UNIVERSITY^{OF} BIRMINGHAM

Generating hardware from game semantics

Dan R. Ghica

1

UNIVERSITY^{OF} BIRMINGHAM

Generating hardware from game semantics

Dan R. Ghica



```
if(sock < 0) {</pre>
      return -1;
    nemset(&saun, 0, sizeof(struct sockaddr un));
    saun.sun_family = AF_UNIX;
    saun.sun path[0]='\0'
    if((address = getenv("AMFHIBIAN")) != NULL) (
      sprintf(saun.sun_path+1, "fish-%s-%s", getenv("USER"), address);
if( connect(sock, (const struct sockaddr *)&saun, sizeof(struct sockaddr_un)) <>
fprintf(stderr, "amphibian: unable to connect fishsocket: %s\n", strerror(err)
         fprintf(stderr, "amphibian: is your solution running with address %s?\n", add
         close(sock);
        return -1;
       int <u>i</u>
      int \underline{zok} = -1;
       for(i=1; i<=100 && zok == -1; i++) {</pre>
        sprintf(saun.sun_path+1, "fish-%s-%d", getenv("USER"), i);
zok = connect(sock, (const struct sockaddr *)&saun, sizeof(struct sockaddr_un *)
       if (zok == -1) {
        fprintf(stderr, "amphibian_app: unable to find a working fishsocket: %s\n", s
         rn sock;
    else {
  return socket(domain, type, protocol);
/* returns 0 on success, -1 on failure */
                                                t struct sockaddr *<u>serv addr</u>, socklen_t <u>addrl</u>;
int 📩
                        ect(int sockfd,
   f(serv addr != NULL && addrlen != 0) {
                 t struct sockaddr_fish *)serv_addr)-> sfish_family == AF_FISH ) (
    if( ((c
              connection_request req;
      char status buf [20];
      int readbytes;
```

from (programming) languages to circuits



2.2			14	<u> 19</u>))	2.2		1			24
2 2	1		4					۳.			1 2	22			1 K	ht	10
3.3	1	1	15	110	1	1	H.	1 2		1.4	2	5.5	5	1 2	4		
2.2		Η.		12	1	01-12	12	52		辞		2.2	-	1	- K		9 I
	44						59	Ц	r 2 - 2 - 2		1 2	2 12		15.	1		5.5
2				17		1)	0	n 27	1	1.2	1/07	2.7	1	1 2 .	8.98	710	
21		- 1	1					1 ile				2 2	12	127	8	- 11	≚.
12.8			15	ΗĒ	P	1	Na -	1.1	195		1-11	÷.	18			199	
3.3	ré.		N.	12	2	46	1.1	i 🖬 🗉 -	10.10	1.79	1.5	22.0	9 S.D	0 21	i'i '	1.4.1	1.1
			I	- II-	1			8C)	M2-		1.41	SPECIE Principality	1			-2	24
	-11			H			Â.	11115	ti ti t	66	e He	- 21	1.5	.	14		
112	e a	53	- Jai	1 2		21.0	8pr	11 Find	t sur-	i a	1.1	1,0122	n Ti S	11.1	1.13	14 1	18
1	11 11	<u>ار ا</u>	1	12		10	2	1.4	2.	1 10	ા હતુ આવેલા છે.	10.00	alle I -		к		
$\mathbf{a}^{\mathbf{d}}$		공영	5			34			11				MI - 14		1.00	÷.	22
100	4	(2 č			E.			1713	0.5 2		1.73	2 2	171	1.02	1.00	di 1	
	2	1.		18	Ň			10.10					67 (18) 19 (19)				93
3.5	ā.	10	ie ir	с÷.	id.		1	1113	en in	i.e		NIME:	N M			PH	31
3 2	4			10	190	QN P		8, 14	12.1		12	12	2 2	P - 1		I N	1.9
1		12					5	100.0	i÷÷			2 2				JP	8
á.		ā:		5		55	2.1	740	15° a	1.01	18		1.	11	1	Þ	
2.2	5			4.4		۲ī 🖠	1 0	1.80	101 -	1 20	0.15	6.6	1.1	1.1		11	N 9
	-	5	а.	या ।		1.1			ΗP	14		1.5		84 e		ΚL	11
÷.	1		5	4.5	1	ei e	ð 1	i.in	182	101	11	° 511	1.1	ni i i	2 12	100	91 S
94		•	1	12	2	걸날	1	1993	10.15	1041		4 - 40	1.19	(H	120		
	1	Ξ÷			-		40	₽₽	-			100	1.5	(11112) 101125	ange : en a s	通貨業	
÷.	1	-	i i	1.1					15	Ы.	3 A.	15	о в	1135	in a s		
3.5	4	e.	11	14		42	12				1.4		1.00	94	2		
1		H-		11		14		100	₽⊢		1 2			1 2	2 2 1 2	÷Ĥ	
14	4	5	11	3.5	9	19	1	1.005		1	1.2	A - 1	1.1	1.5	20	a 1	ά.
4.4	2	5	2	14		11			(* 1 1 4		1.0	1.1	1.00	100	2 1 0		
		3-		ĩ 1			i.	1.09.0	1000	i fa	1	8.8		1.11	a 12		, ii
•	12	\mathcal{A}_{i}	12	12	12	21 - 1	4.5	1.08	20	140	4 . 1	8 (8)	a ka j	31.F 310	s giù		12
				_										201 C			
	10	5.	11	110	-	8.2		in a s	1.2817	1.1	1.1	184	and s		151		
**	- A	- 3 4	172	23		4 8	19 A	1.4	1 1 1	16.1	1.4	21-31	- 1	al i	$(-1)^{2}$		112
8			2	1.5	2		12					40.44	10.4	111	1.1		
	4			2.5		čik.	- 1		T	IN.	a chi	a^{*} 0^{*}	151	1.1	1000		

basic syntactic control of interference

Identity

$$\Gamma \vdash M : \theta$$
 $x: \theta \vdash x: \theta$
 $\Gamma, x: \theta' \vdash M: \theta$

$$\frac{\Gamma, x: \theta' \vdash M: \theta}{\Gamma \vdash \lambda x.M: \theta' \to \theta} \rightarrow \text{Introduction}$$

$$\frac{\Gamma \vdash F: \theta' \to \theta \quad \Delta \vdash M: \theta'}{\Gamma, \Delta \vdash FM: \theta} \rightarrow \text{Eline}$$

$$\frac{\Gamma \vdash M: \theta' \quad \Gamma \vdash N: \theta}{\Gamma \vdash \langle M, N \rangle: \theta' \times \theta} \times \text{Introduction}$$

– Weakening

)n

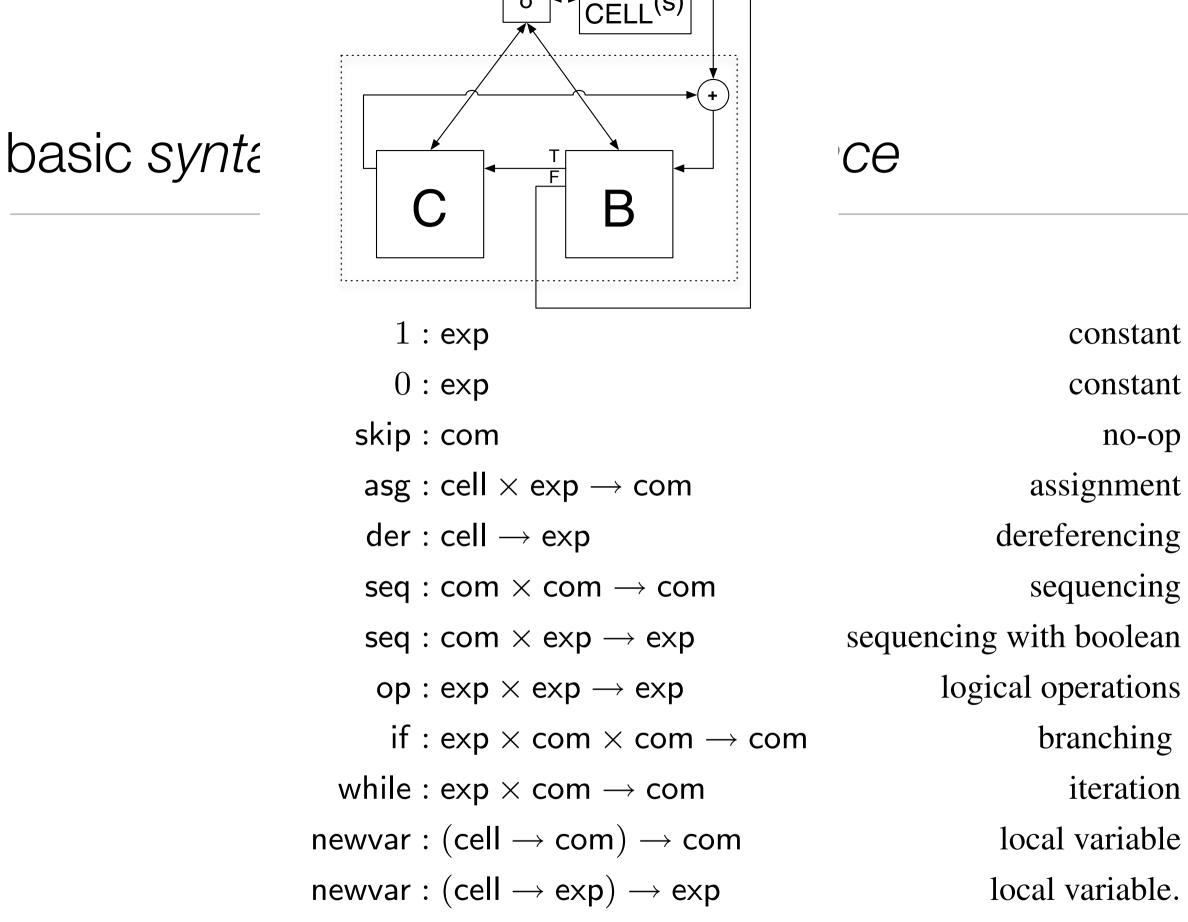
mination

iction

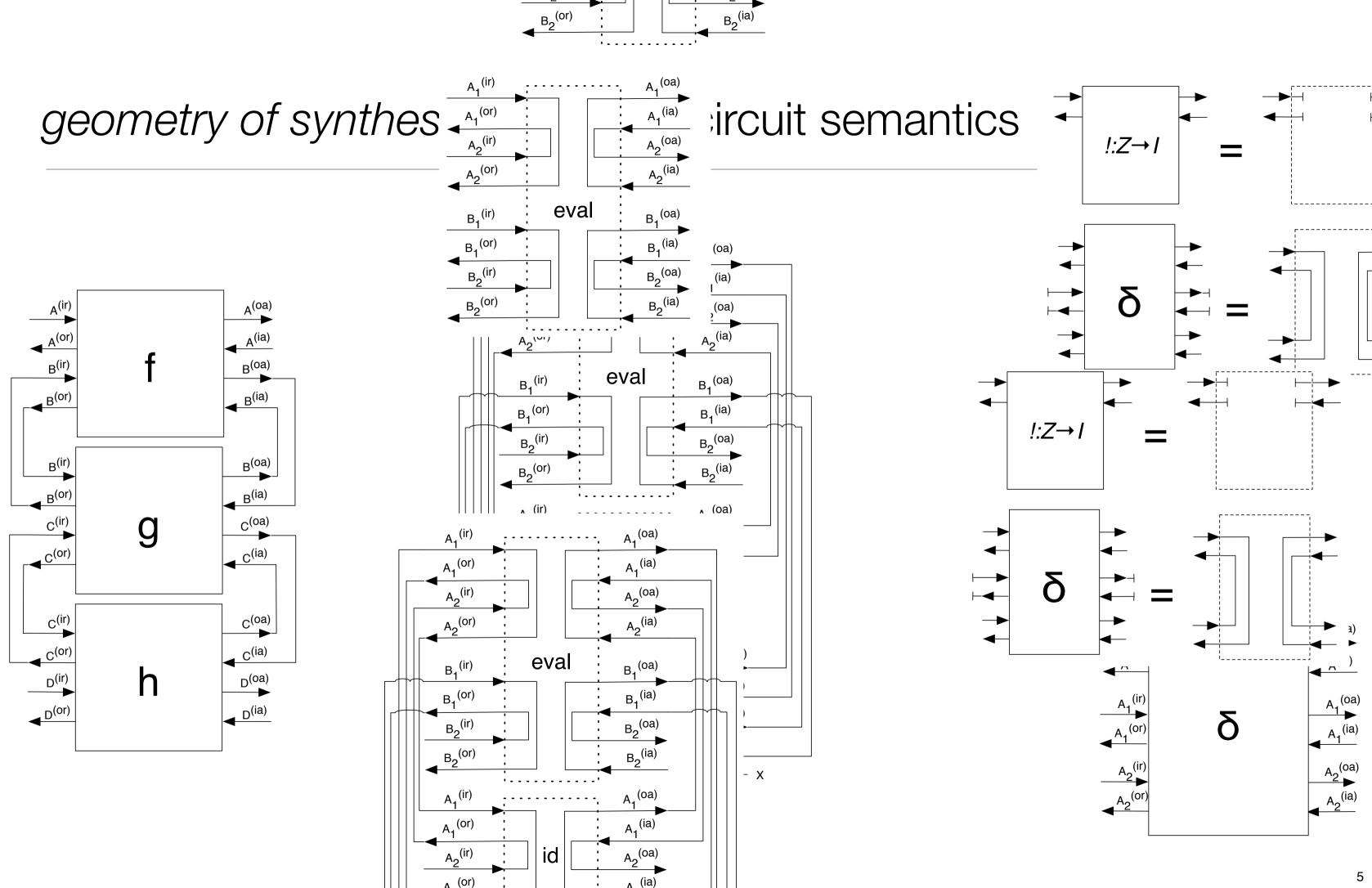
basic syntactic control of interference

cons	$1: \exp$
cons	0: exp
n	skip : com
assignr	asg:cell imesexp ocom
dereferen	$der:cell\toexp$
sequen	$seq:com\timescom\tocom$
sequencing with boo	$seq:com\timesexp\toexp$
logical operat	op:exp imesexp oexp
branch	$if:exp\timescom\timescom\tocom$
itera	while : exp \times com \rightarrow com
local vari	$newvar:(cell\tocom)\tocom$
local varia	$newvar:(cell\toexp)\toexp$

- stant
- stant
- io-op
- ment
- ncing
- ncing
- olean
- tions
- hing
- ation
- iable
- able.



par : com \rightarrow com \rightarrow com.





closed monoida July , man July produce

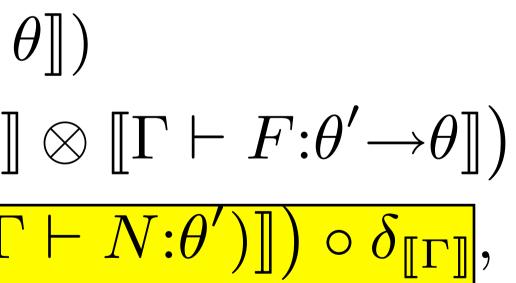
$$\begin{bmatrix} x: \theta \vdash x: \theta \end{bmatrix} = id_{\llbracket\theta} \end{bmatrix}$$
$$\begin{bmatrix} \Gamma, x: \theta' \vdash M: \theta \end{bmatrix} = \llbracket \Gamma \vdash M: \theta \rrbracket \circ \pi_1$$
$$\begin{bmatrix} \Gamma \vdash \lambda x.M: \theta' \to \theta \end{bmatrix} = \Lambda(\llbracket \Gamma, x: \theta' \vdash M: \theta)$$
$$\equiv [\Gamma, \Delta \vdash FM: \theta] = \text{eval} \circ (\llbracket \Delta \vdash M: \theta']$$
$$\equiv [\Gamma \vdash \langle M, N \rangle: \theta \times \theta'] = (\llbracket \Gamma \vdash M: \theta] \otimes \llbracket \rho(\Gamma)$$

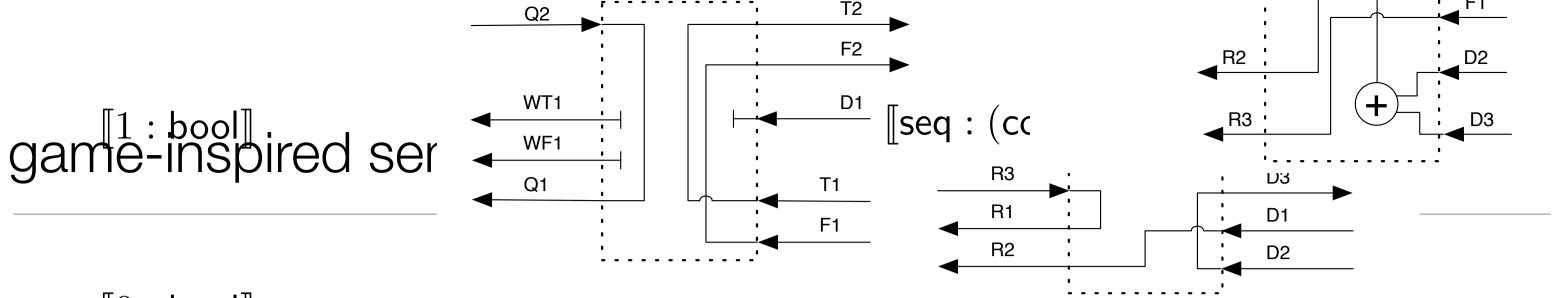
θ]) $]\!] \otimes [\![\Gamma \vdash F : \theta' \rightarrow \theta]\!])$ $\Gamma \vdash N: \theta')]\!]) \circ \delta_{\llbracket \Gamma \rrbracket},$



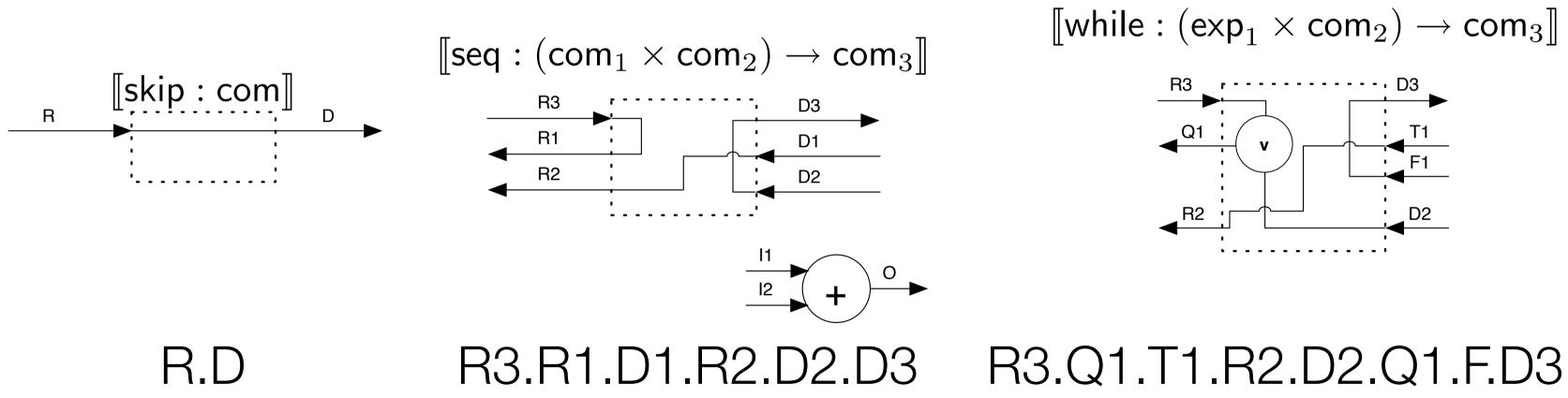
closed monoida July , man July produce

$$\begin{bmatrix} x:\theta \vdash x:\theta \end{bmatrix} = id_{\llbracket\theta} \end{bmatrix}$$
$$\begin{bmatrix} \Gamma, x:\theta' \vdash M:\theta \end{bmatrix} = \llbracket \Gamma \vdash M:\theta \rrbracket \circ \pi_1$$
$$\begin{bmatrix} \Gamma \vdash \lambda x.M:\theta' \to \theta \end{bmatrix} = \Lambda(\llbracket \Gamma, x:\theta' \vdash M: \\ \llbracket \Gamma, \Delta \vdash FM:\theta \rrbracket = \text{eval} \circ \left(\llbracket \Delta \vdash M:\theta' \end{bmatrix}$$
$$\begin{bmatrix} \Gamma \vdash \langle M, N \rangle : \theta \times \theta' \rrbracket = \left(\llbracket \Gamma \vdash M:\theta \rrbracket \otimes \llbracket \rho(\Gamma \restriction M) \right)$$

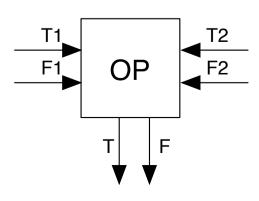




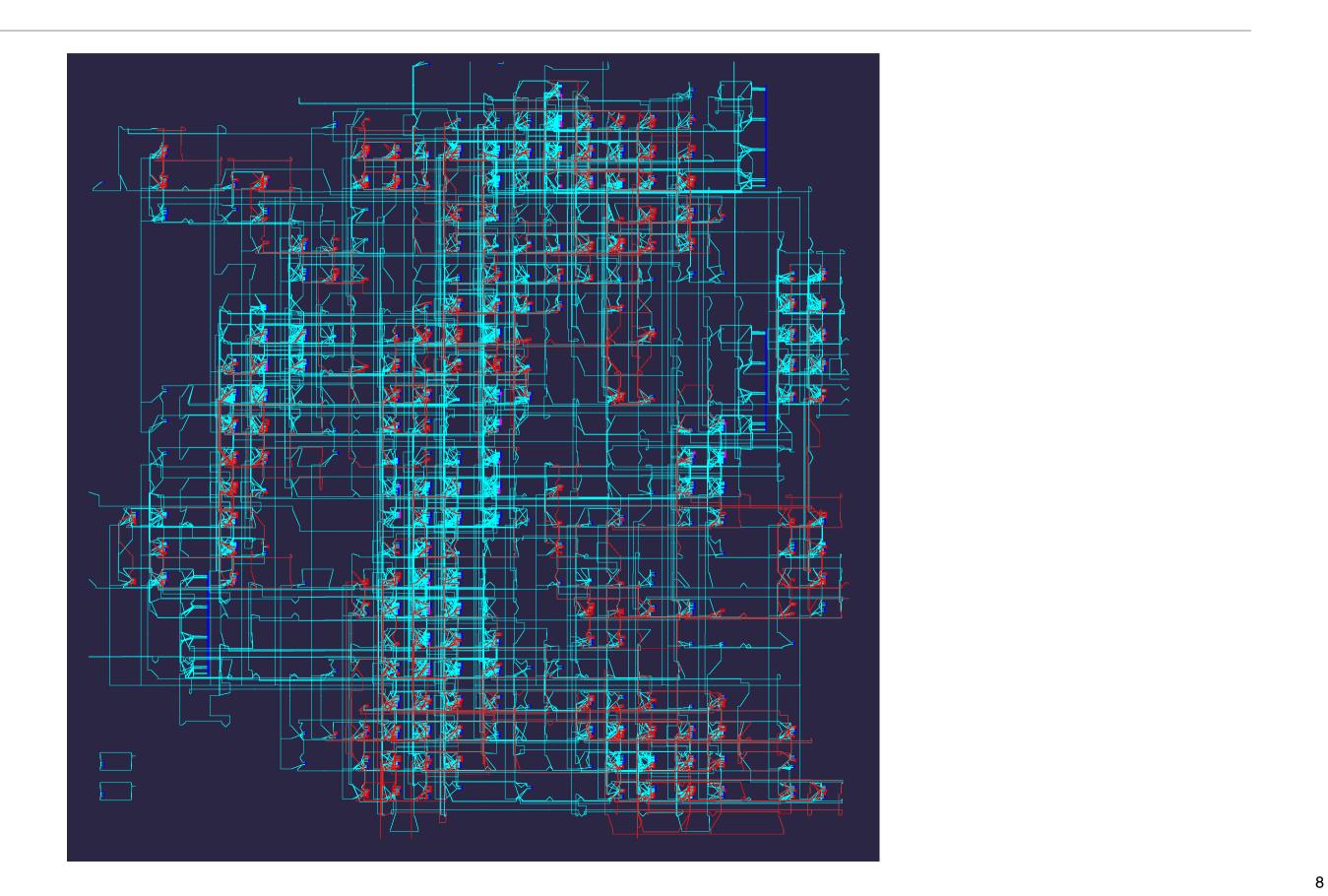






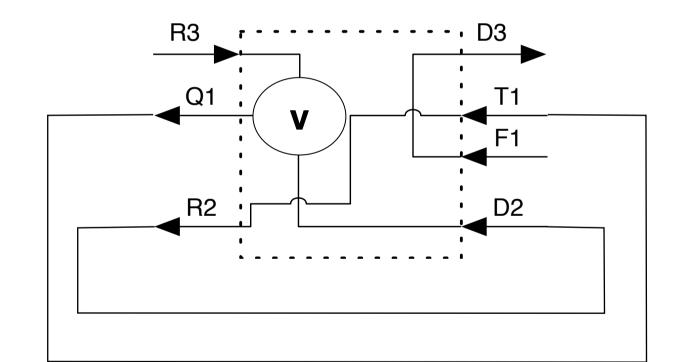


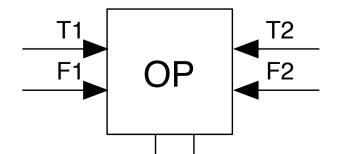
it works



$\llbracket While \stackrel{!}{:} (exp_1^{+} \stackrel{rue}{\times} com_2^{\circ}) \stackrel{skip}{\to} com_3 \rrbracket$



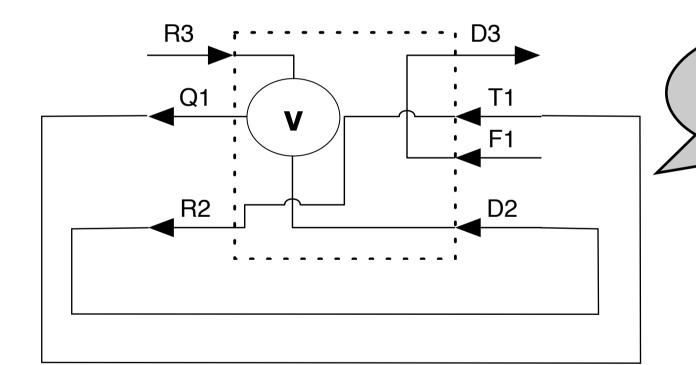


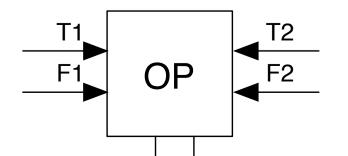


9

$\llbracket While \stackrel{!}{:} (exp_1^{r} \stackrel{vecodo}{\times} com_2^{o}) \stackrel{skip}{\rightarrow} com_3 \rrbracket$



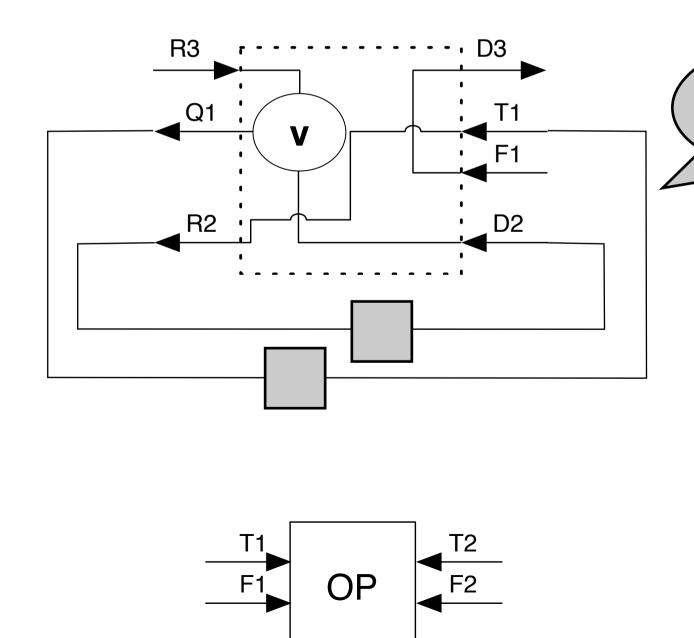


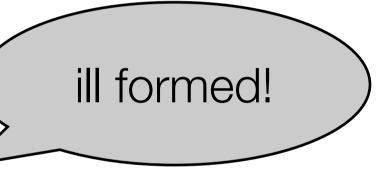




$\llbracket While \stackrel{!}{:} (exp_1^{r} \stackrel{vecodo}{\times} com_2^{o}) \stackrel{skip}{\rightarrow} com_3 \rrbracket$

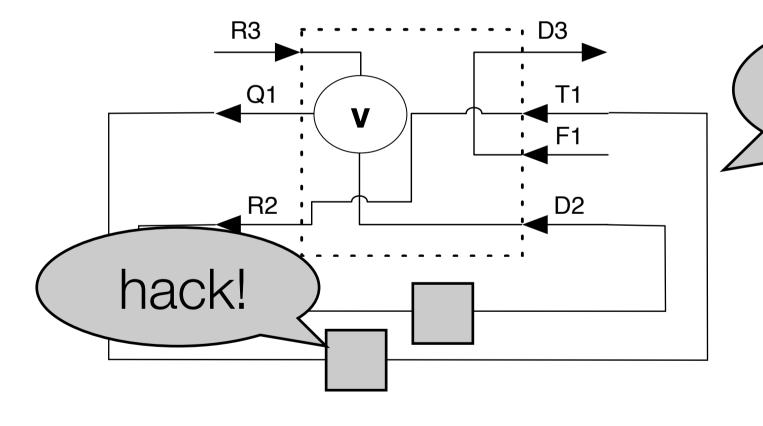


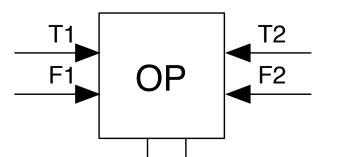




$\llbracket While \stackrel{!}{:} (exp_1^{+} \stackrel{rue}{\times} com_2^{\circ}) \stackrel{skip}{\to} com_3 \rrbracket$

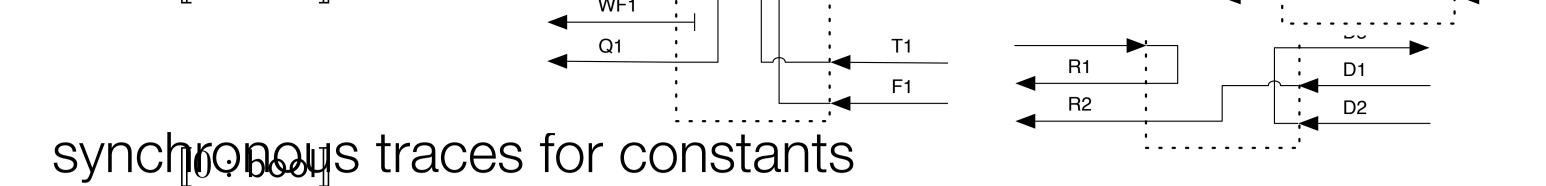


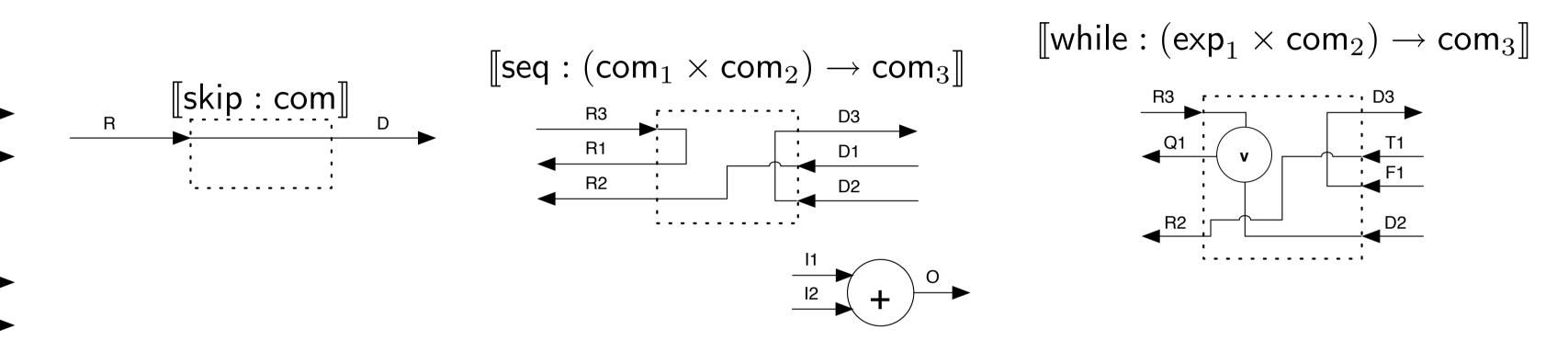


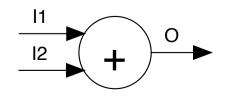


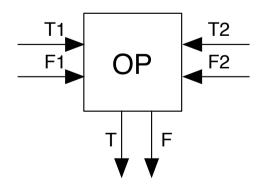


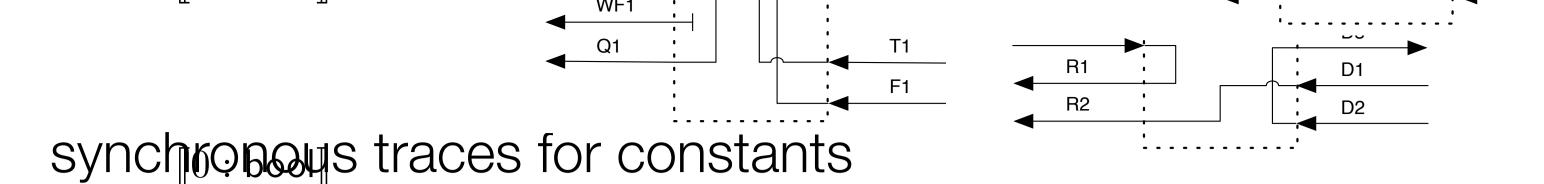
digital (clocked) hardware is synchronous game-semantic models are asynchronous

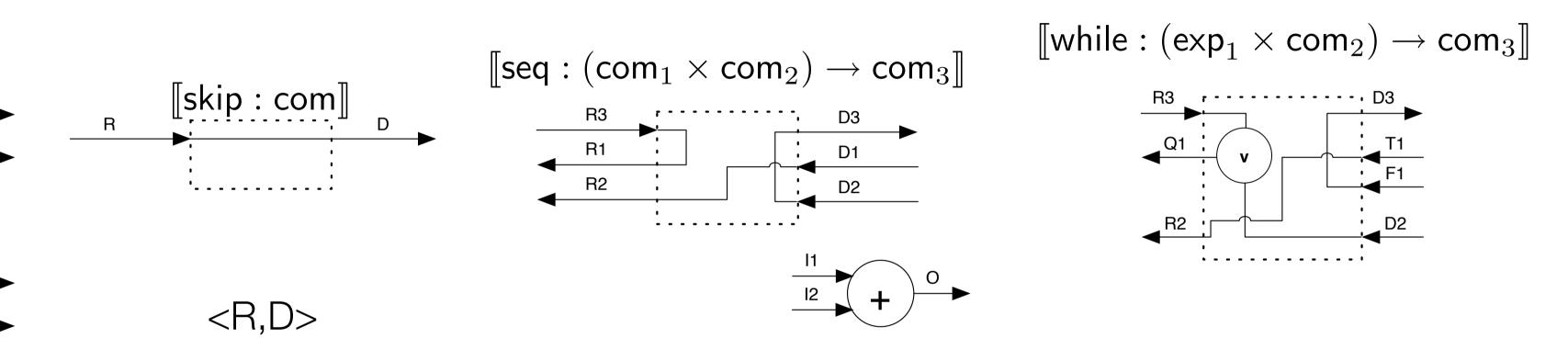


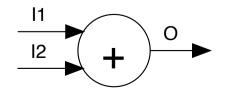


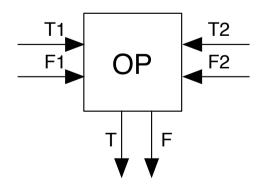


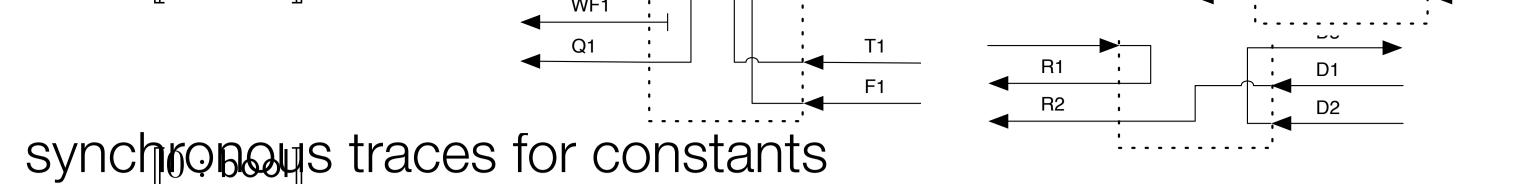


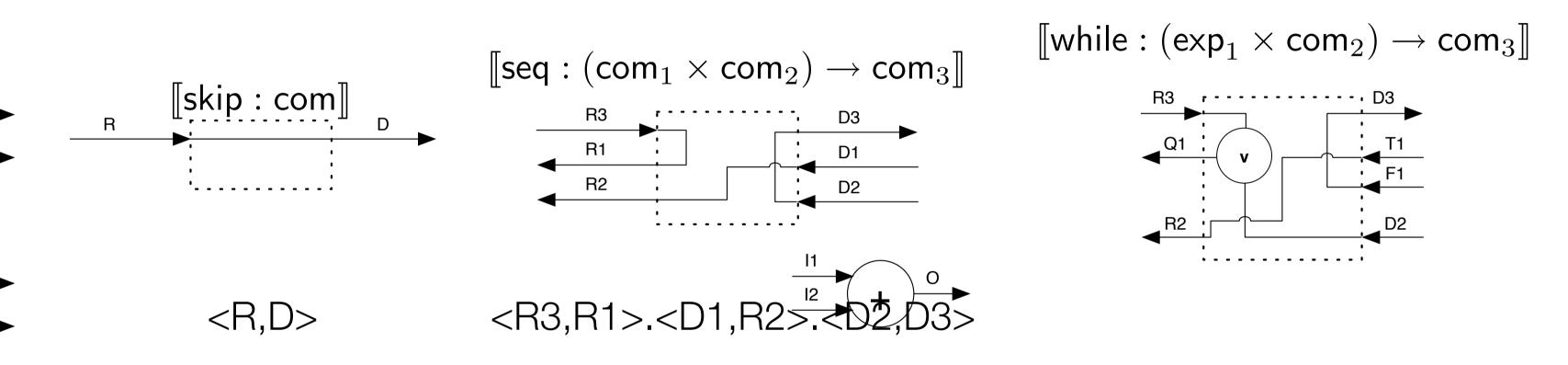


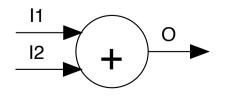


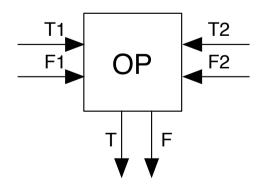


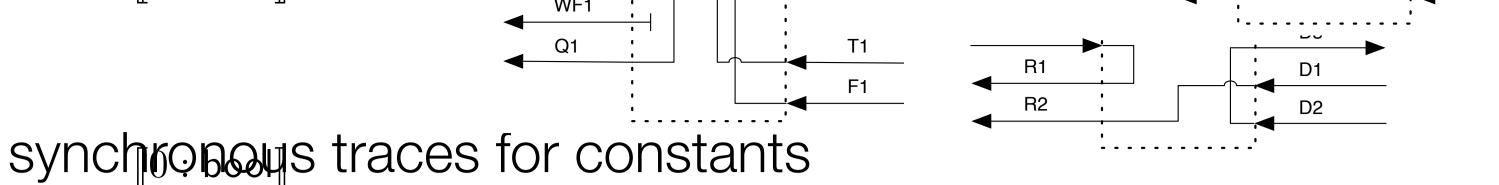


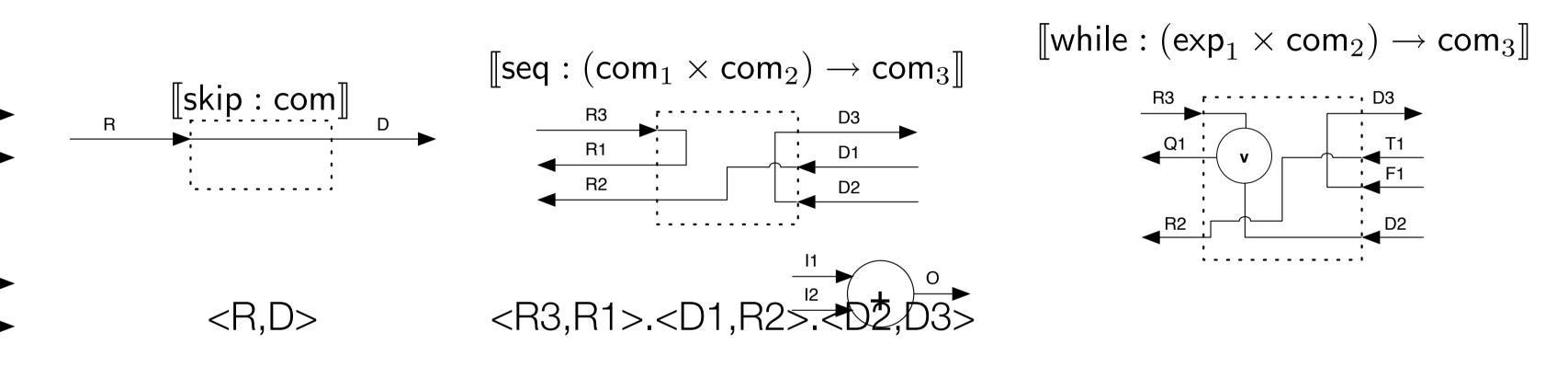


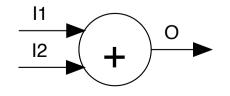


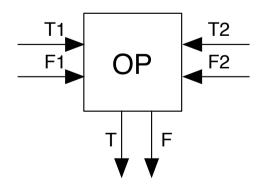


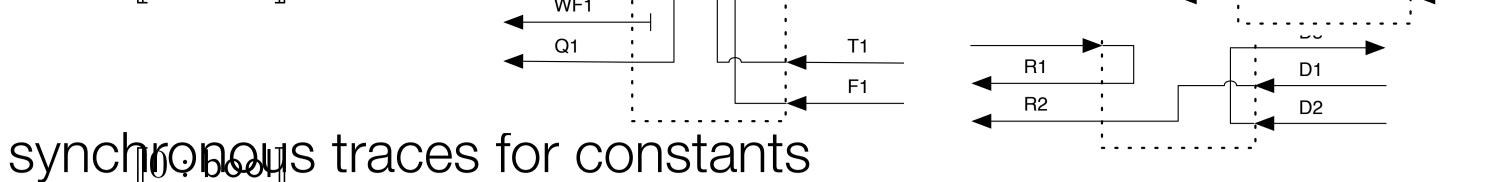


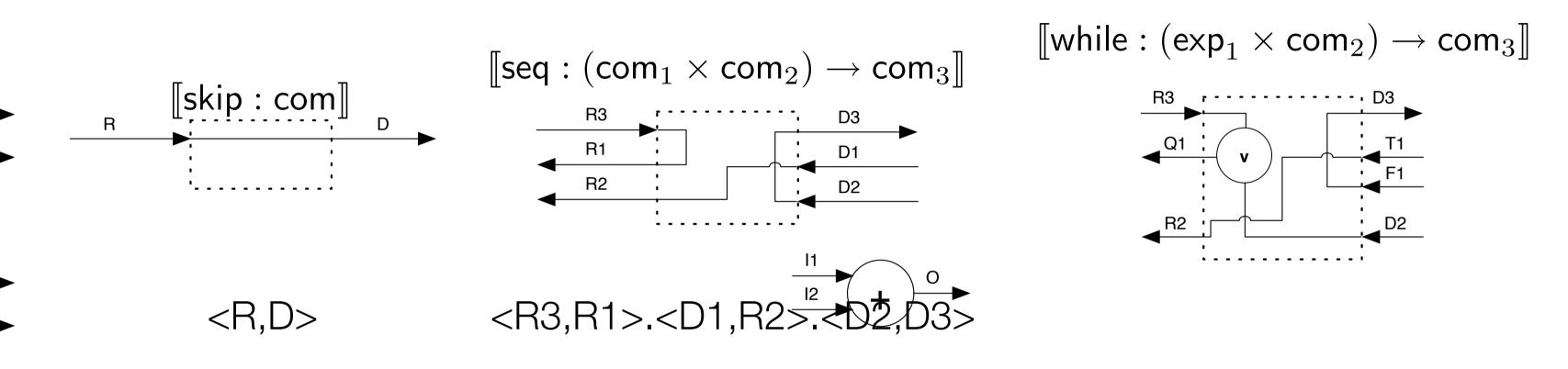




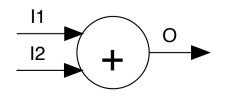


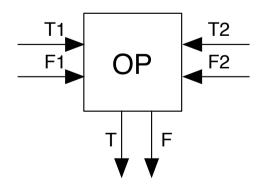


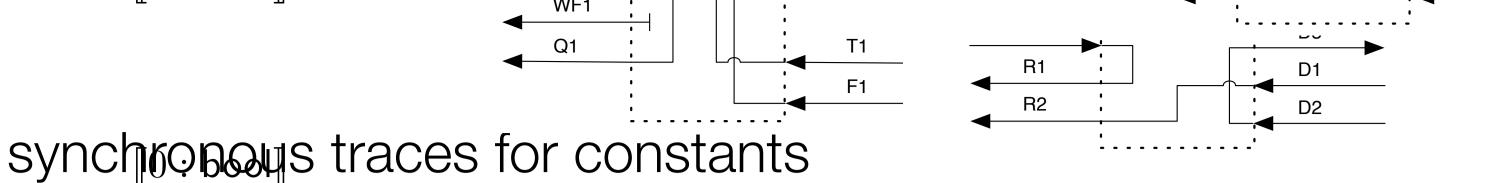


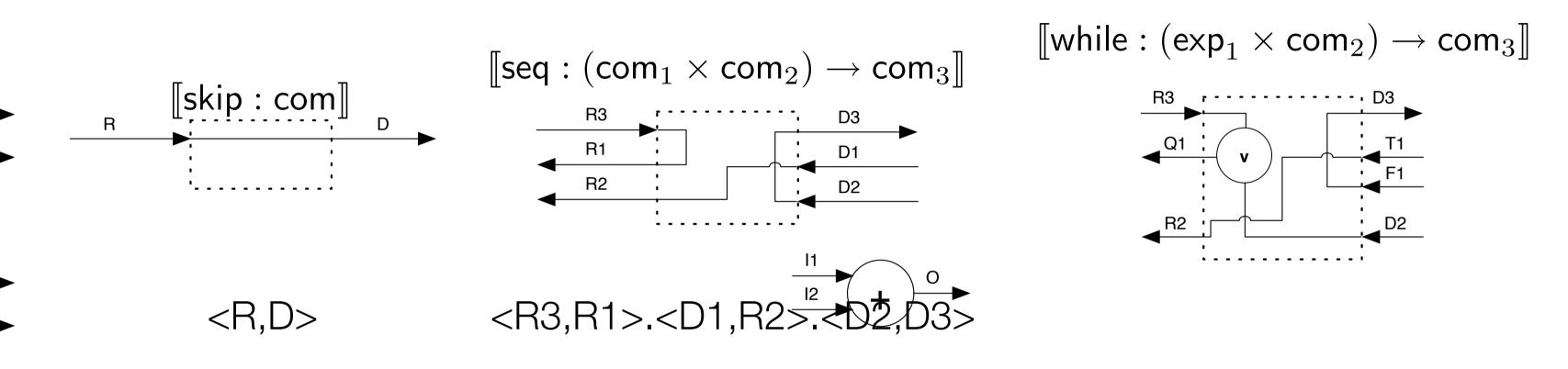


<R3,R1>.<D1,R2,D2,D3>



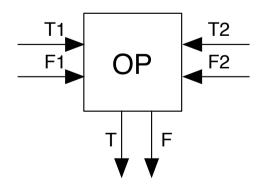


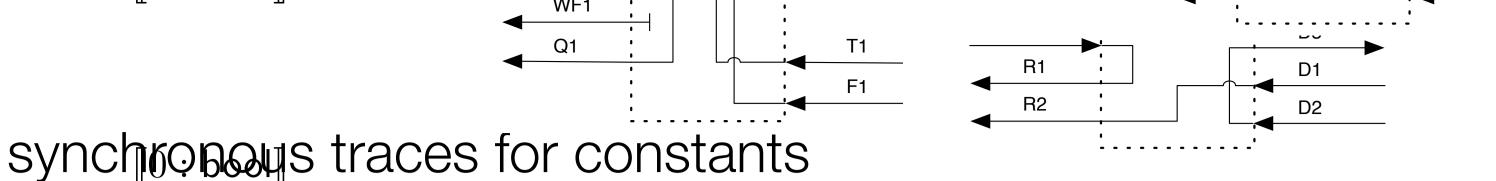


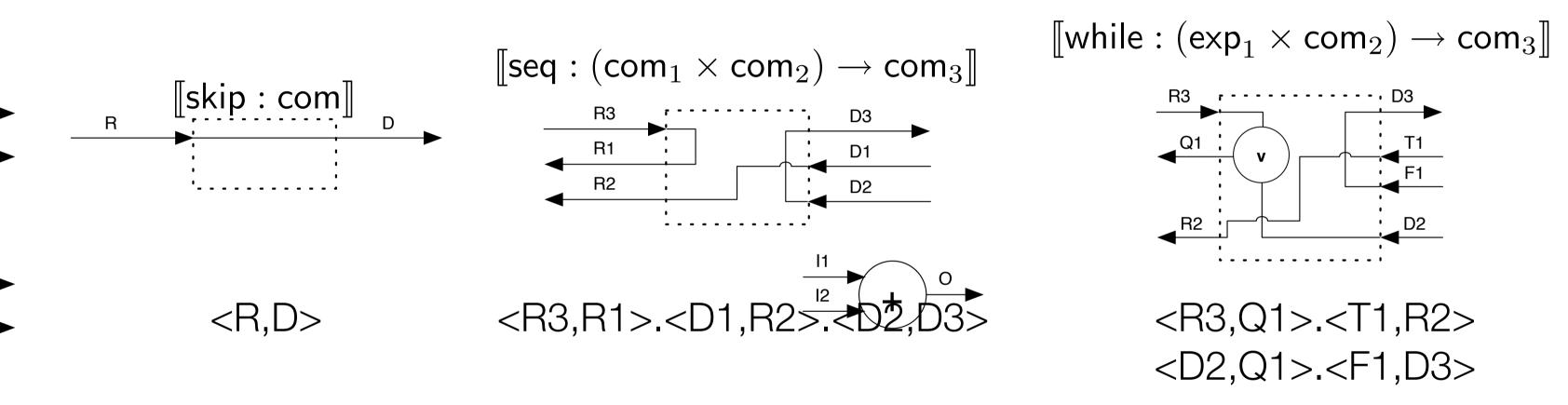


<R3,R1>.<D1,R2,D2,D3>



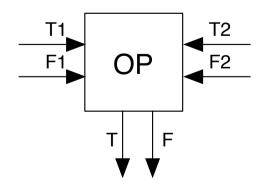


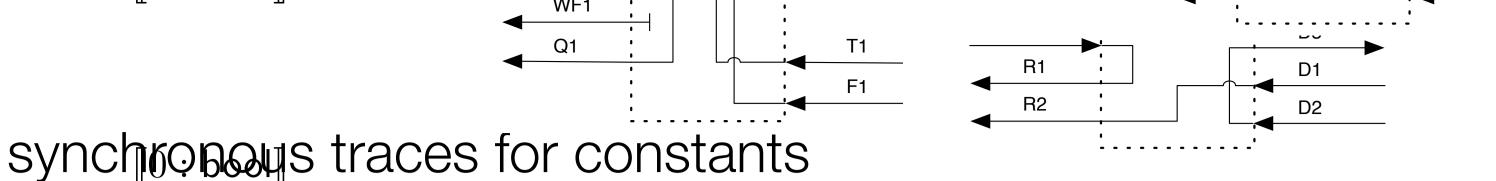


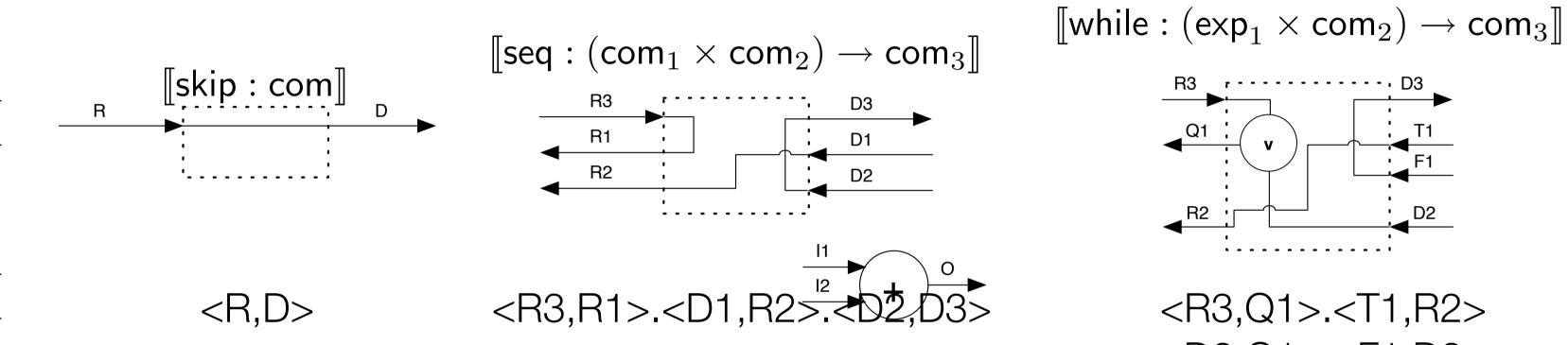


<R3,R1>.<D1,R2,D2,D3>





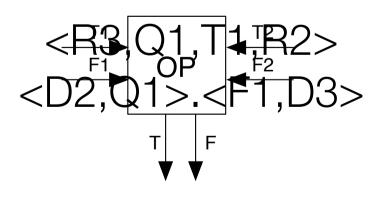


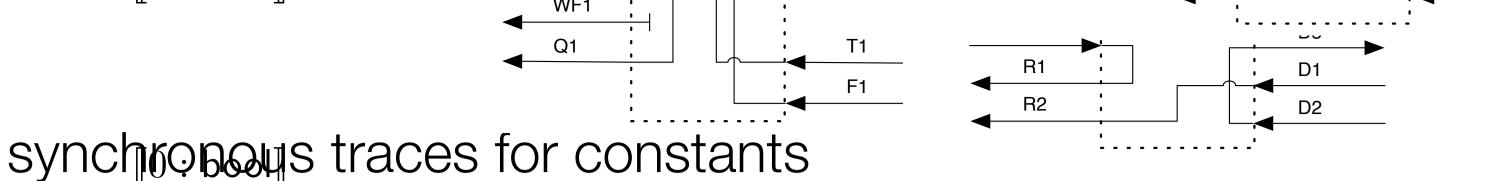


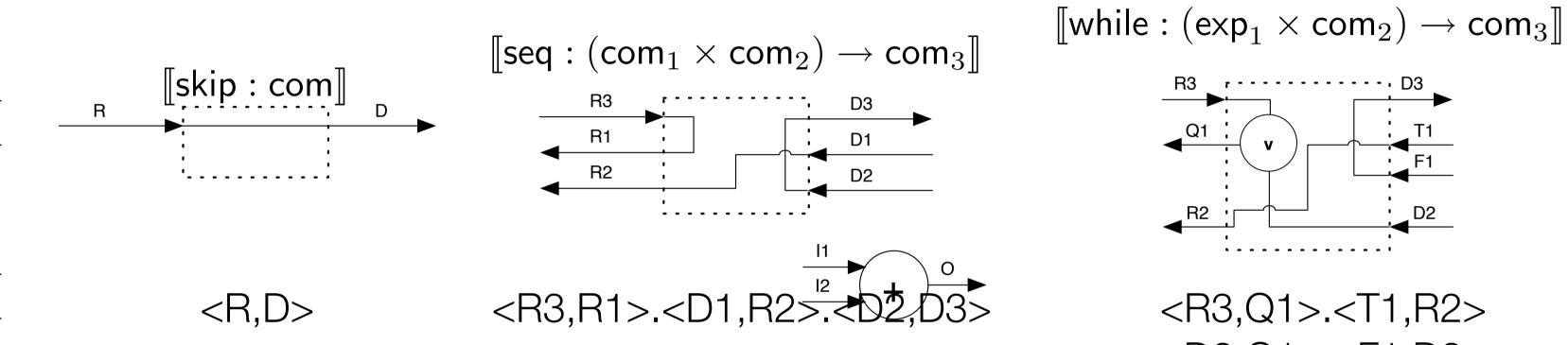
<R3,R1>.<D1,R2,D2,D3>



<D2,Q1>.<F1,D3>



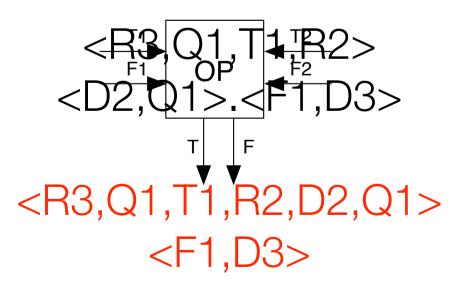




<R3,R1>.<D1,R2,D2,D3>



<D2,Q1>.<F1,D3>



more complex languages: scc

$$\frac{\Gamma, x : \theta^m, y : \theta^n \vdash_r M}{\Gamma, x : \theta^m + n \vdash_r M[x/y]}$$

$\frac{\boldsymbol{\Lambda}:\boldsymbol{\theta}'}{\boldsymbol{y}]:\boldsymbol{\theta}'}$

more complex languages: scc

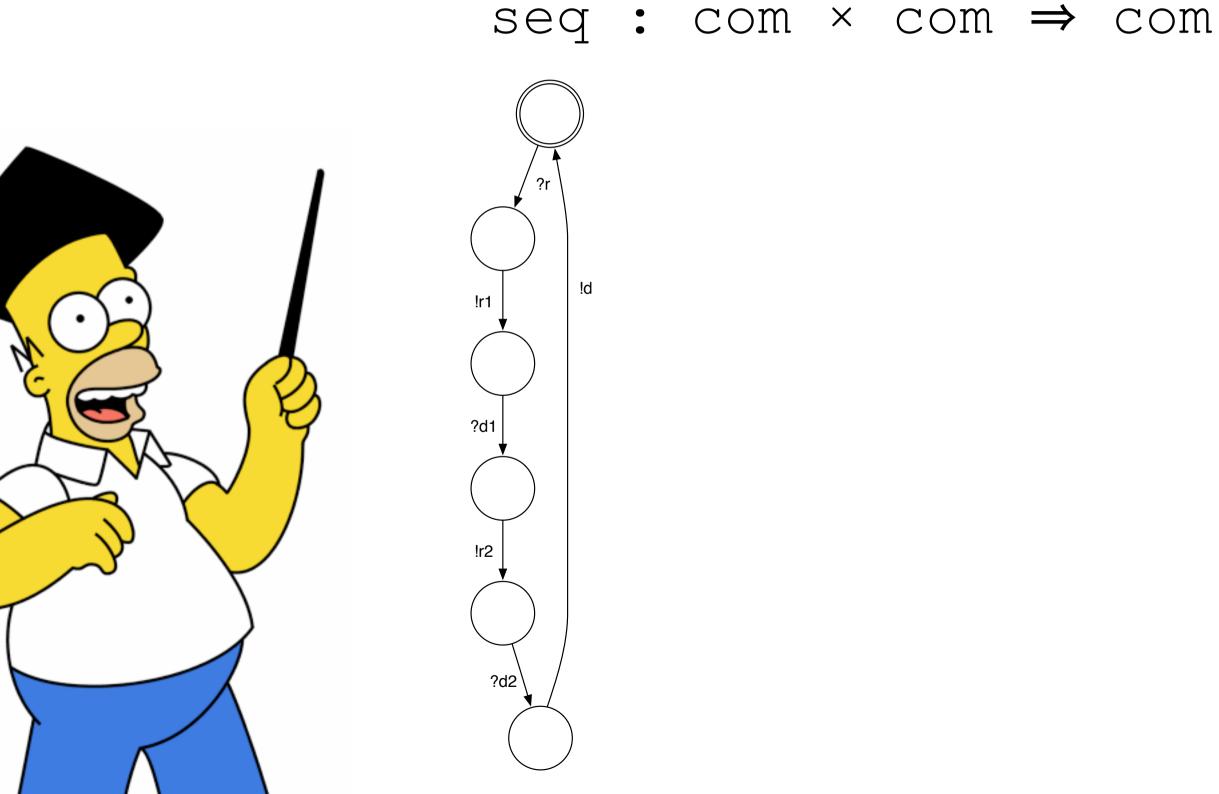
$$\begin{array}{c} \mathsf{\Gamma}, x : \theta^m, y : \theta^n \vdash_r M \\ \mathsf{\Gamma}, x : \theta^{m+n} \vdash_r M[x/y] \end{array} \end{array}$$

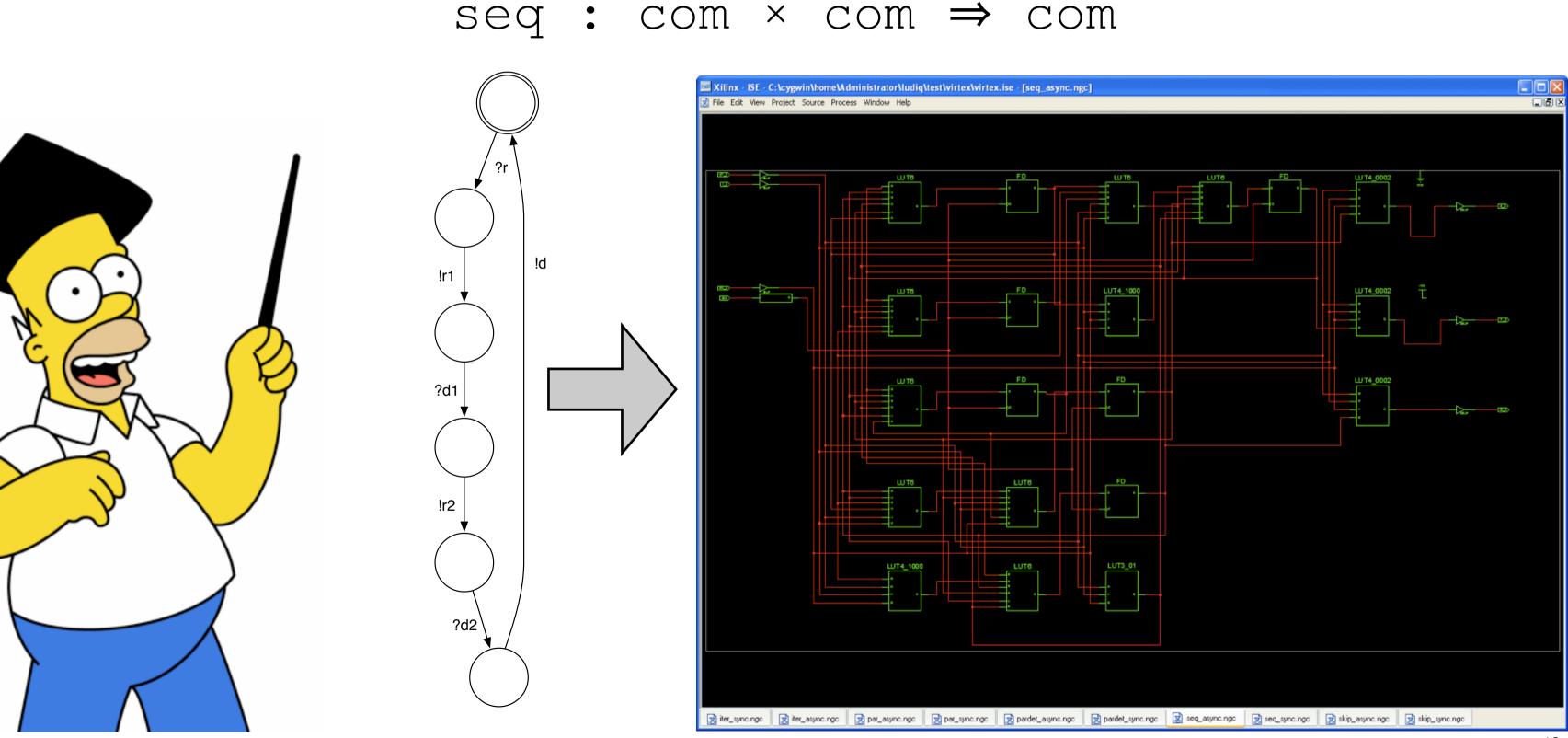
$$A^{n} \to B \equiv \underbrace{ A \otimes \cdots \otimes A}_{n} \to B$$
$$\begin{bmatrix} A \odot B \end{bmatrix} = \begin{bmatrix} A \end{bmatrix} \begin{bmatrix} B \end{bmatrix} \cup \begin{bmatrix} B \end{bmatrix} \begin{bmatrix} A \end{bmatrix}$$

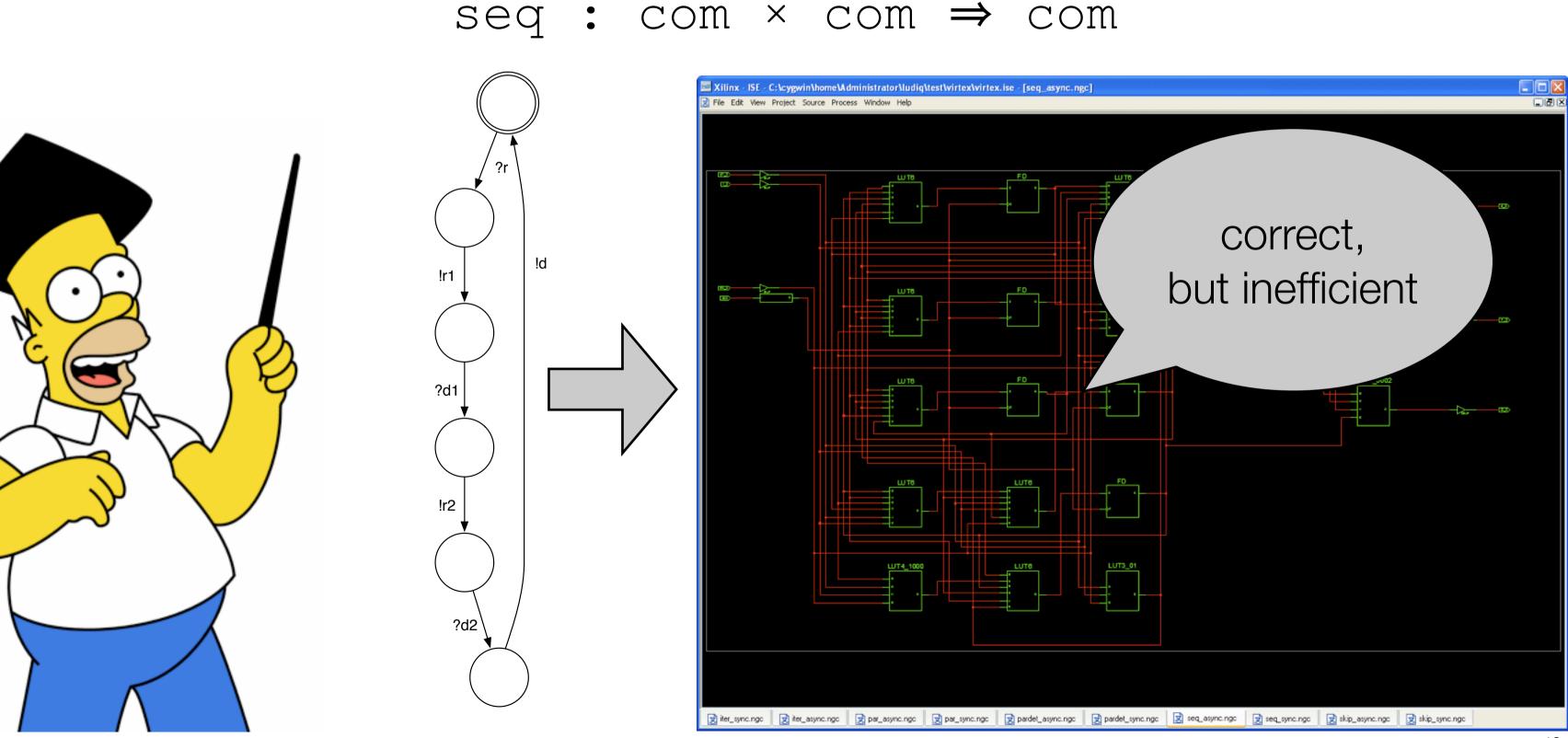
$\frac{A: heta'}{y]: heta'}$

$\llbracket A \rrbracket = \llbracket A \rrbracket^*$

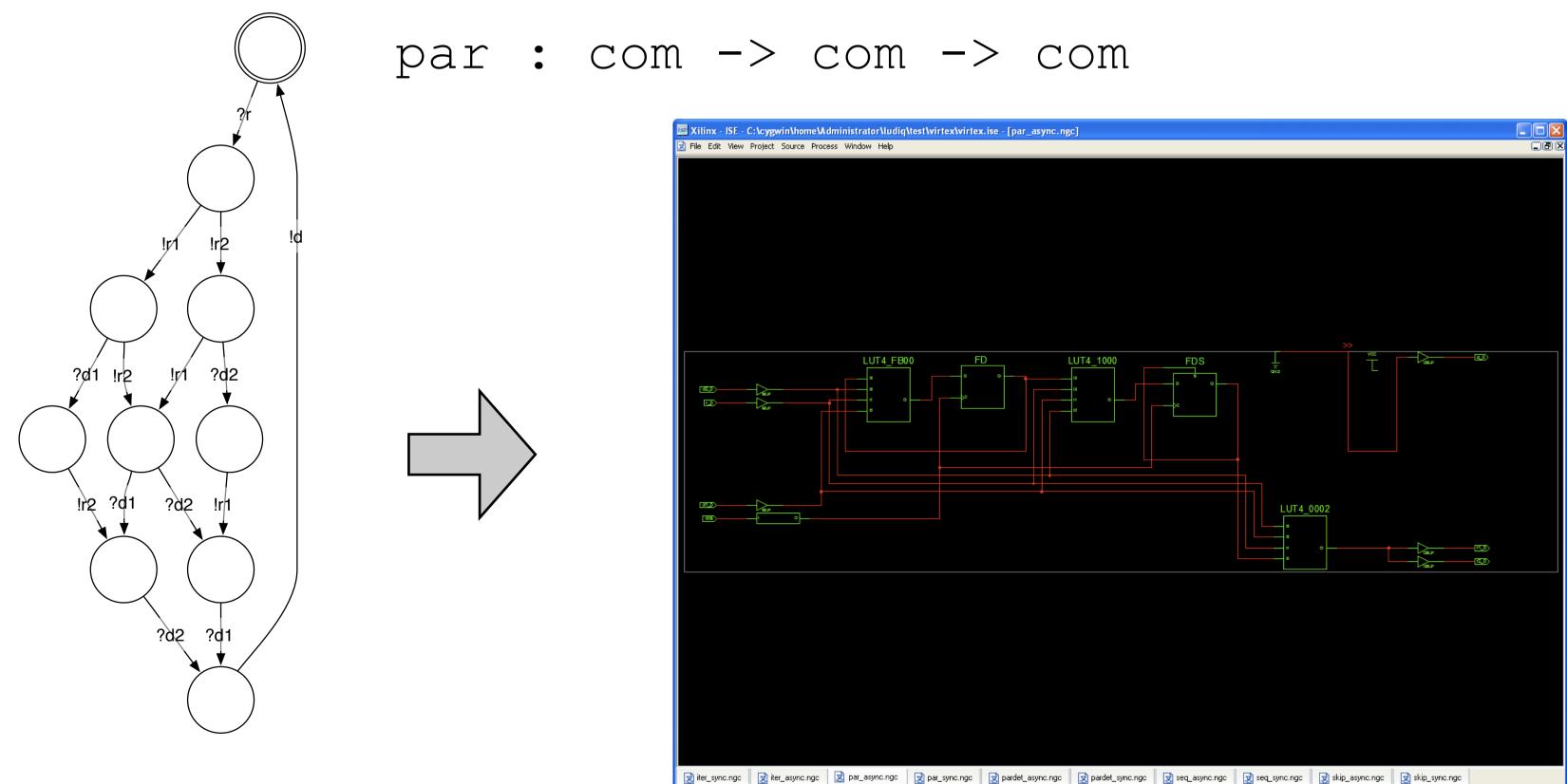




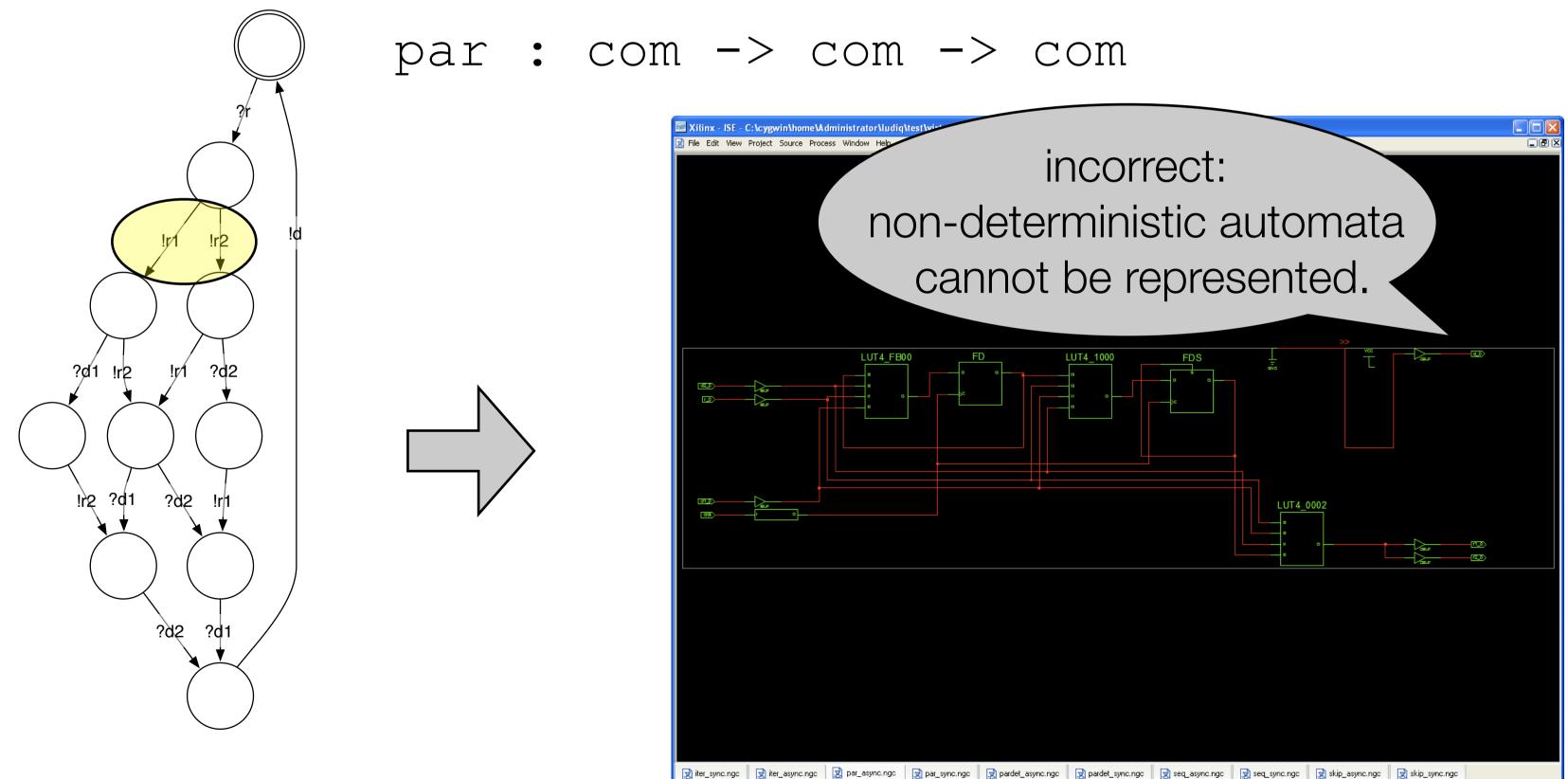




more on representing game models in hardware



more on representing game models in hardware



a solution: round abstraction

15

a solution: round abstraction

• "Reactive Modules", Alur & Henzinger. LICS 1996 / FMSD 1999.

a solution: round abstraction

- "Reactive Modules", Alur & Henzinger. LICS 1996 / FMSD 1999.
- create synchronous "rounds" of signals controlled by specific signals used as "clocks"

FMSD 1999. ov specific signals

a solution: round abstraction

- "Reactive Modules", Alur & Henzinger. LICS 1996 / FMSD 1999.
- create synchronous "rounds" of signals controlled by specific signals used as "clocks"
- we use a "maximal" form of round abstraction

FMSD 1999. ov specific signals

a solution: round abstraction

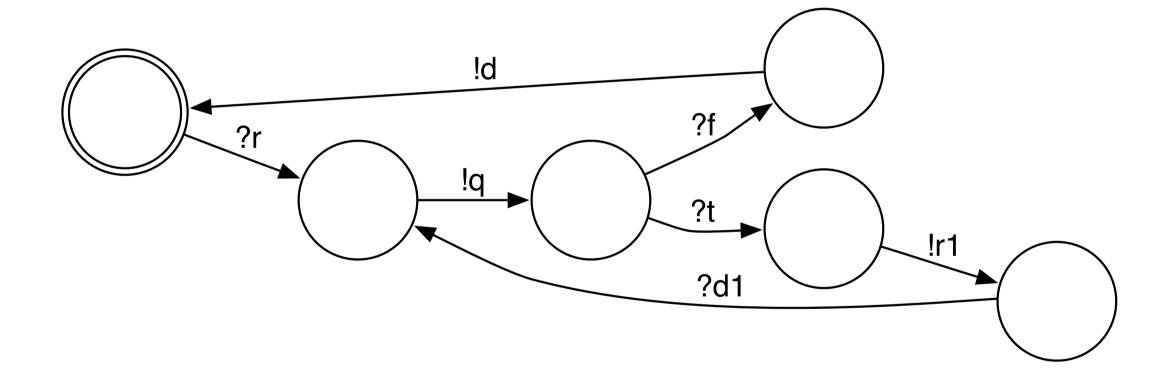
- "Reactive Modules", Alur & Henzinger. LICS 1996 / FMSD 1999.
- create synchronous "rounds" of signals controlled by specific signals used as "clocks"
- we use a "maximal" form of round abstraction
 - make the rounds as long as possible

FMSD 1999. ov specific signals

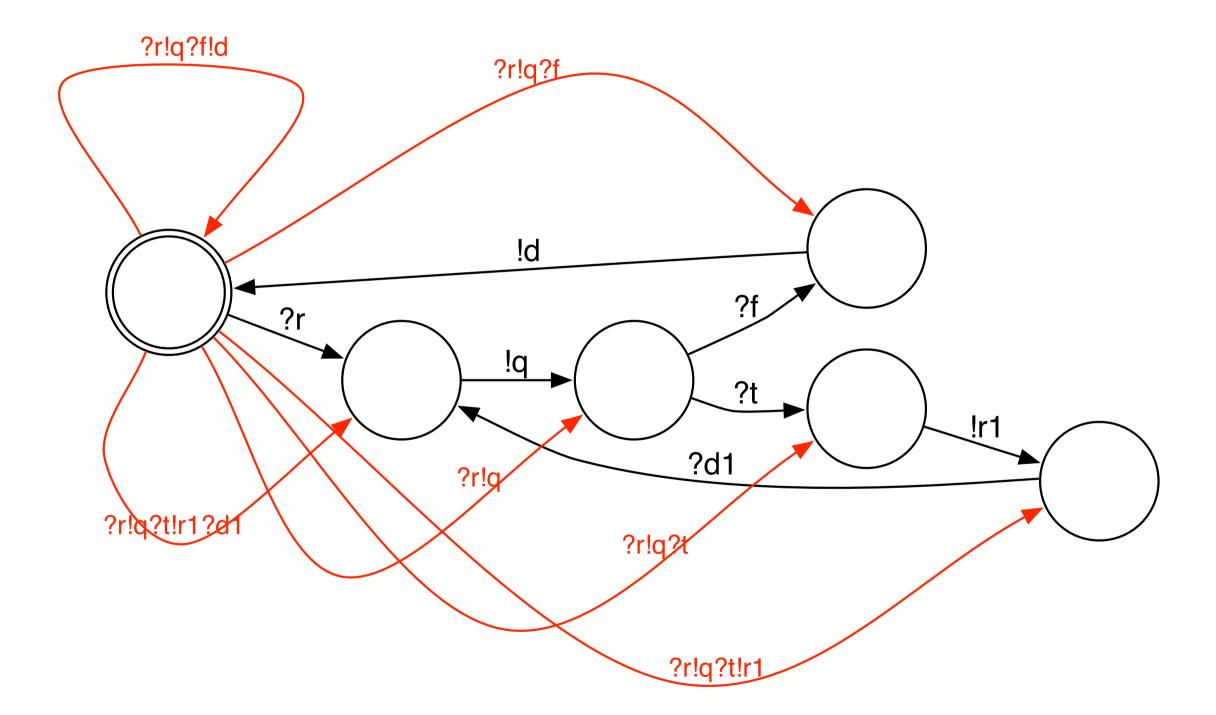
a solution: round abstraction

- "Reactive Modules", Alur & Henzinger. LICS 1996 / FMSD 1999.
- create synchronous "rounds" of signals controlled by specific signals used as "clocks"
- we use a "maximal" form of round abstraction
 - make the rounds as long as possible
 - ... but avoid using the same signal twice in one round (cf "schizophrenia" in Esterel)

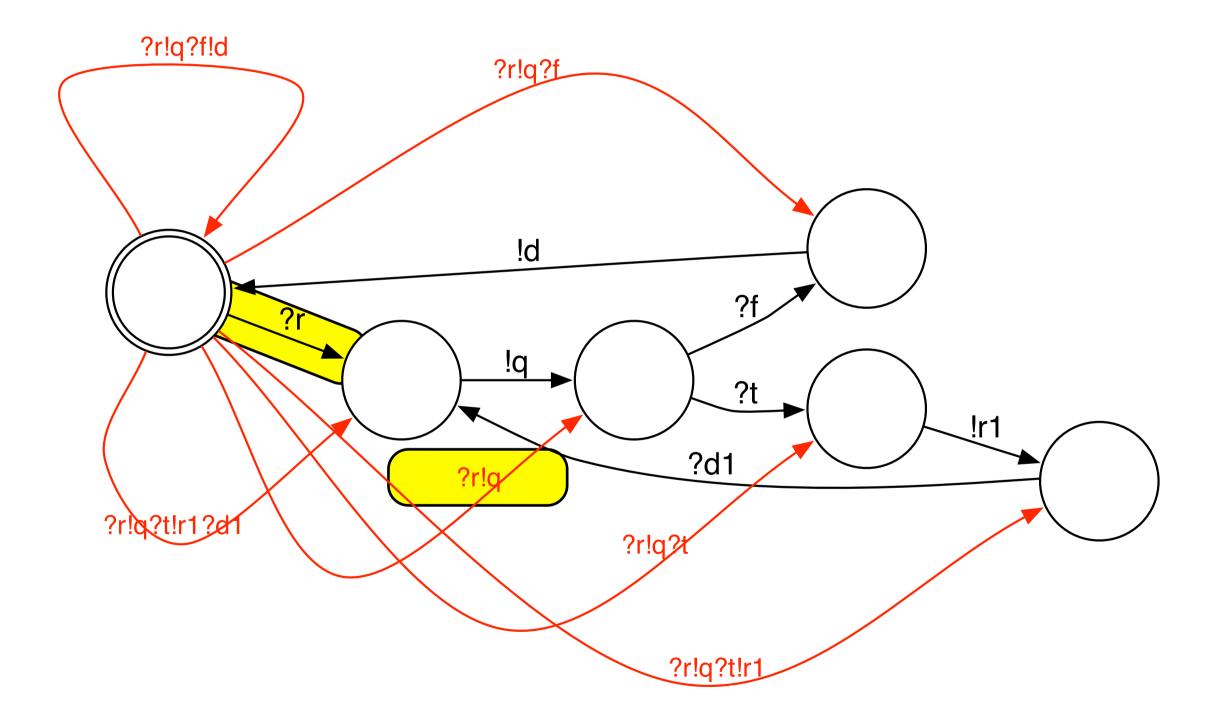
asynchronous automaton: while



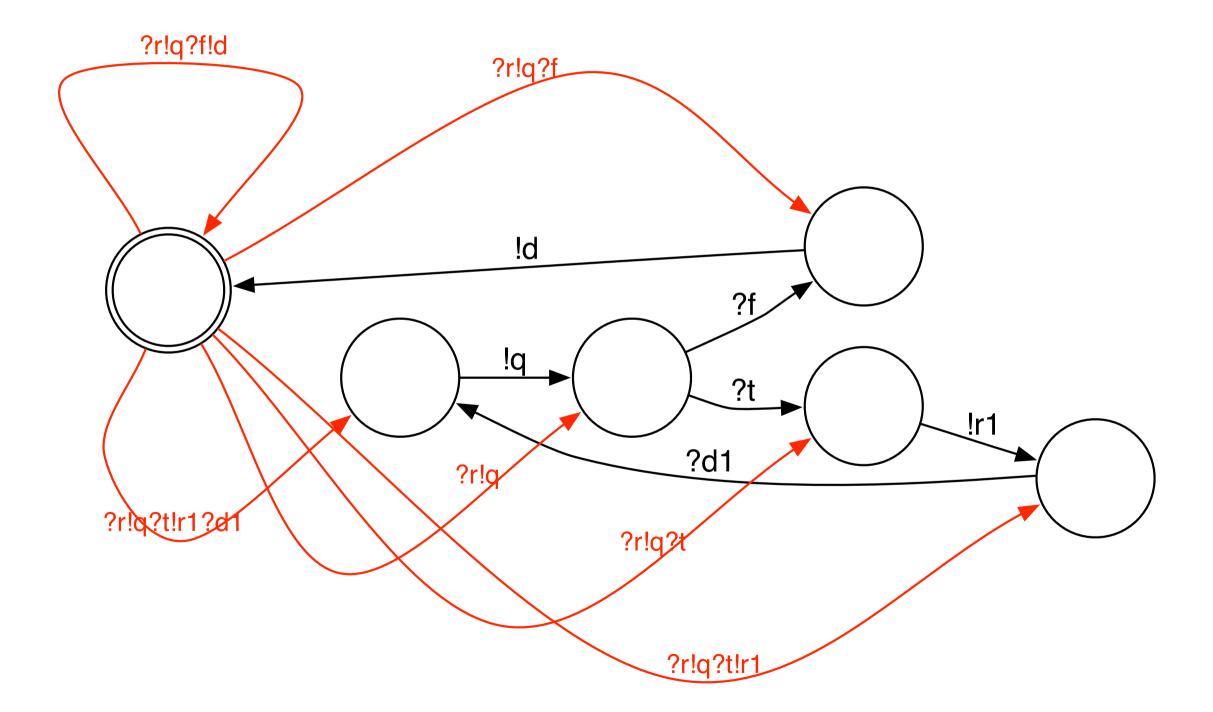
step 1: round generation



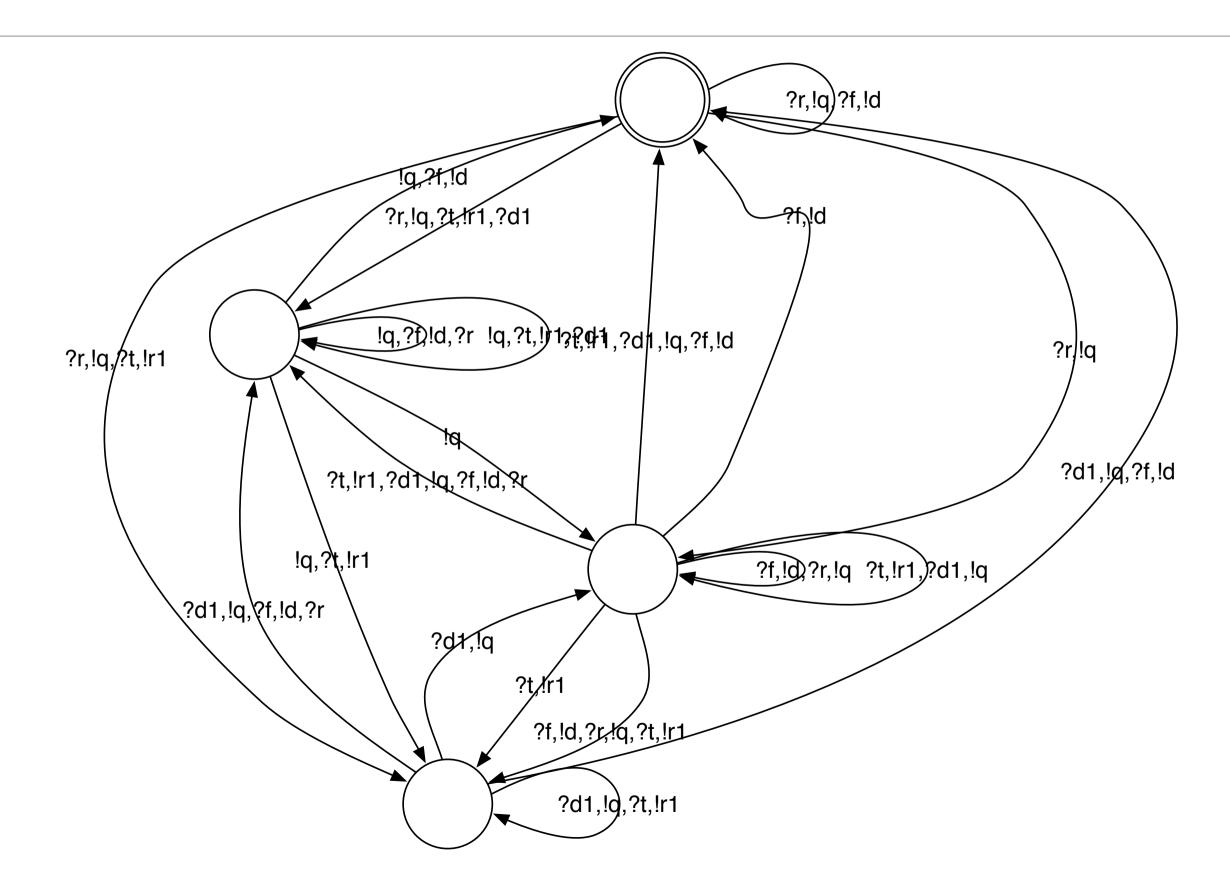
step 1: round generation

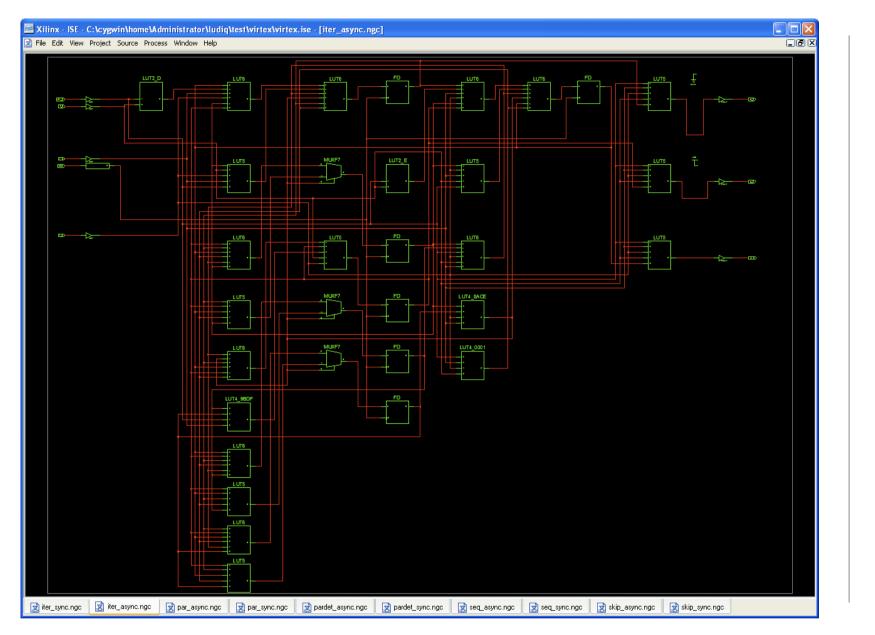


Step 2: reduction

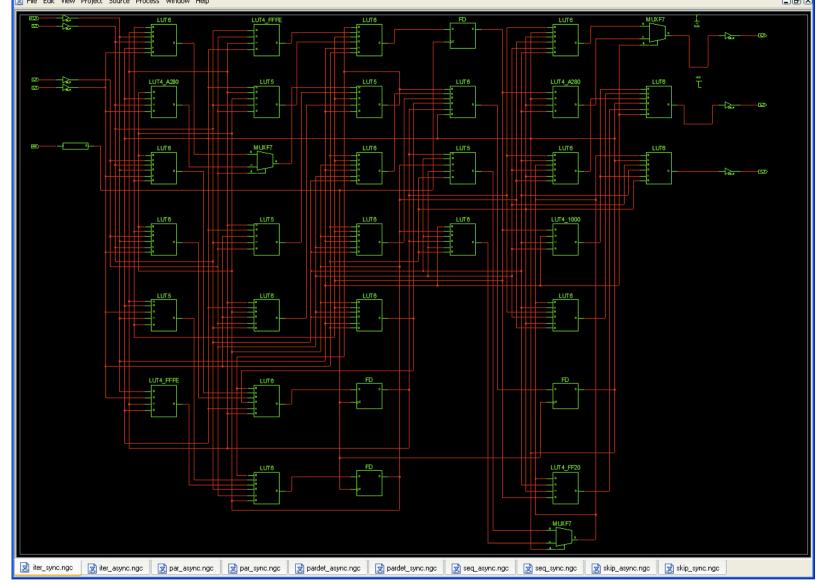


synchronous automaton for while









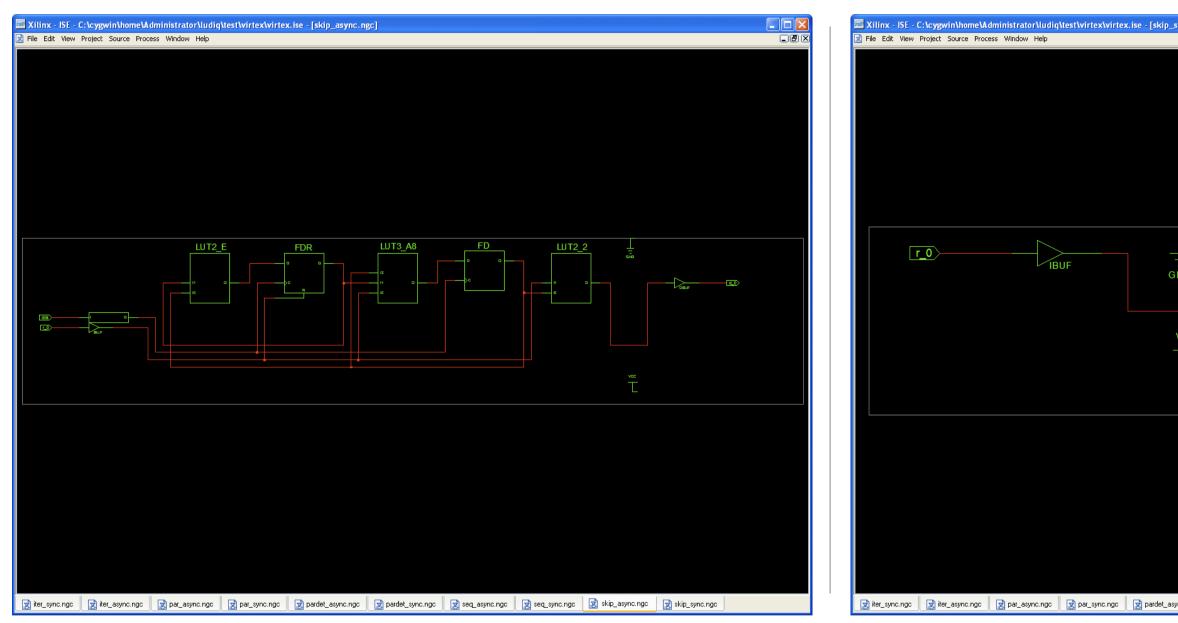
6 states, 23 LUTs



asynchronous versus synchronous representations for iteration

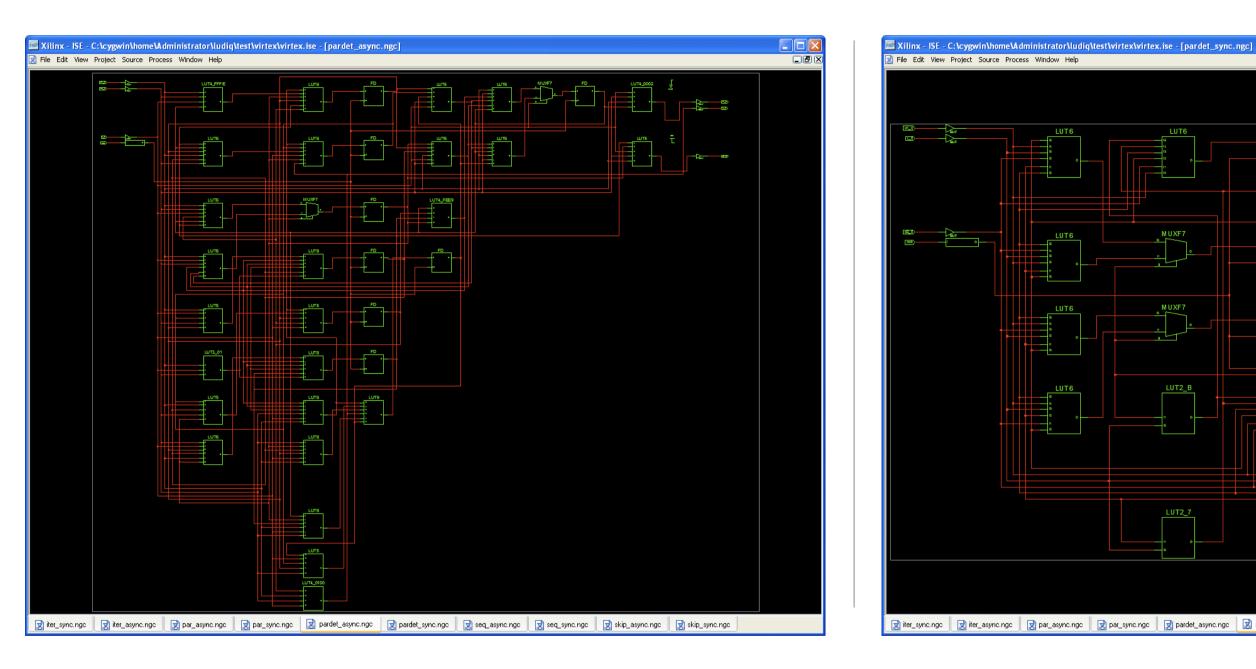


4 states, 28 LUTs



asynchronous versus synchronous representations for skip

_sync.ngc]						
						. B ×
		\sim				
GND		OBUF		O		
GND						
VCC						
¬					🕞 aluin, auna 11-11	
sync.ngc 🛃	pardet_sync.ngc	👿 seq_async.ngc	🛃 seq_sync.ngc	🛃 skip_async.ngc	📓 skip_sync.ngc	



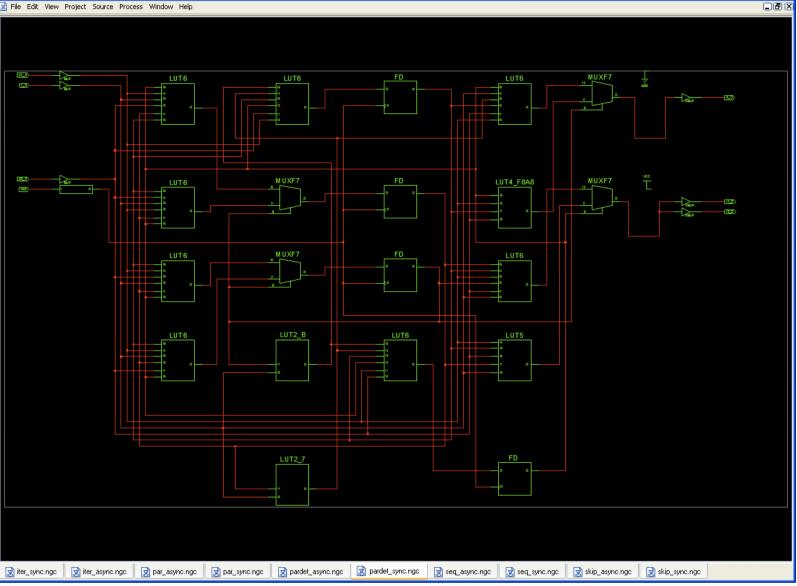
8 states, 26 LUTs

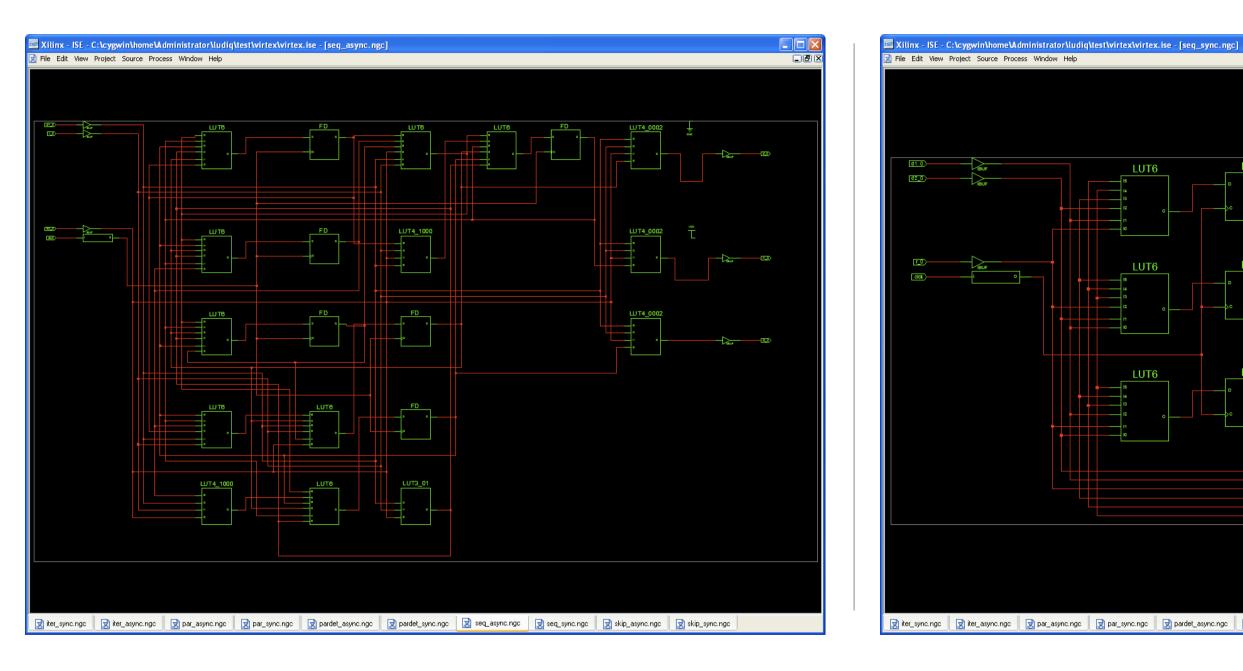
4 states, 12 LUTs

async vs sync representations for (deterministic) parallel composition







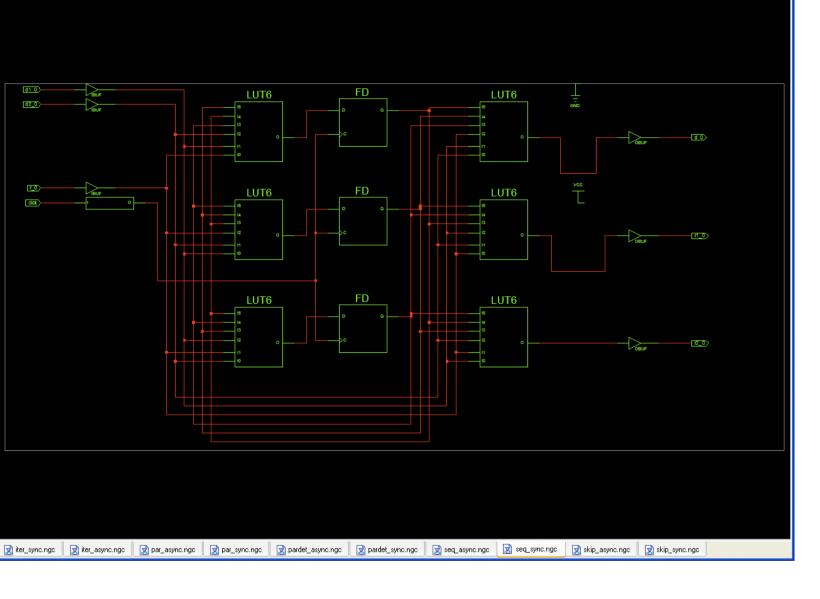


8 states, 26 LUTs

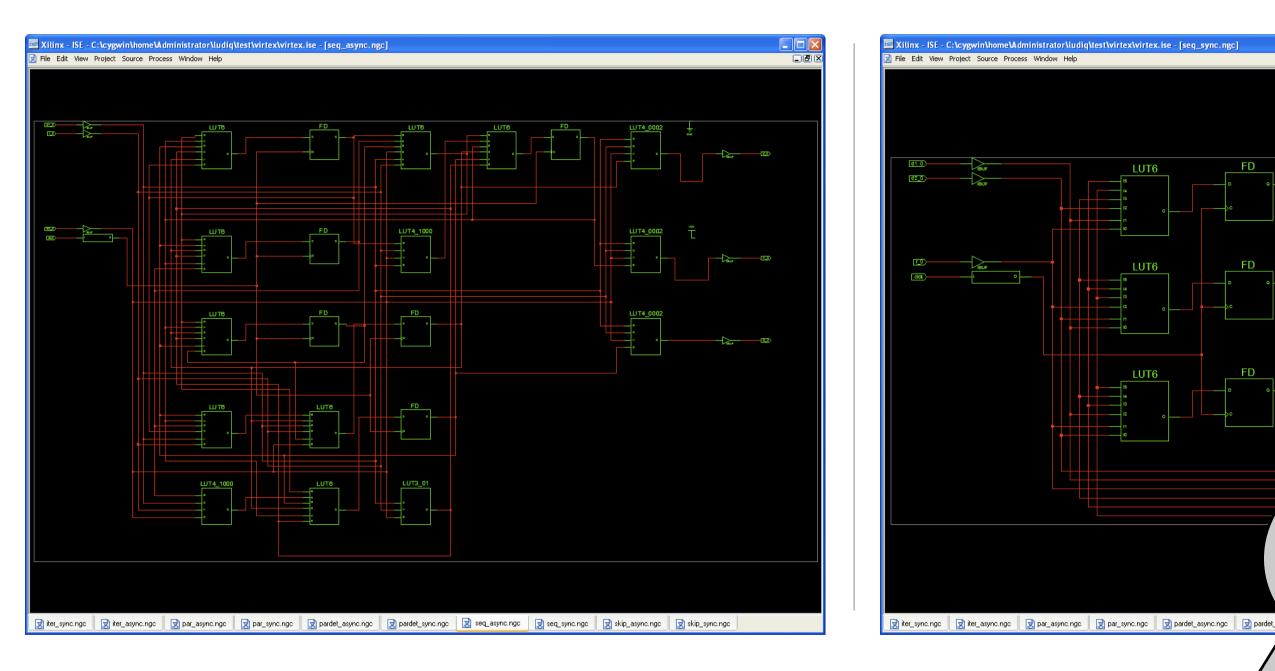
async versus sync representations for sequential composition



B



4 states, 12 LUTs

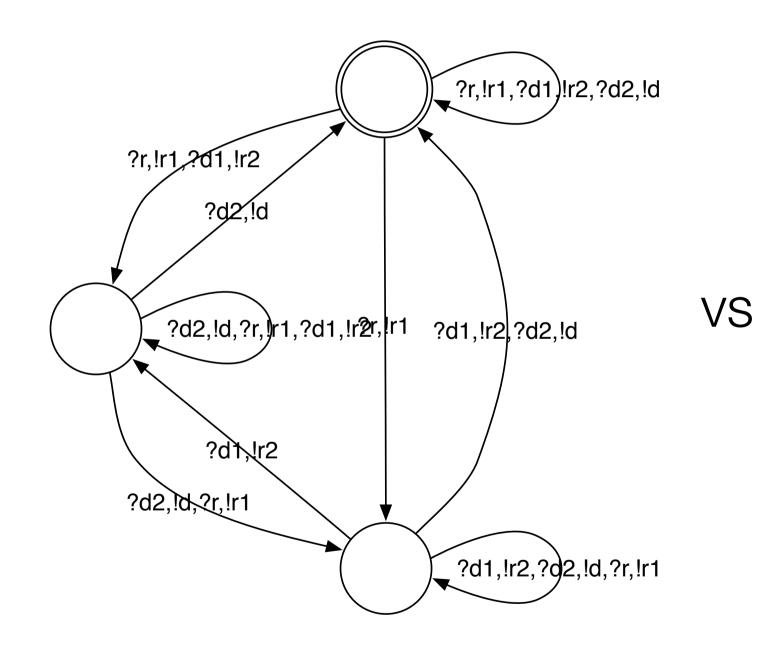


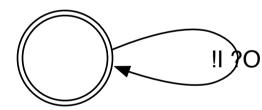
8 states, 26 LUTs

async versus sync representations for sequential composition

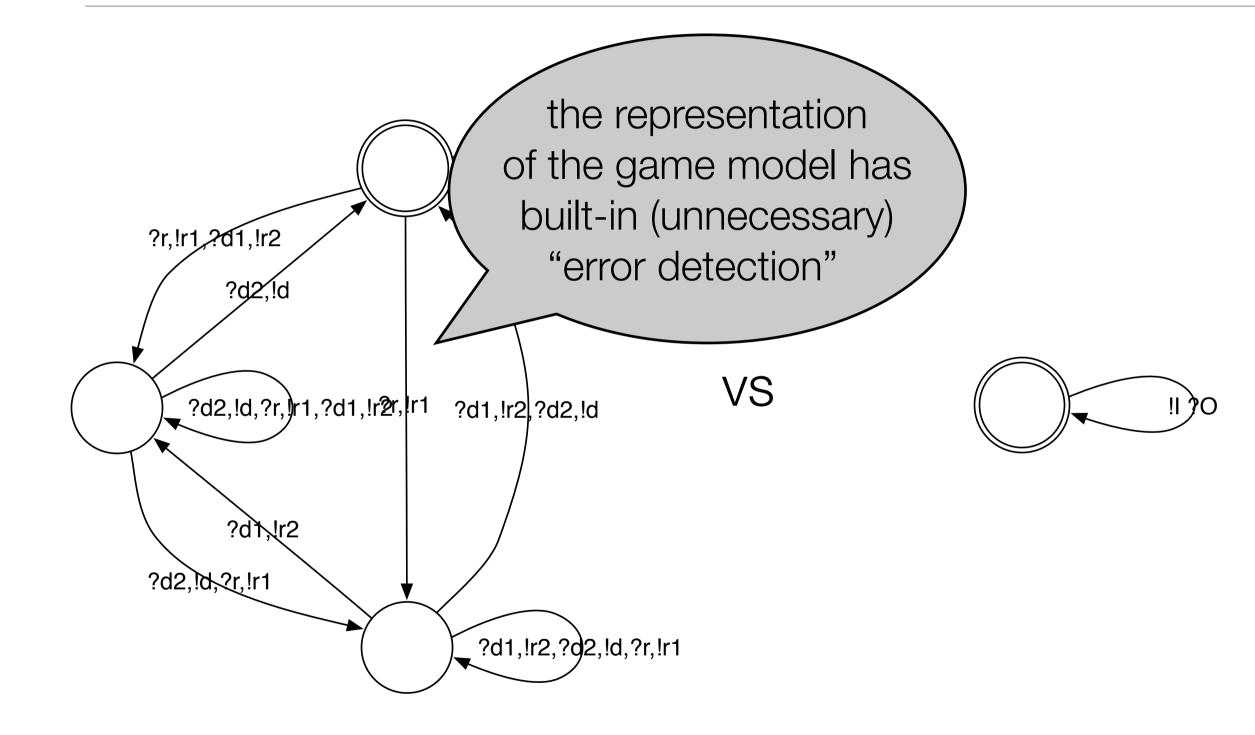
FD LUT6 LUT6 LUT6 12_0 Better, but not just wires! 4 states, 12 LUTs



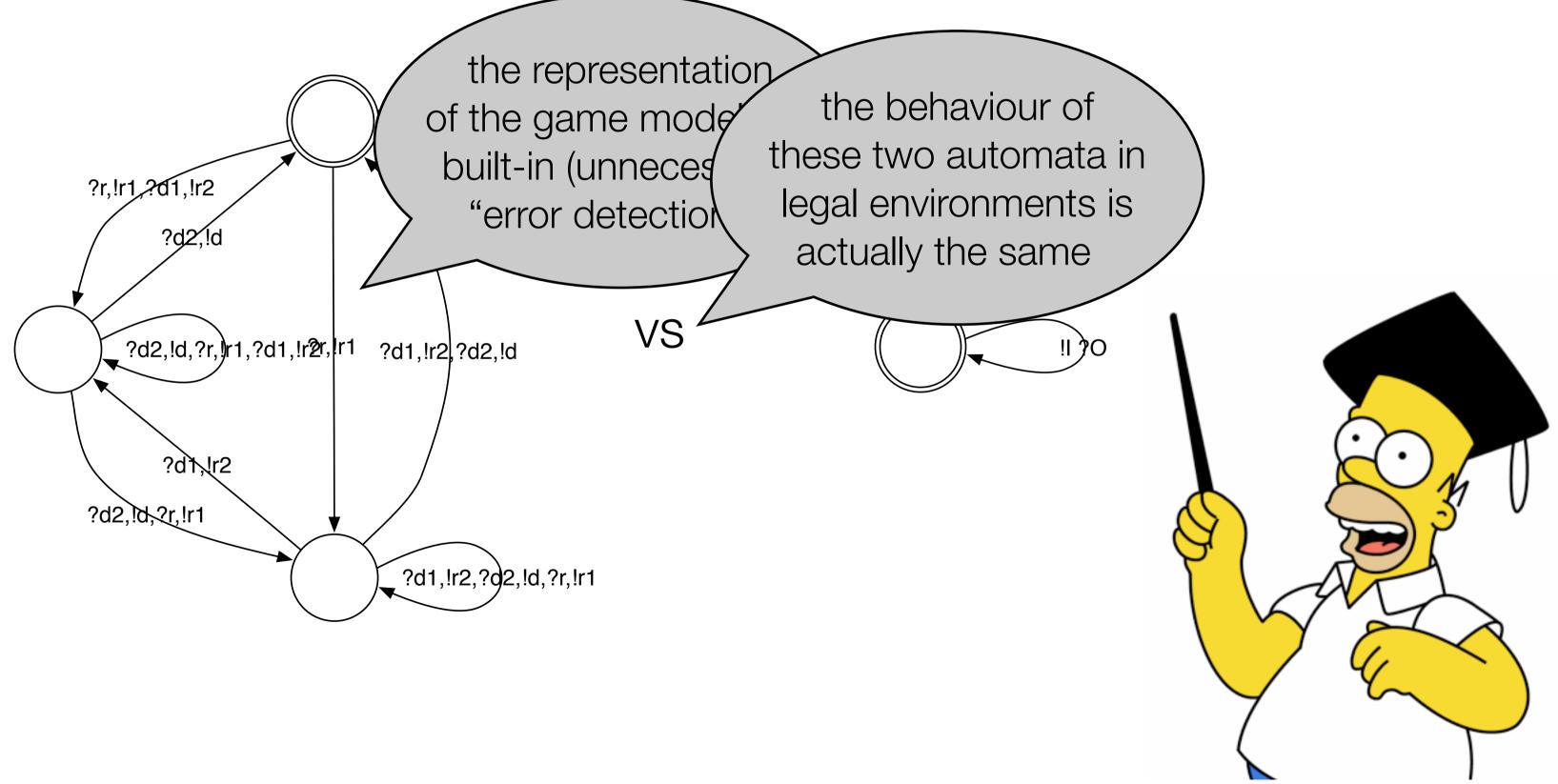




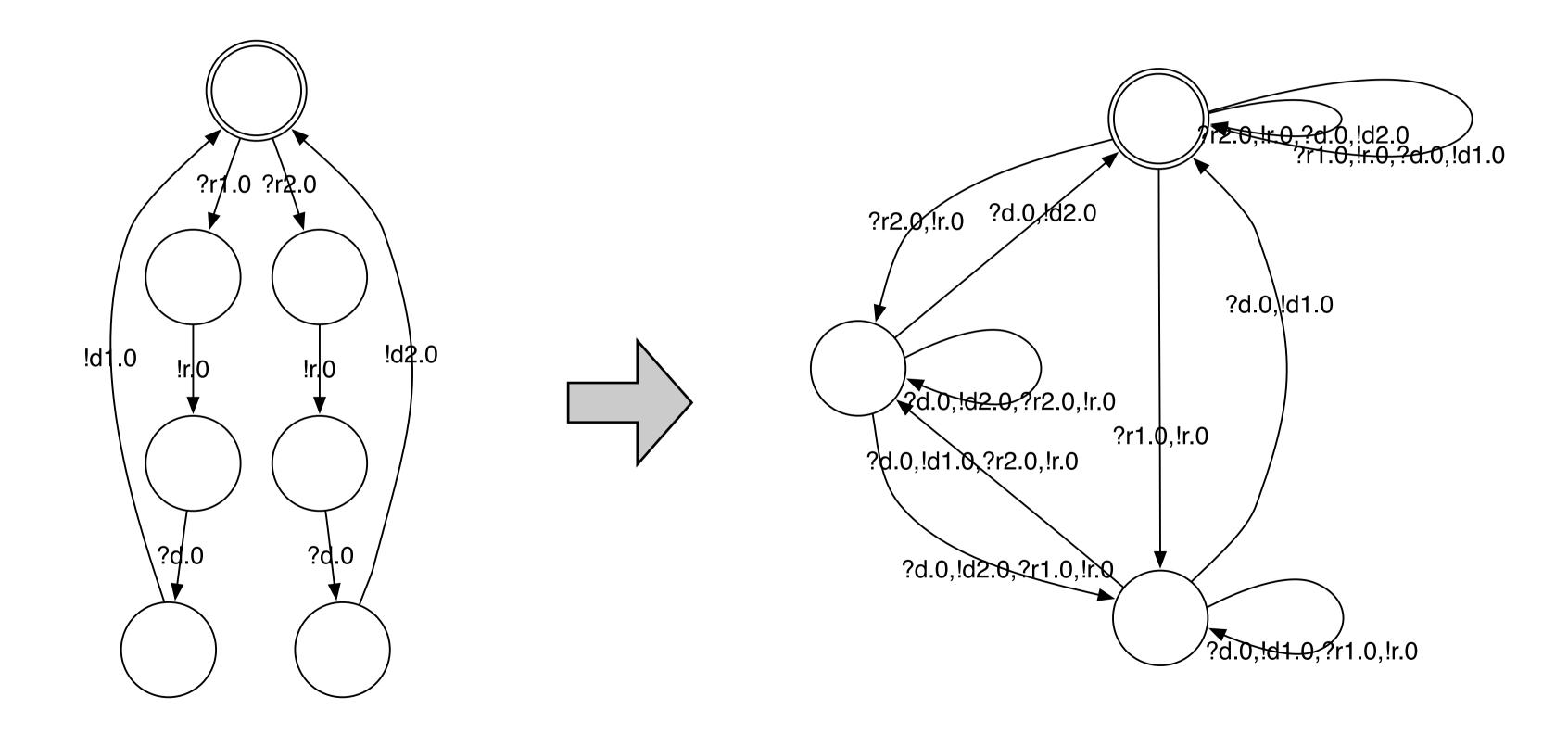


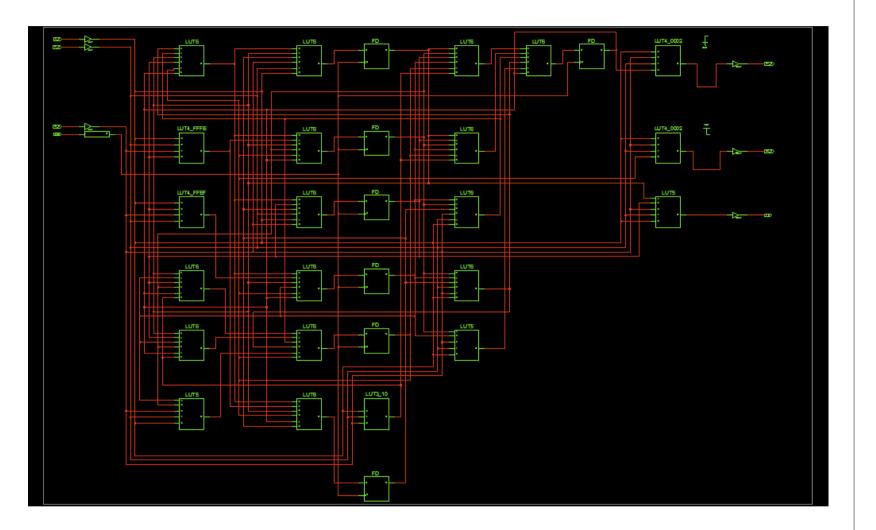


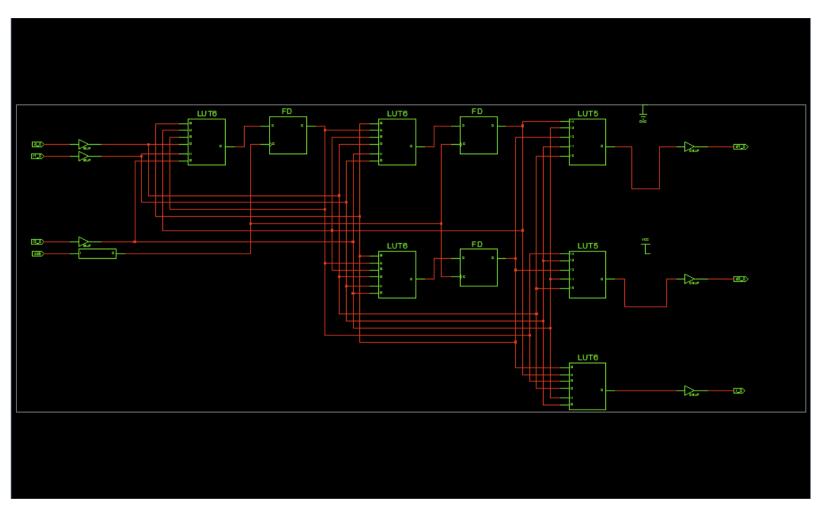




a genuine application: diagonals (bsci)





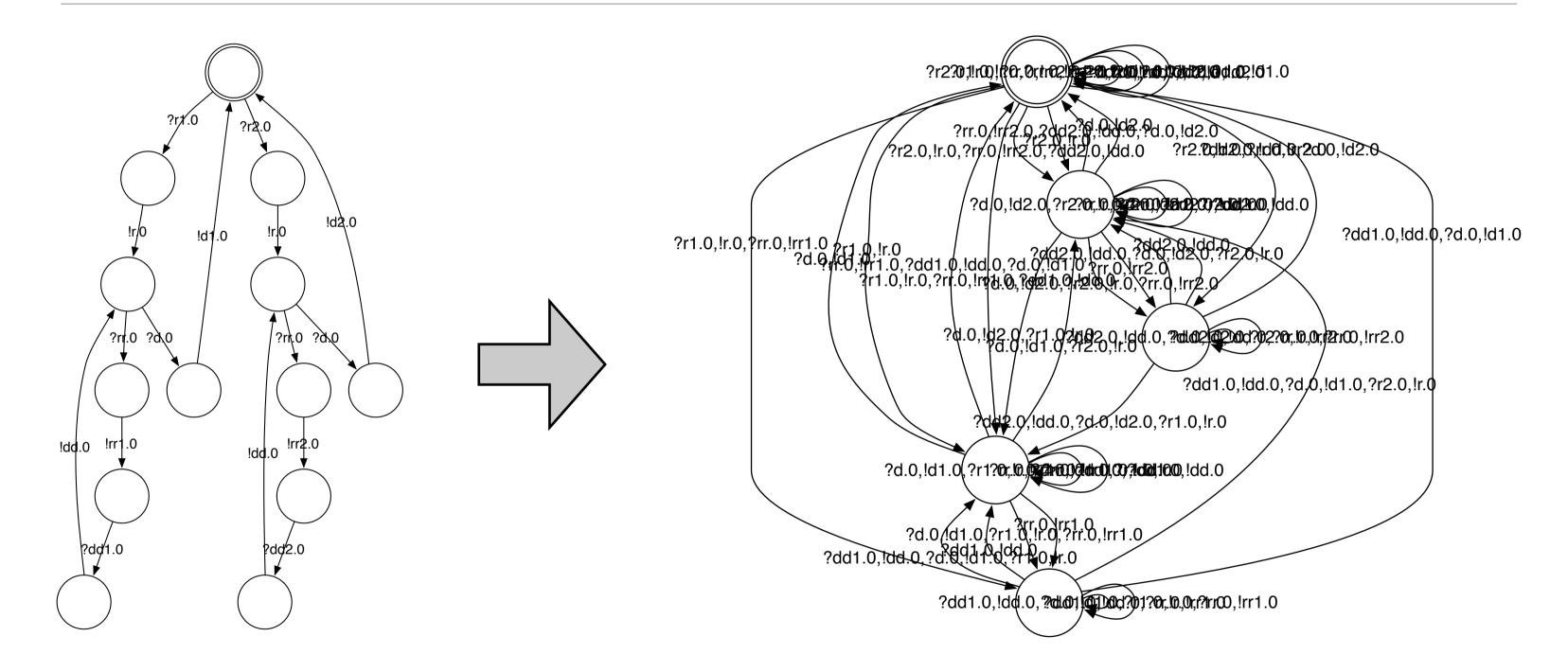


7 registers 22 LUTs

asynchronous vs. synchronous diagonals on com

3 registers 6 LUTs

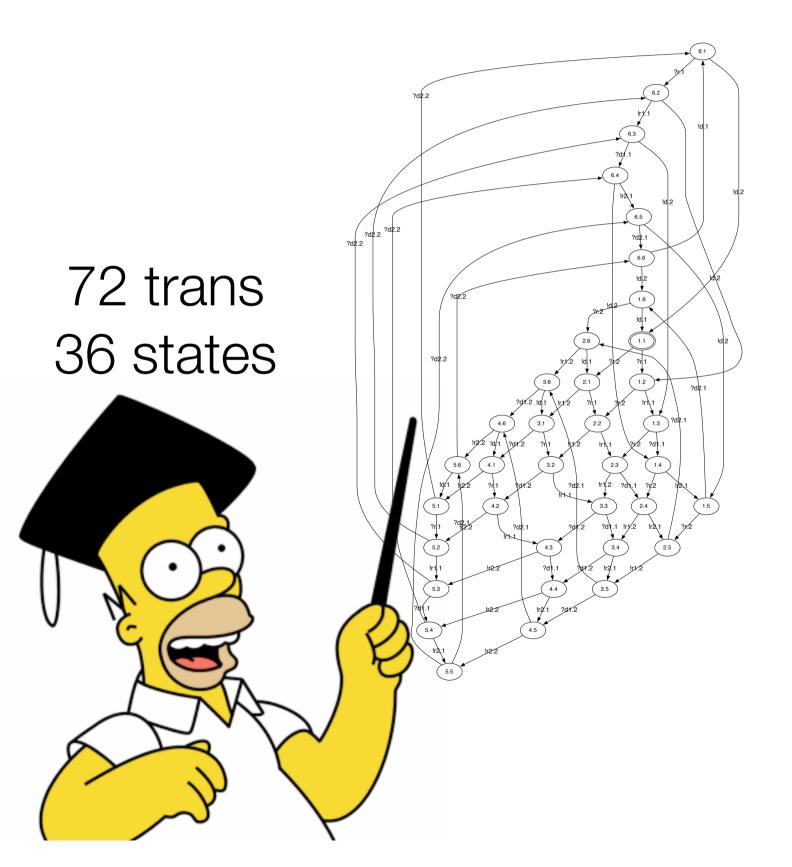
diagonal for $com \Rightarrow com$ (*bsci*)



13 registers 94 LUTs

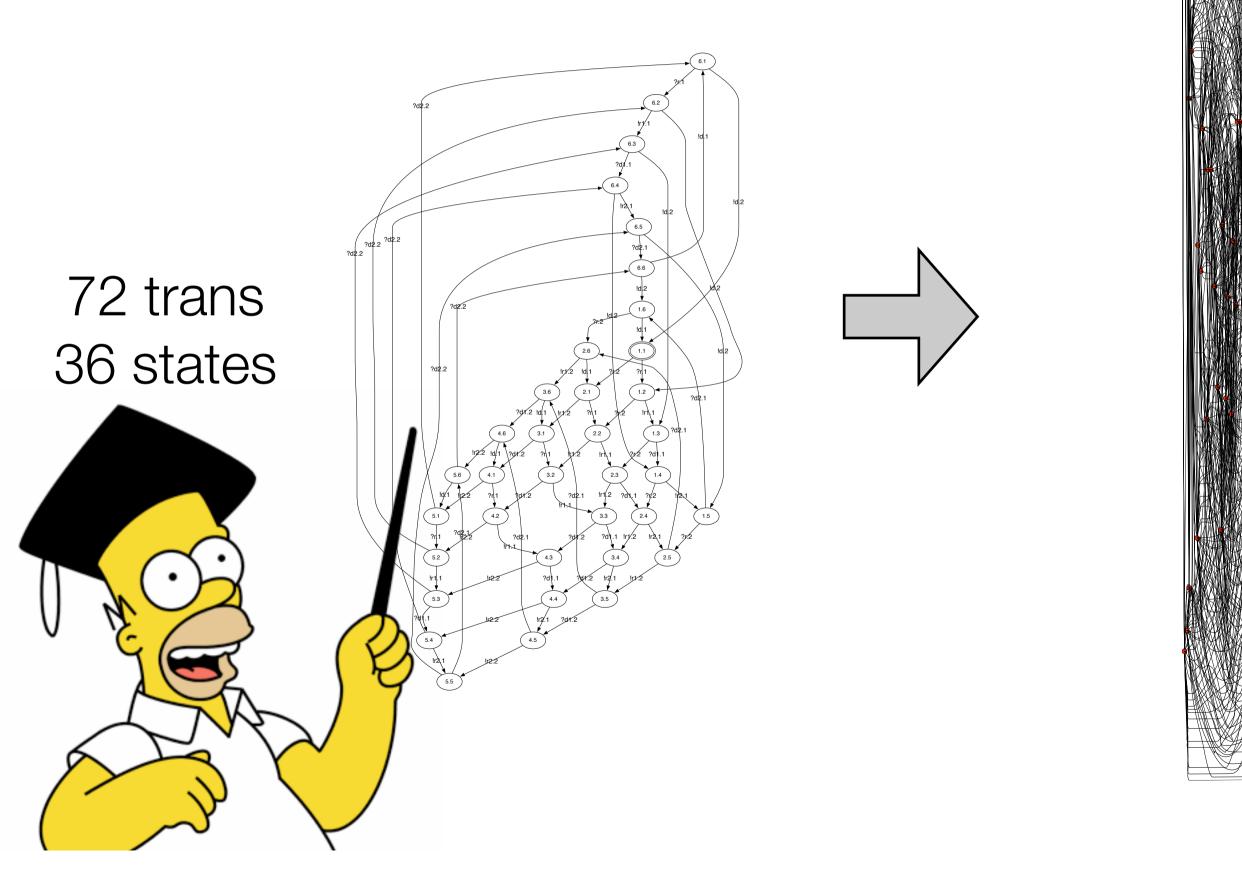
7 registers 77 LUTs

how about concurrent sharing? seq ⊗ seq



28

how about concurrent sharing? seq ⊗ seq



34 states 642 trans (fails synthesis)

conclusion

29

• it is nice to be able to reuse game models

29

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction

ery inefficient ne model can be

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction
 - usually results in smaller circuits (always faster)

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction
 - usually results in smaller circuits (always faster)
 - still room for optimisation (eliminate "error detection")

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction
 - usually results in smaller circuits (always faster)
 - still room for optimisation (eliminate "error detection")
- r.a. can be applied to any game model, not just to game models of constants

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction
 - usually results in smaller circuits (always faster)
 - still room for optimisation (eliminate "error detection")
- r.a. can be applied to any game model, not just to game models of constants
 - peep-hole optimisation for games

- it is nice to be able to reuse game models
 - but naive representation of game model is very inefficient
- synchronous representation of (asynchronous) game model can be done via round abstraction
 - usually results in smaller circuits (always faster)
 - still room for optimisation (eliminate "error detection")
- r.a. can be applied to any game model, not just to game models of constants
 - peep-hole optimisation for games
- concurrent sharing not feasible