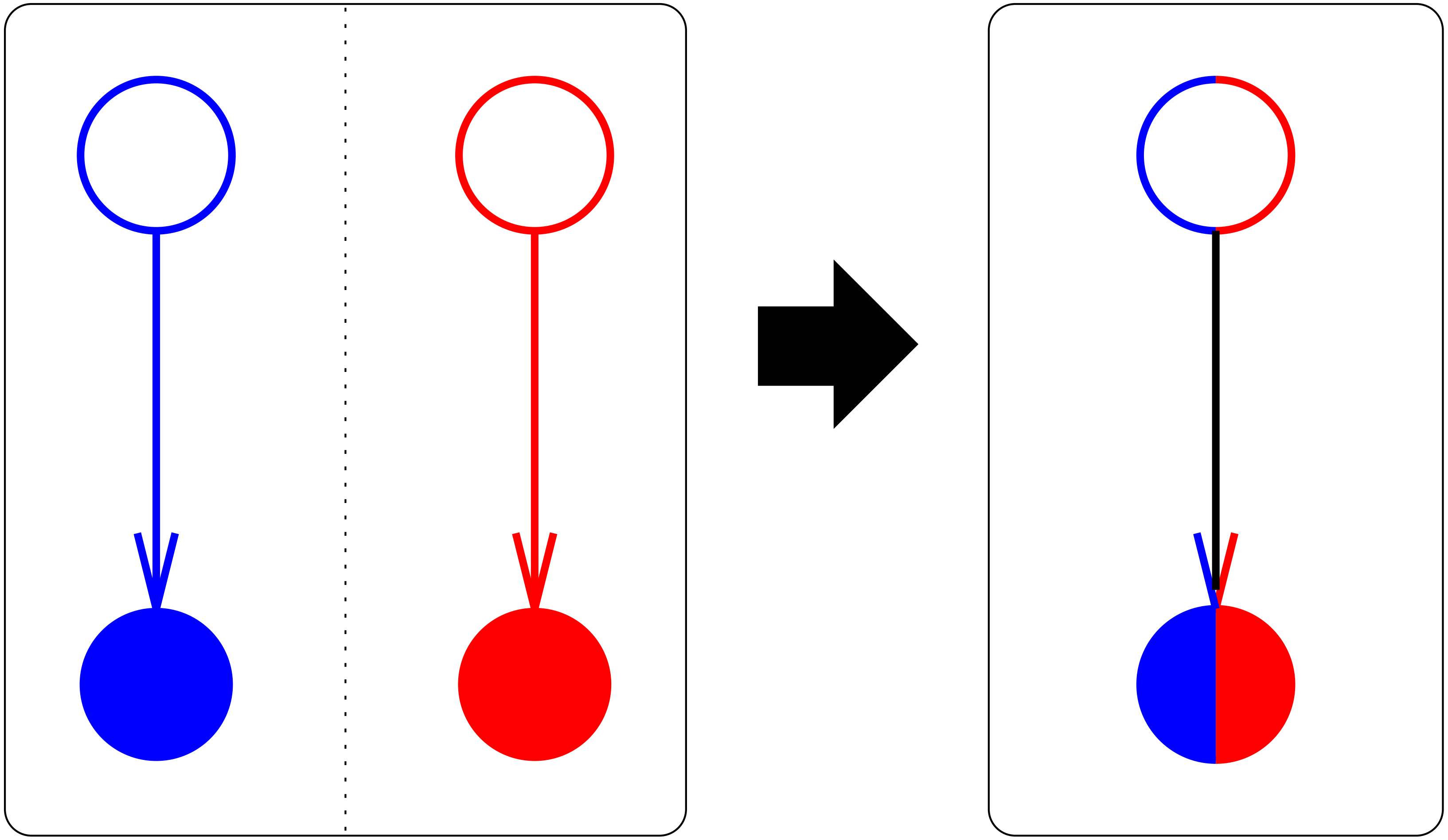
```
initial w<=1; w=0;

initial #0 v=w;
```

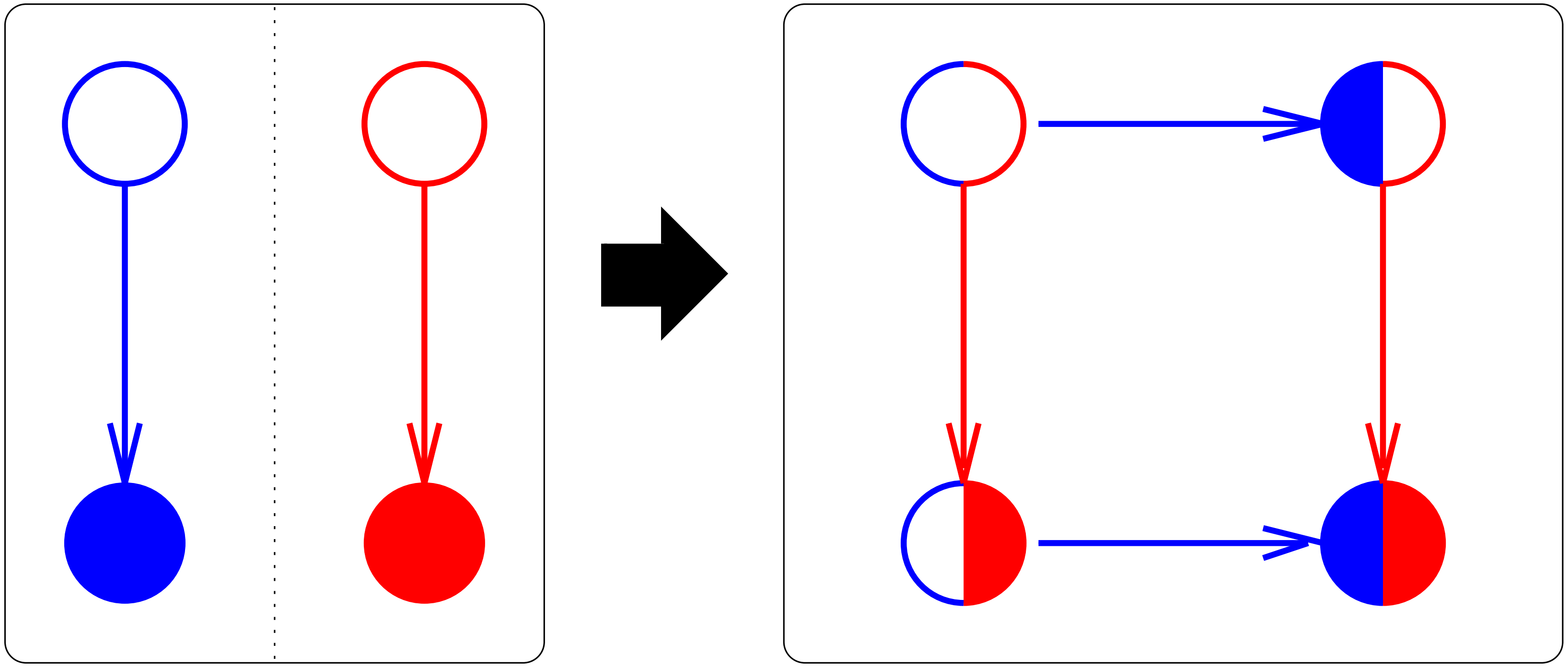
Verilog Simulation Cycle

Aims

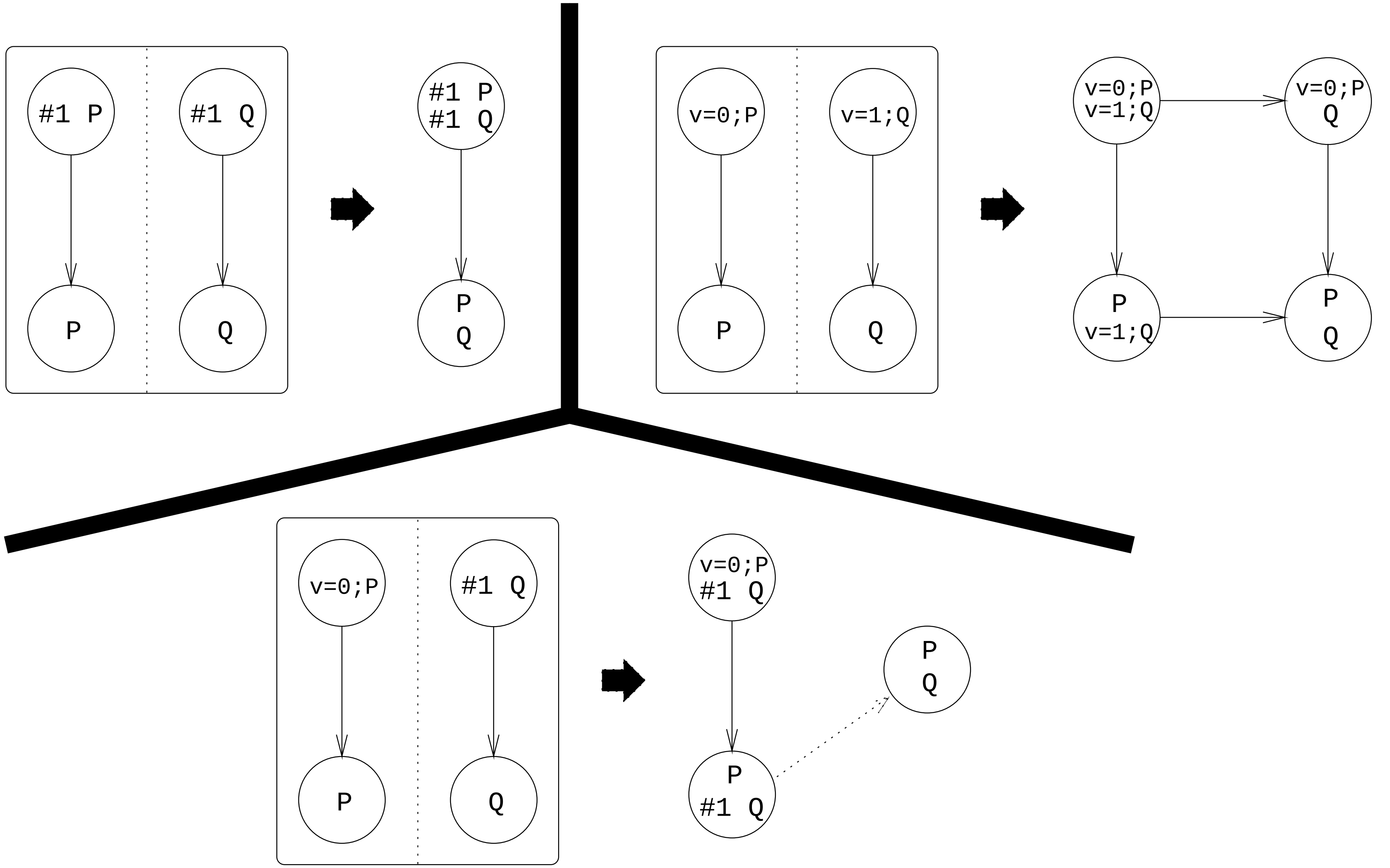
Synchronous composition is well understood . . .



Interleaving composition is also well understood . . .



But what about Verilog's "parallel composition"?



The solution . . .

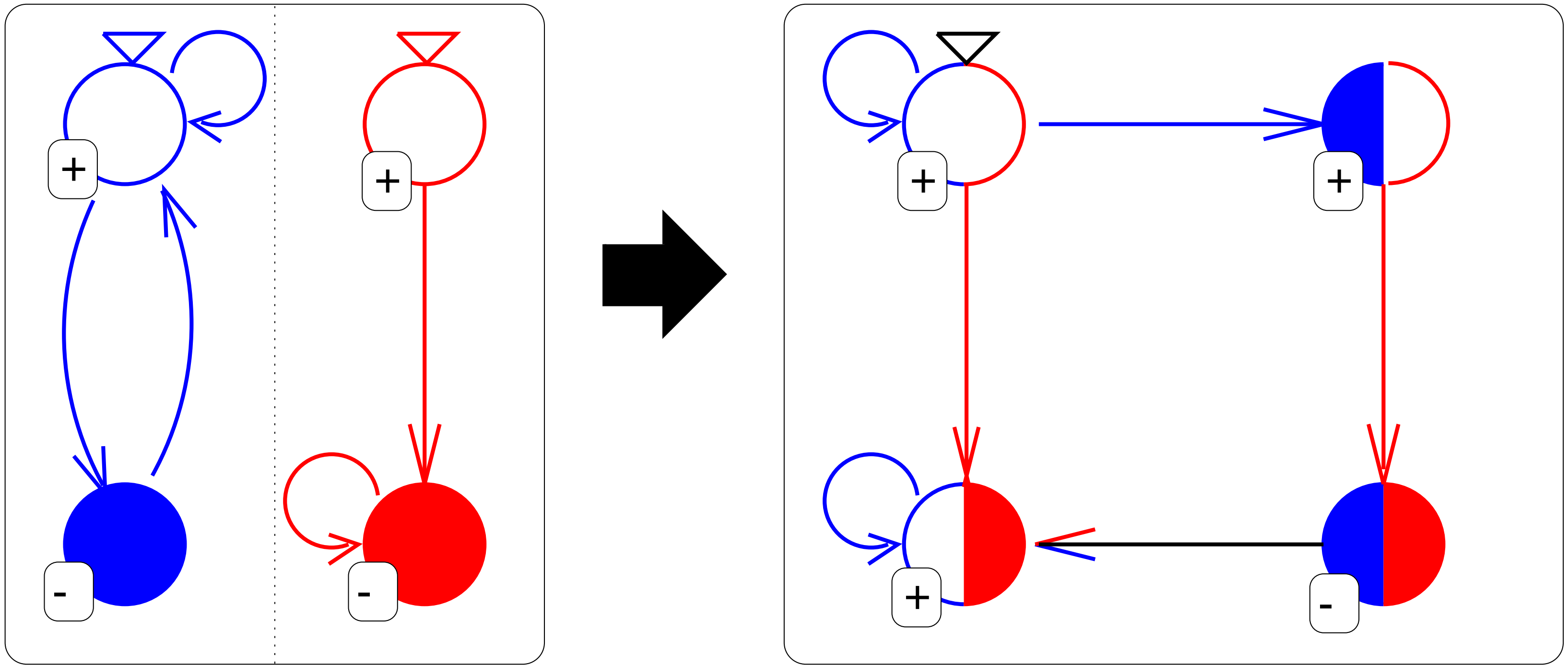
Place the states within an ordered set of *layers*.

. . . and . . .

Say how each layer will compose (synchronous, interleaving, etc).

- Combining two states of different layers allows only transitions of the higher priority state.
- Combining two states belonging to the same layer will allow transitions using the composition method of that layer.

An example



+ Interleave

\- Synchronous composition

The advantages

- Layers allow compositional semantics.
- Layer information is used to combine TS together but ...
- TS semantics left unchanged, meaning ...
- Techniques and tools for standard TS can be used unchanged.

Verilog's parallel composition

Five layers!

$\|$ Guards firing

$|||$ Enabled programs

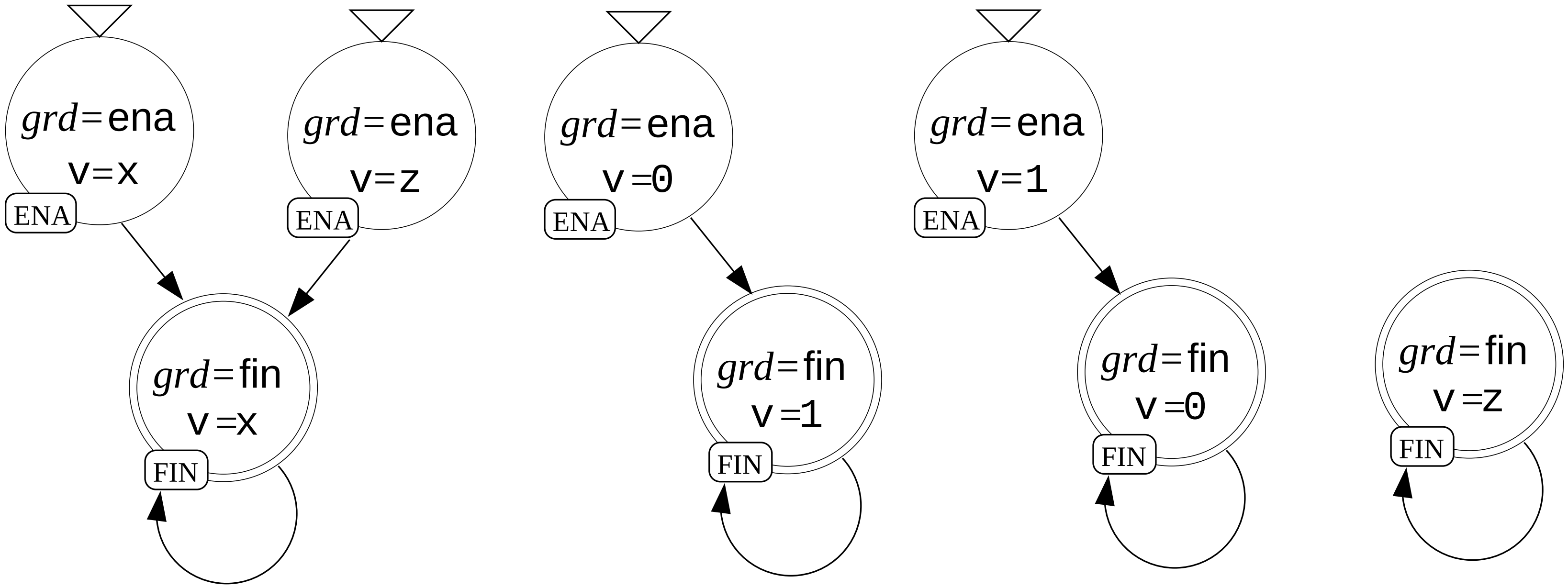
$\|$ Zero-delays

$|||$ Non-blocking updates

$\|$ Time Passing

What about sequential programs?

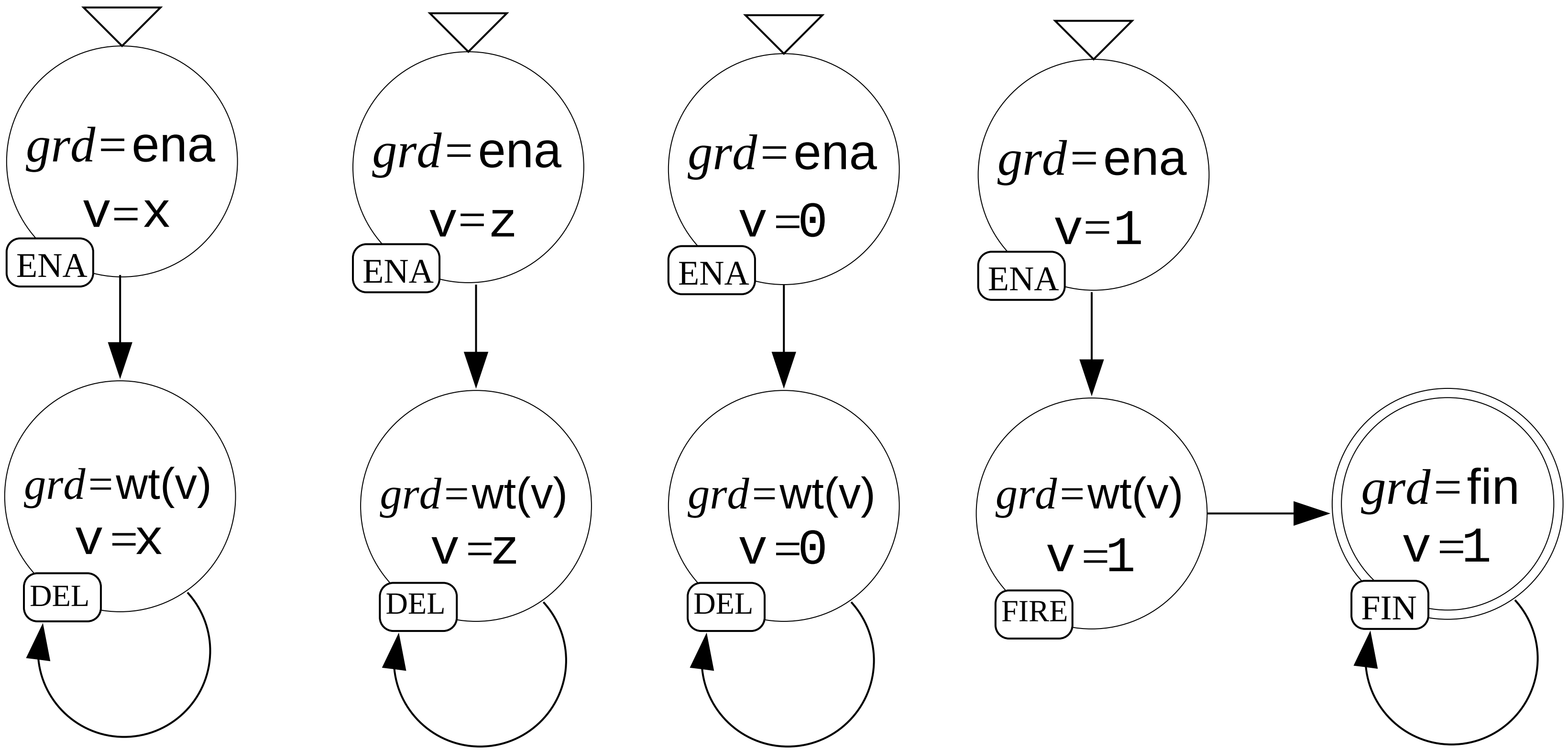
Assignment



`x=!x`

What about sequential programs?

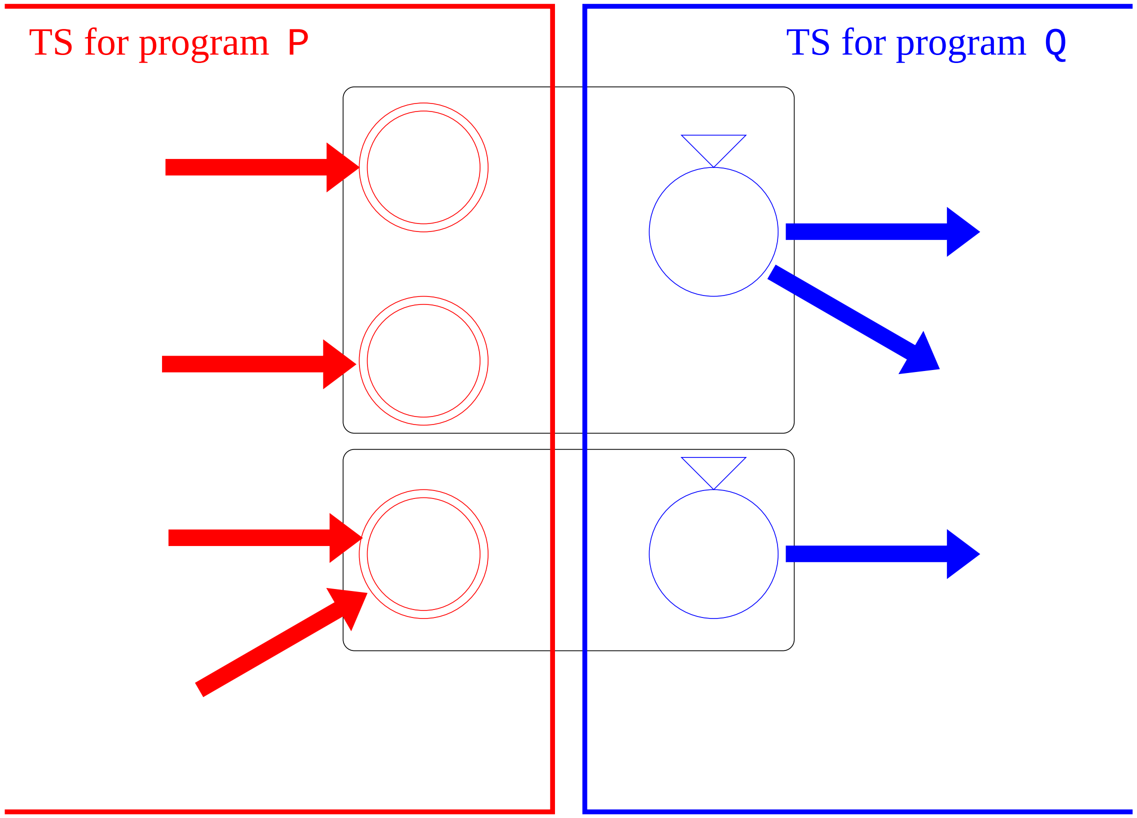
Guards



`wait (v)`

What about sequential programs?

Sequential Composition



P ; Q

`chaos(V)`: The unconstrained program

- Arbitrarily changes variables in set `V`.
- May, or may not terminate.

Theorem: Given a program context C, a program P and a safety property s, then:

$$\frac{\begin{array}{l}\mathrm{output}(P) \subseteq V \\ C(\texttt{chaos(V)}) \vdash \Box s\end{array}}{C(P) \vdash \Box s}$$

Example: Mutual exclusion

$$\texttt{HS(x,y)} \triangleq \texttt{x=1; wait(y);}$$

Does the following program guarantee that `P1` and `Q2` are never executed at the same time (similarly `P2` and `Q1`)?

```
initial x=0; P1; HS(x,y); Q1;

initial y=0; P2; HS(y,x); Q2;
```

How to prove it?

Just prove $\Box(\neg(\texttt{inP1} \wedge \texttt{inQ2}) \wedge \neg(\texttt{inP2} \wedge \texttt{inQ1}))$ for:

```
initial                                initial
  x=0;                                   y=0;
  inP1=1; chaos(V); inP1=0;              inP2=1; chaos(V); inP2=0;
  HS(x,y);                               HS(y,x);
  inQ1=1; chaos(V); inQ1=0;              inQ2=1; chaos(V); inQ2=0;
```

Conclusions

- A Verilog-to-TS (SMV) translator has been written.
- Other small examples have been verified (counters, adders).
- The symbolic analysis gives insight about the complex simulation (eg initialisation is such a pain!).
- Thorough analysis of simulation paths helps by pointing out code dependent on simulator implementations.
- Layered TS can easily be used to express semantics of other simulation based languages (eg VHDL).

Conclusions (but . . .)

- The approach does not scale much up due to asynchronous (interleaving) composition.
- Comparison of simulation, with synthesis semantics still not achieved.