TECHNISCHE UNIVERSITÄT DRESDEN

Formal Verification for Dynamic Coordination Models

Sascha Klüppelholz, Christel Baier,
Tobias Blechmann, Joachim Klein
Technische Universität Dresden, Germany

The Outline

Interfaces and Component Connectors

- REO and Constraint Automaton

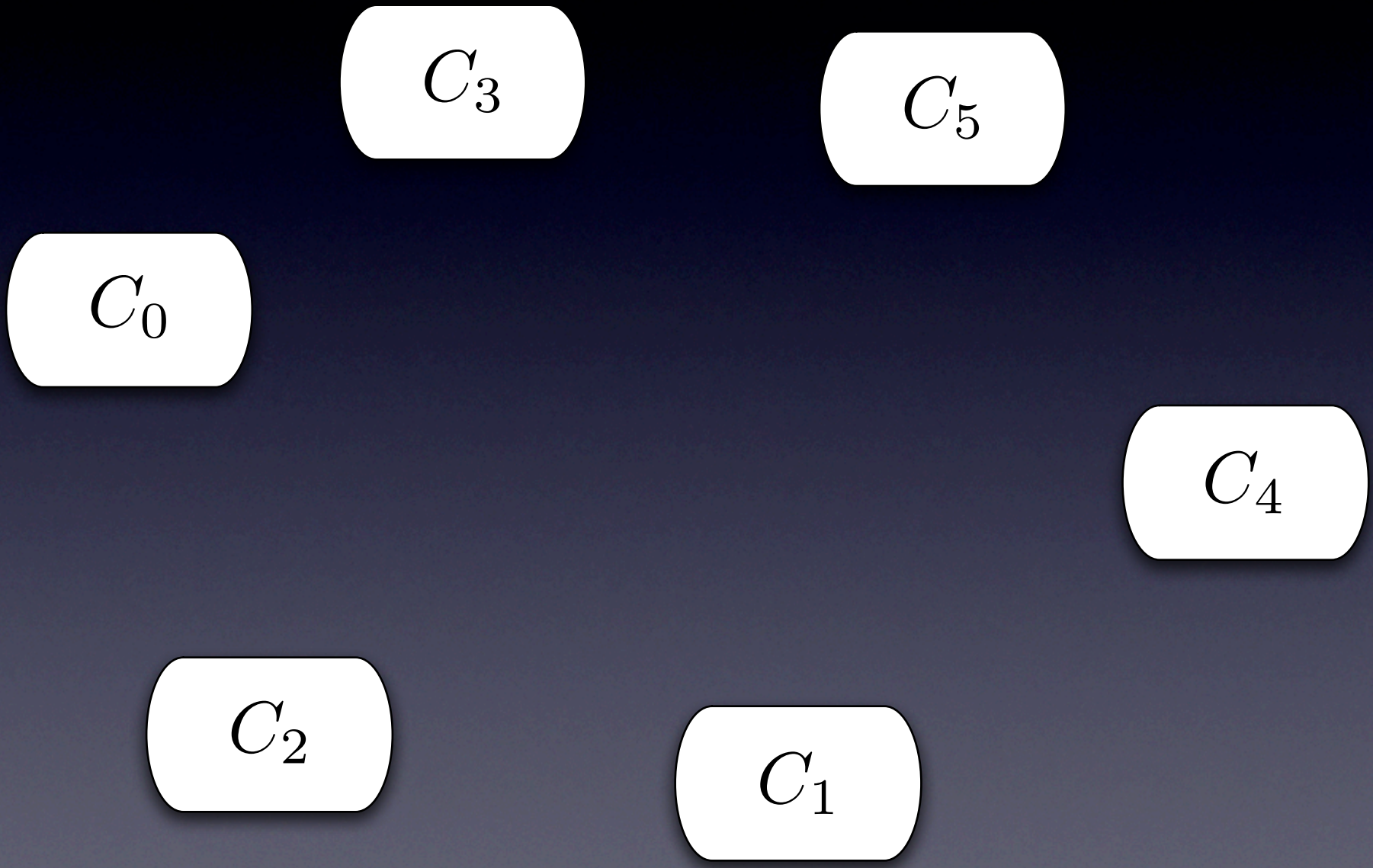
Model Checking

- Temporal Logics

Vereofy

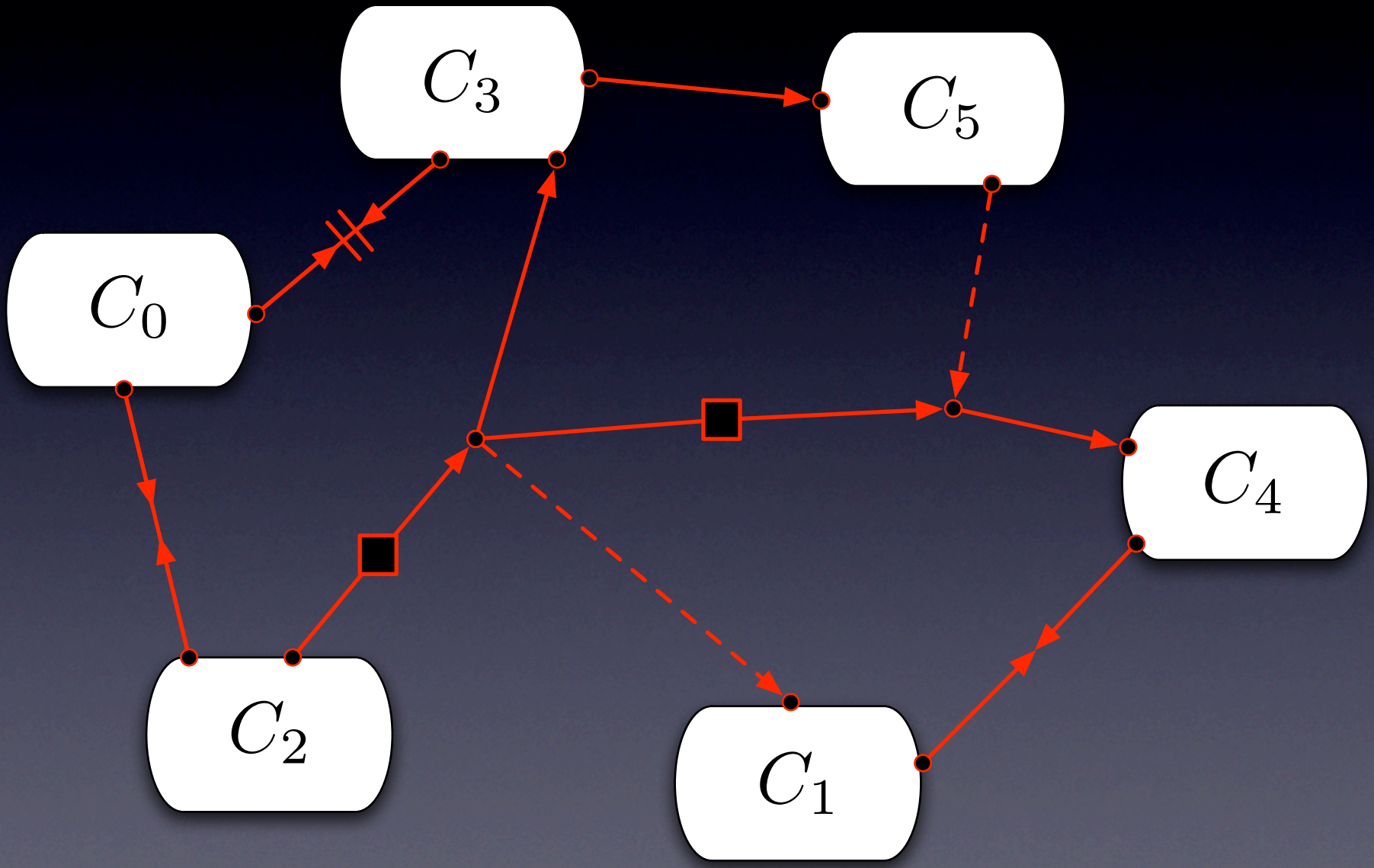
The Framework

The Framework

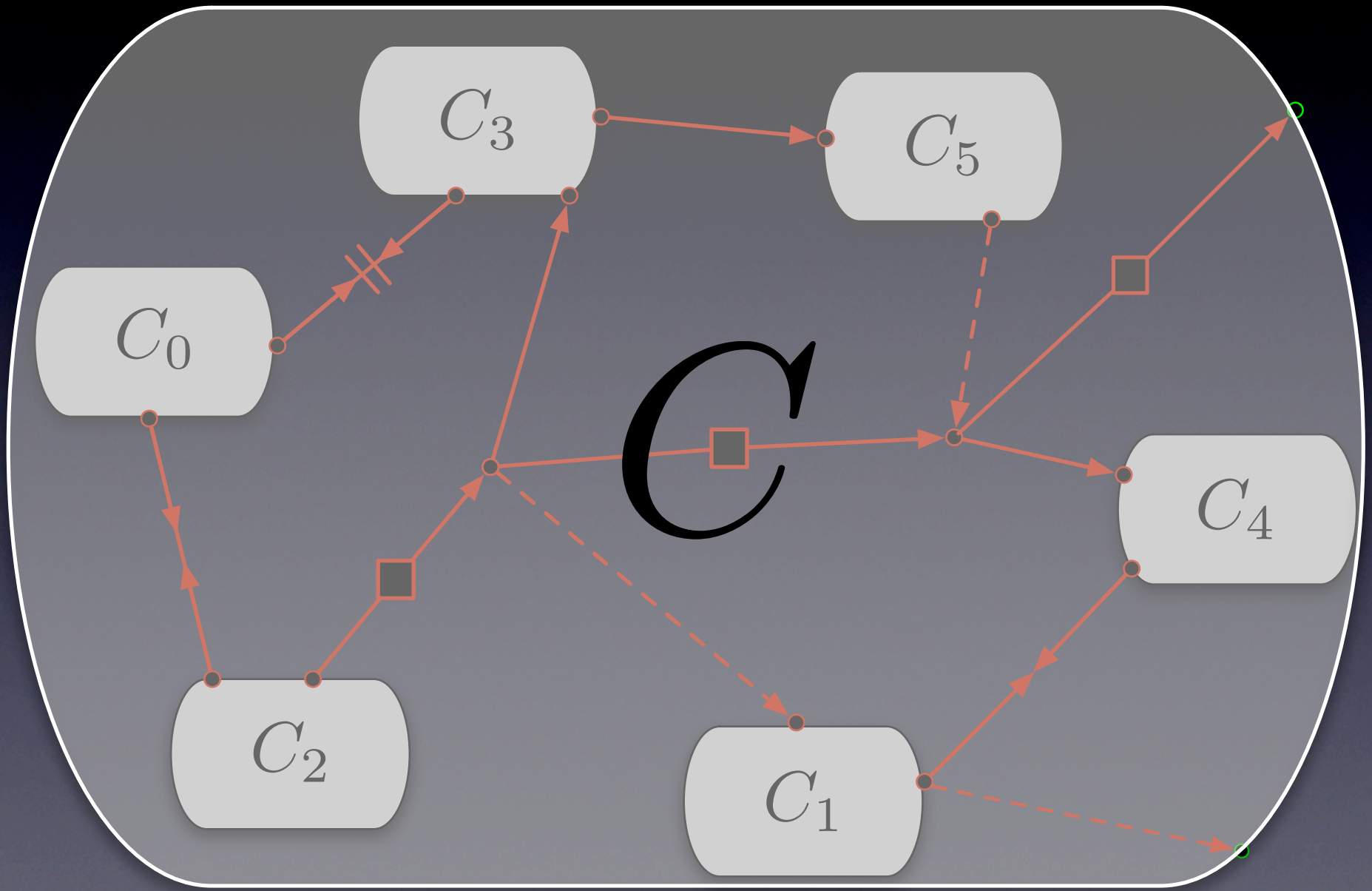


The Framework

Formal Verification for Dynamic Coordinations Models

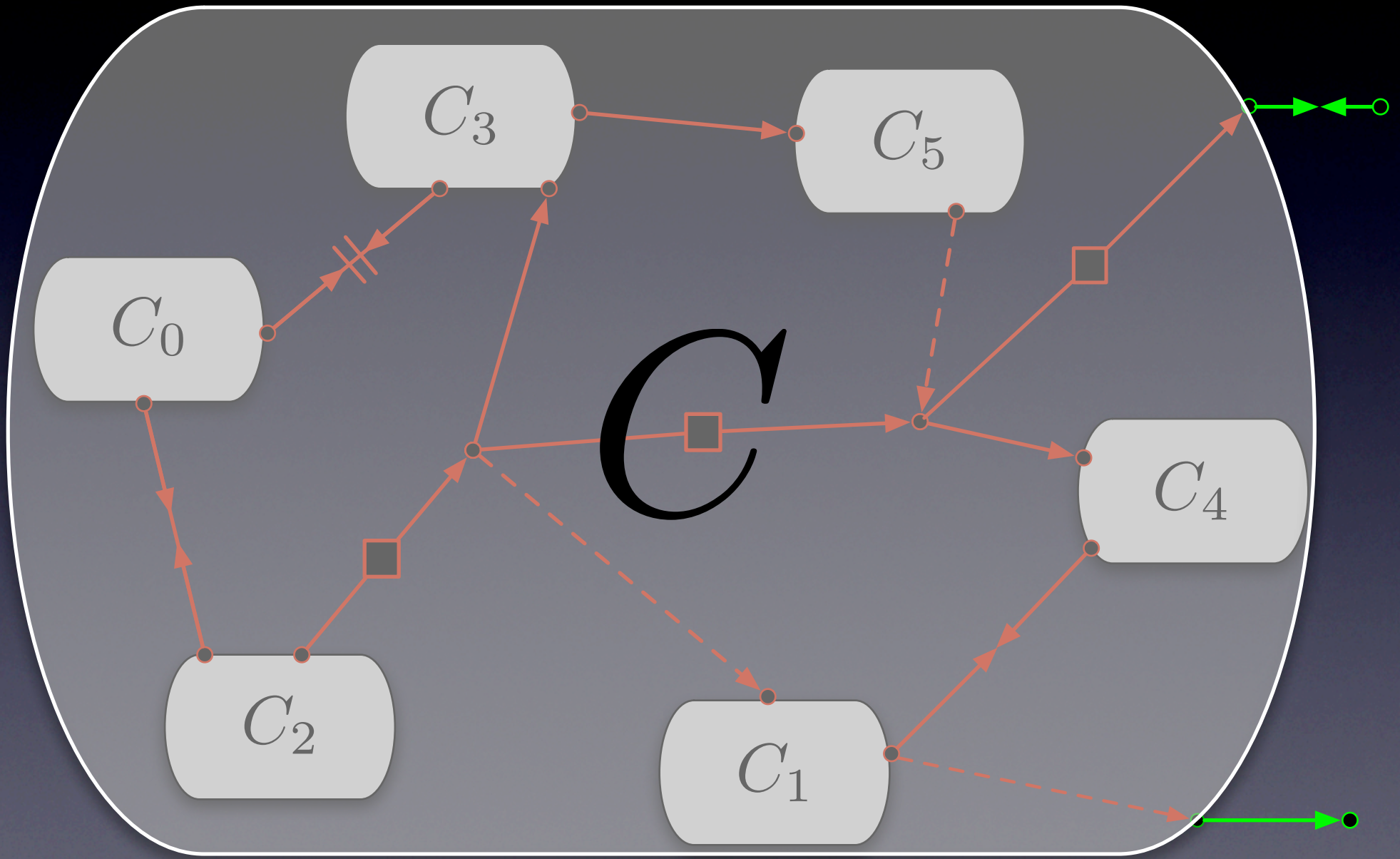


The Framework



The Framework

Formal Verification for Dynamic Coordinations Models



Reo (Arbab et. al.)

Formal Verification for Dynamic Coordinations Models

synchronous channel

lossy synchr. channel

synchronous spout

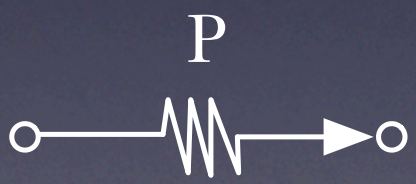
synchronous drain

fifo channel (1-bounded)

lossy fifo channel (1-bounded)

asynchronous spout

asynchronous drain

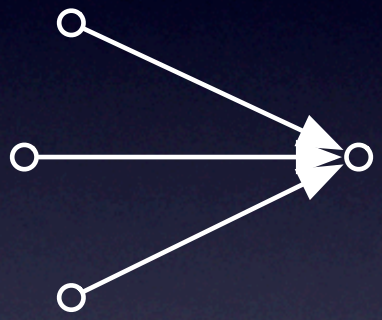


filter channel

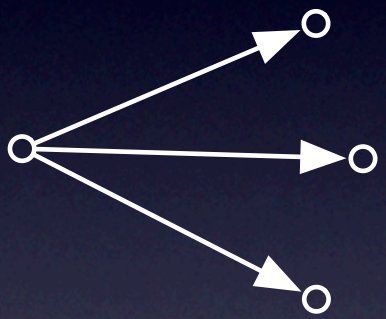
P-producer

+ Operators

JOIN

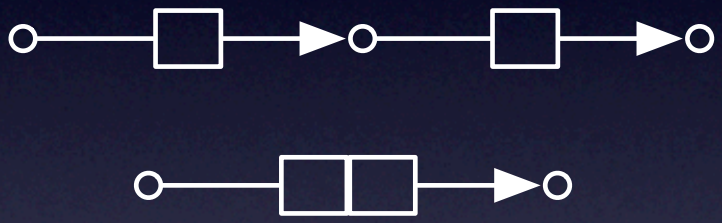


merge



copy

HIDE

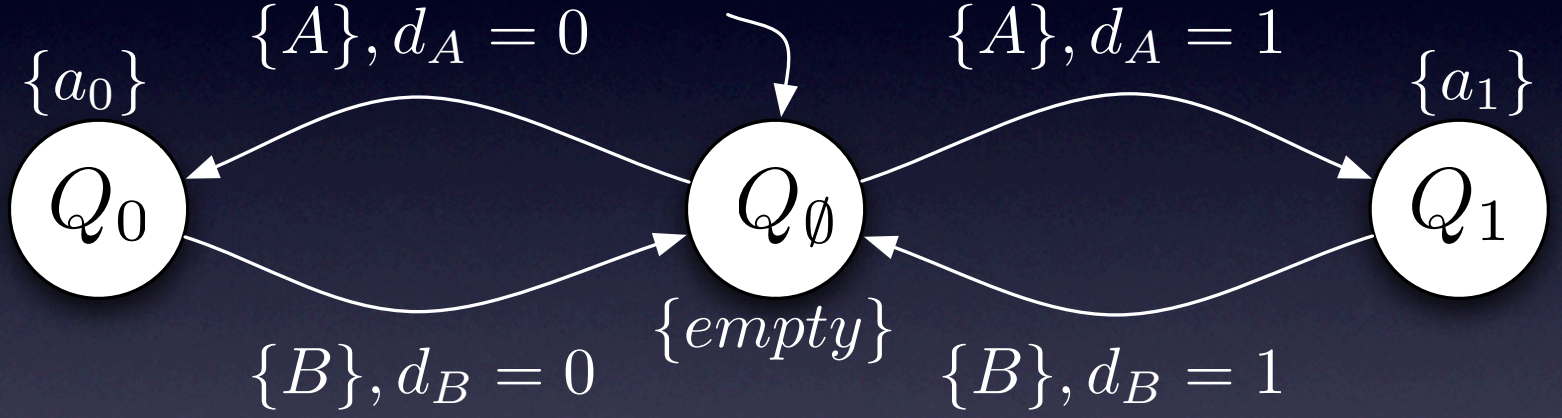


Constraint Automata Example

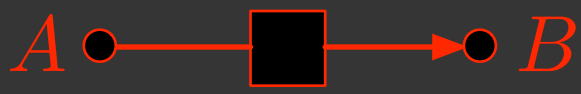
$$\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0, AP, L) \qquad (Data = \{0, 1\})$$

Constraint Automata Example

$$\mathcal{A} = (Q, \mathcal{N}, \longrightarrow, Q_0, AP, L) \qquad (Data = \{0, 1\})$$



$\mathcal{N}^{src} = \{A\}$ and $\mathcal{N}^{snk} = \{B\}$



JOIN (& HIDE) for CA

JOIN (& HIDE) for CA

$$\frac{q_1 \xrightarrow{N,g} q_2,\ N \cap \mathcal{N}_2 = \emptyset}{(q_1, p_1) \xrightarrow{N,g} (q_2, p_1)} \qquad \frac{p_1 \xrightarrow{N,g} p_2,\ N \cap \mathcal{N}_1 = \emptyset}{(q_1, p_1) \xrightarrow{N,g} (q_1, p_2)}$$

JOIN (& HIDE) for CA

$$\frac{q_1 \xrightarrow{N,g} q_2,\ N \cap \mathcal{N}_2 = \emptyset}{(q_1,p_1) \xrightarrow{N,g} (q_2,p_1)} \qquad \frac{p_1 \xrightarrow{N,g} p_2,\ N \cap \mathcal{N}_1 = \emptyset}{(q_1,p_1) \xrightarrow{N,g} (q_1,p_2)}$$

$$\frac{q_1 \xrightarrow{N_1,g_1} q_2,\ p_1 \xrightarrow{N_2,g_2} p_2,\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1}{(q_1,p_1) \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} (q_2,p_2)}$$

The Outline

Interfaces and Component Connectors

- REO and Constraint Automaton

Model Checking

- Temporal Logics

Vereofy

Branching Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

Branching Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid \Phi_1 U \Phi_2$$

Branching Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid \Phi_1 U \Phi_2$$

$$\alpha ::= (N, g) \mid stop \mid \alpha^* \mid \neg\alpha$$

$$\alpha_1 \,;\, \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \cap \alpha_2$$

Branching Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid \Phi_1 U \Phi_2$$

$$\alpha ::= (N, g) \mid stop \mid \alpha^* \mid \neg\alpha$$

$$\alpha_1 \,;\, \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \cap \alpha_2$$

Derived Operators:

$$\exists[\alpha]\Phi = \neg\forall\langle\alpha\rangle\neg\Phi \qquad \forall[\alpha]\Phi = \neg\exists\langle\alpha\rangle\neg\Phi$$

Branching Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid \Phi_1 U \Phi_2$$

$$\alpha ::= (N, g) \mid stop \mid \alpha^* \mid \neg\alpha$$

$$\alpha_1 \,;\, \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \cap \alpha_2$$

Derived Operators:

$$\exists[\alpha]\Phi = \neg\forall\langle\alpha\rangle\neg\Phi \qquad \forall[\alpha]\Phi = \neg\exists\langle\alpha\rangle\neg\Phi$$

$$\exists X\Phi = \exists\langle\cdot\rangle\Phi \qquad \forall X\Phi = \forall[\cdot]\Phi$$

Alternating Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{E}_N \varphi$$

Alternating Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{E}_N\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid [\alpha]\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 R \Phi_2$$

Alternating Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{E}_N\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid [\alpha]\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 R \Phi_2$$

$$\alpha ::= c \mid stop \mid \alpha^* \mid \neg\alpha$$

$$\alpha_1 \,;\, \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \cap \alpha_2$$

Alternating Time Stream Logic

Formal Verification for Dynamic Coordinations Models

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{E}_N\varphi$$

$$\varphi ::= \langle\alpha\rangle\Phi \mid [\alpha]\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 R \Phi_2$$

$$\alpha ::= c \mid stop \mid \alpha^* \mid \neg\alpha$$

$$\alpha_1 \,;\, \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \cap \alpha_2$$

Derived Operator:

$$\mathbb{A}_N[\alpha]\Phi = \neg\mathbb{E}_N\langle\alpha\rangle\neg\Phi$$

$$\mathbb{A}_N\langle\alpha\rangle\Phi = \neg\mathbb{E}_N[\alpha]\neg\Phi$$

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 U \varphi_2$$

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 U \varphi_2$$

Derived Operators:

$$\Diamond\varphi = true\ U\varphi \qquad \Box\varphi = \neg\Diamond\neg\varphi$$

Linear Time Logic

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 U \varphi_2$$

Derived Operators:

$$\Diamond\varphi = true\ U\varphi \qquad \Box\varphi = \neg\Diamond\neg\varphi$$

Interesting Properties: $\Box\Diamond a,\ \Diamond\Box a,\ \Box\Diamond a \rightarrow \Box\Diamond b$

Linear Time Logic

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 U \varphi_2$$

Derived Operators:

$$\Diamond\varphi = true\ U\varphi \qquad \Box\varphi = \neg\Diamond\neg\varphi$$

Interesting Properties: $\Box\Diamond a,\ \Diamond\Box a,\ \Box\Diamond a \rightarrow \Box\Diamond b$

Model Checking: $\mathcal{O}(2^{|\varphi|} \cdot |\mathcal{A}|)$

$$\mathcal{A} \models \varphi \ \Leftrightarrow\ \mathrm{Traces}(\mathcal{A}) \cap \mathrm{Words}(\neg\varphi) = \emptyset$$

The Outline

- Interfaces and Component Connectors
  - REO and Constraint Automaton
- Model Checking
  - Temporal Logics
- Vereofy

TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy

Formal Verification for Dynamic Coordinations Models

Verification Tool for REO and Constraint Automaton

Verification Tool for REO and Constraint Automaton

Linux, Mac, and Windows

Verification Tool for
REO and Constraint Automaton

Linux, Mac, and Windows

Symbolic (BDD-based)
new heuristics for variable ordering

Vereofy

Formal Verification for Dynamic Coordinations Models

Verification Tool for
REO and Constraint Automaton

Linux, Mac, and Windows

Symbolic (BDD-based)
new heuristics for variable ordering

Dotty: Network & System Model

Vereofy

Formal Verification for Dynamic Coordinations Models

Verification Tool for
REO and Constraint Automaton

Linux, Mac, and Windows

Symbolic (BDD-based)
new heuristics for variable ordering

Dotty: Network & System Model

Input languages for
REO + Reactive Modules

Vereofy

Formal Verification for Dynamic Coordinations Models

Verification Tool for
REO and Constraint Automaton

Linux, Mac, and Windows

Symbolic (BDD-based)
new heuristics for variable ordering

Dotty: Network & System Model

Input languages for
REO + Reactive Modules

Plugin for ECT GUI (CWI)

TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL Model Checking

CTL / BTSL
Model Checking

LTL Model Checking

Vereofy Key Features

CTL / BTSL
Model Checking

LTL Model Checking

ASL Model Checking

Vereofy Key Features

CTL / BTSL
Model Checking

LTL Model Checking

ASL Model Checking

(BTSL* Model Checking)

Vereofy Key Features

CTL / BTSL
Model Checking

LTL Model Checking

ASL Model Checking

(BTSL* Model Checking)

Dynamic Reconfiguration

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL
Model Checking

LTL Model Checking

ASL Model Checking

(BTSL* Model Checking)

(Compatibility)

Dynamic Reconfiguration

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL
Model Checking

LTL Model Checking

ASL Model Checking

Counterexamples/
Witnesses

(BTSL* Model Checking)

(Compatibility)

Dynamic Reconfiguration

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL
Model Checking

LTL Model Checking

Synthesis

ASL Model Checking

Counterexamples/
Witnesses

(BTSL* Model Checking)

(Compatibility)

Dynamic Reconfiguration

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL
Model Checking

CA-Bisimulation

LTL Model Checking

Synthesis

ASL Model Checking

Counterexamples/
Witnesses

(BTSL* Model Checking)

(Compatibility)

Dynamic Reconfiguration

Vereofy Key Features

Formal Verification for Dynamic Coordinations Models

CTL / BTSL Model Checking

(Minimization)

CA-Bisimulation

LTL Model Checking

Synthesis

ASL Model Checking

Counterexamples/ Witnesses

(BTSL* Model Checking)

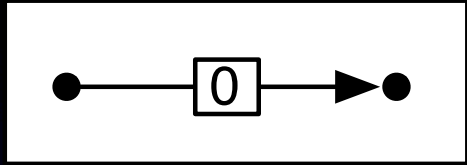
(Compatibility)

Dynamic Reconfiguration

Vereofy - Modules

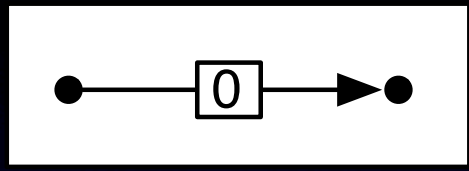
Formal Verification for Dynamic Coordinations Models

Vereofy - Modules



Vereofy - Modules

Formal Verification for Dynamic Coordinations Models




```
MODULE FIFO1_INIT_RM<k>{
  in: A;
  out: B;

  var: enum {EMPTY, FULL} state := FULL;
  var: Data value := k;

  state==EMPTY -[ {A} ]-> state:=FULL & value:=#A;
  state==FULL -[ {B} & #B==value ]-> state:=EMPTY;
}
```

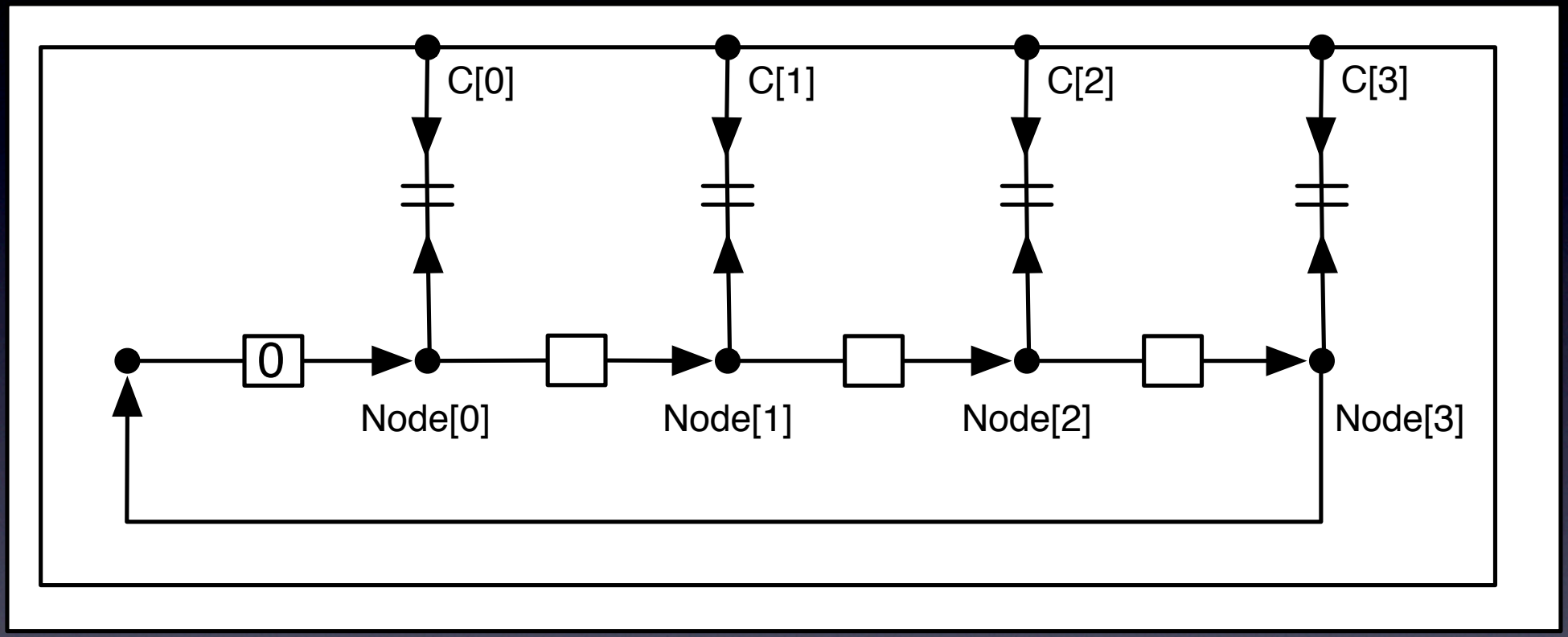
TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Circuits

Formal Verification for Dynamic Coordinations Models

Vereofy - Circuits

Formal Verification for Dynamic Coordinations Models



TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Circuits

Formal Verification for Dynamic Coordinations Models

Vereofy - Circuits


```
CIRCUIT my_sequencer<k>{
  F[0] = new FIFO1_INIT<0>(A[0];B[0]);
  SD[0] = new SYNC_DRAIN(C[0],D[0]);
  Node[0] = join(B[0],D[0]);

  for (i=1;i<k;i=i+1){
    F[i] = new FIFO1(A[i];B[i]);
    SD[i] = new SYNC_DRAIN(C[i],D[i]);
    Node[i] = join(B[i],D[i]);
    Node[i-1] = join(Node[i-1],A[i]);
  }

  Node[k-1] = join(Node[k-1],A[0]);
}
```

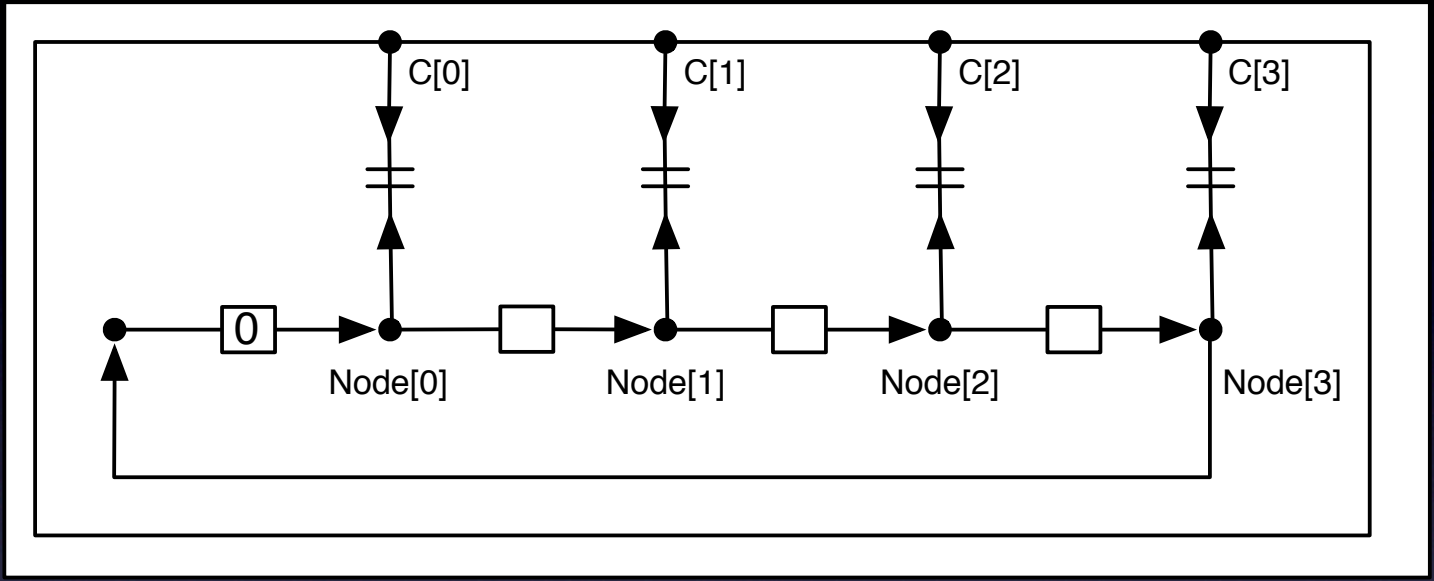
TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Circuits

Formal Verification for Dynamic Coordinations Models

Vereofy - Circuits

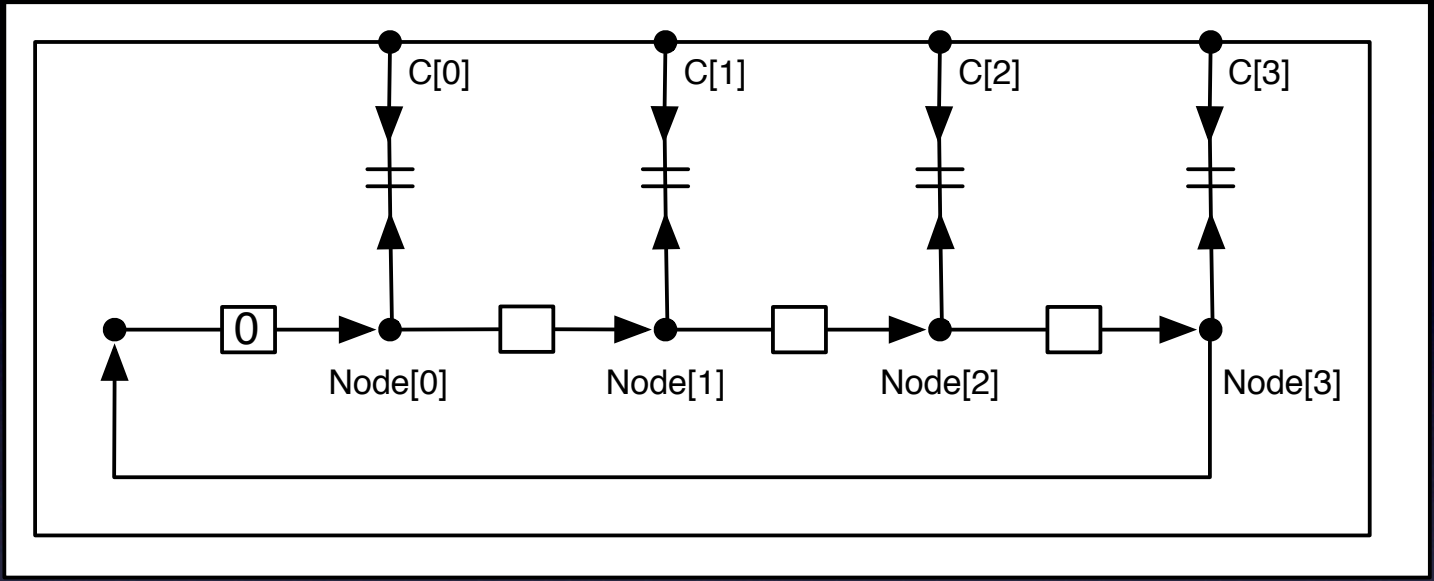
Formal Verification for Dynamic Coordinations Models



for (i=0;i<k;i=i+1){ source[i] = C[i]; }

Vereofy - Circuits

Formal Verification for Dynamic Coordinations Models




```
for (i=0;i<k;i=i+1){ source[i] = C[i]; }
```

```
CIRCUIT main{
  SEQ = new my_sequencer<size>;
  N = join(SEQ.source[size-1], ...);
}
```

TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Dynamic Modules

Formal Verification for Dynamic Coordinations Models

Vereofy - Dynamic Modules

Formal Verification for Dynamic Coordinations Models

```
TYPE Data = enum {X,Y, rsignal};

MODULE dynamic_component{
 in: A; out: B1;
 in: reconf;

 var: enum {Mode0, Mode1} Mode := Mode0;

 // behaviour:
 Mode==Mode0 -[ {A} ]->;
 Mode==Mode1 -[ {A,B1} & #A==#B1 ]->;

 // reconfiguration:
 Mode!=Mode0 -[ {reconf} & #reconf==rsignal]-> Mode:=Mode0;
 Mode!=Mode1 -[ {reconf} & #reconf==rsignal]-> Mode:=Mode1;
}
```

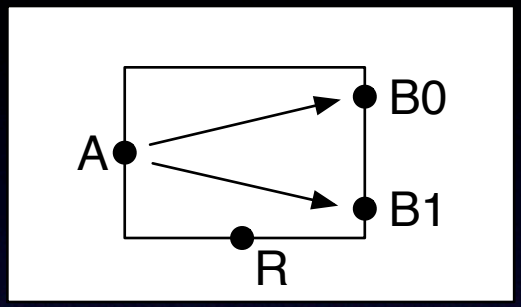
TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Dynamic Circuits

Formal Verification for Dynamic Coordinations Models

Vereofy - Dynamic Circuits

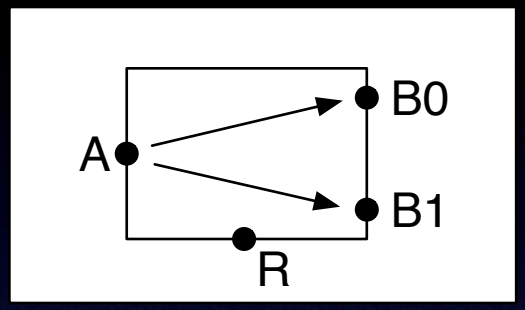
Formal Verification for Dynamic Coordinations Models

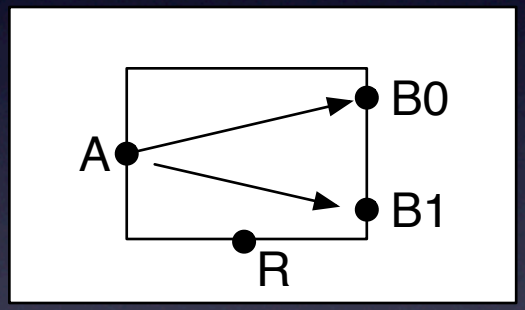



```
RECONFIGURABLE CIRCUIT dynamic_circuit{
  source[0] := A;
  sink[0] := B0; sink[1] := B1;
  SC[0] = new SYNC; SC[1] = new SYNC;

  conf(1){
    join(SC[0].source[0],A);
    join(SC[0].sink[0],B0);
  }

  conf(2){
    join(SC[1].source[0],A);
    join(SC[1].sink[0],B1);
  }
}
```

Vereofy - Dynamic Circuits

Formal Verification for Dynamic Coordinations Models

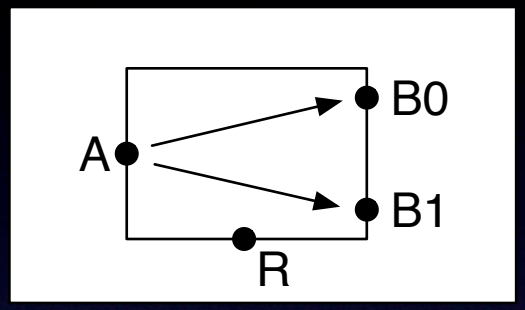


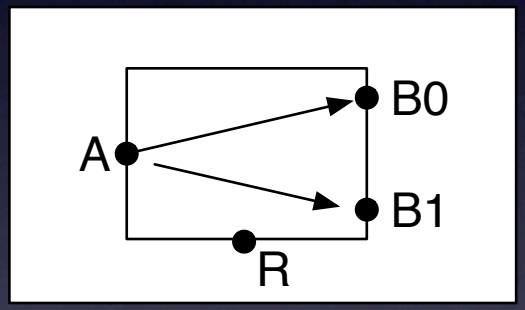


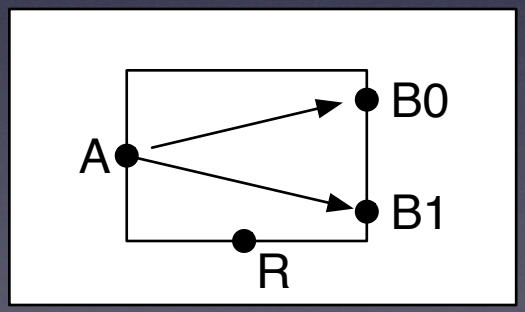

```
RECONFIGURABLE CIRCUIT dynamic_circuit{
  source[0] := A;
  sink[0] := B0; sink[1] := B1;
  SC[0] = new SYNC; SC[1] = new SYNC;

  conf(1){
    join(SC[0].source[0],A);
    join(SC[0].sink[0],B0);
  }

  conf(2){
    join(SC[1].source[0],A);
    join(SC[1].sink[0],B1);
  }
}
```

Vereofy - Dynamic Circuits

Formal Verification for Dynamic Coordinations Models








```
RECONFIGURABLE CIRCUIT dynamic_circuit{
  source[0] := A;
  sink[0] := B0; sink[1] := B1;
  SC[0] = new SYNC; SC[1] = new SYNC;

  conf(1){
    join(SC[0].source[0],A);
    join(SC[0].sink[0],B0);
  }

  conf(2){
    join(SC[1].source[0],A);
    join(SC[1].sink[0],B1);
  }
}
```

TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Dynamic Circuits

Formal Verification for Dynamic Coordinations Models

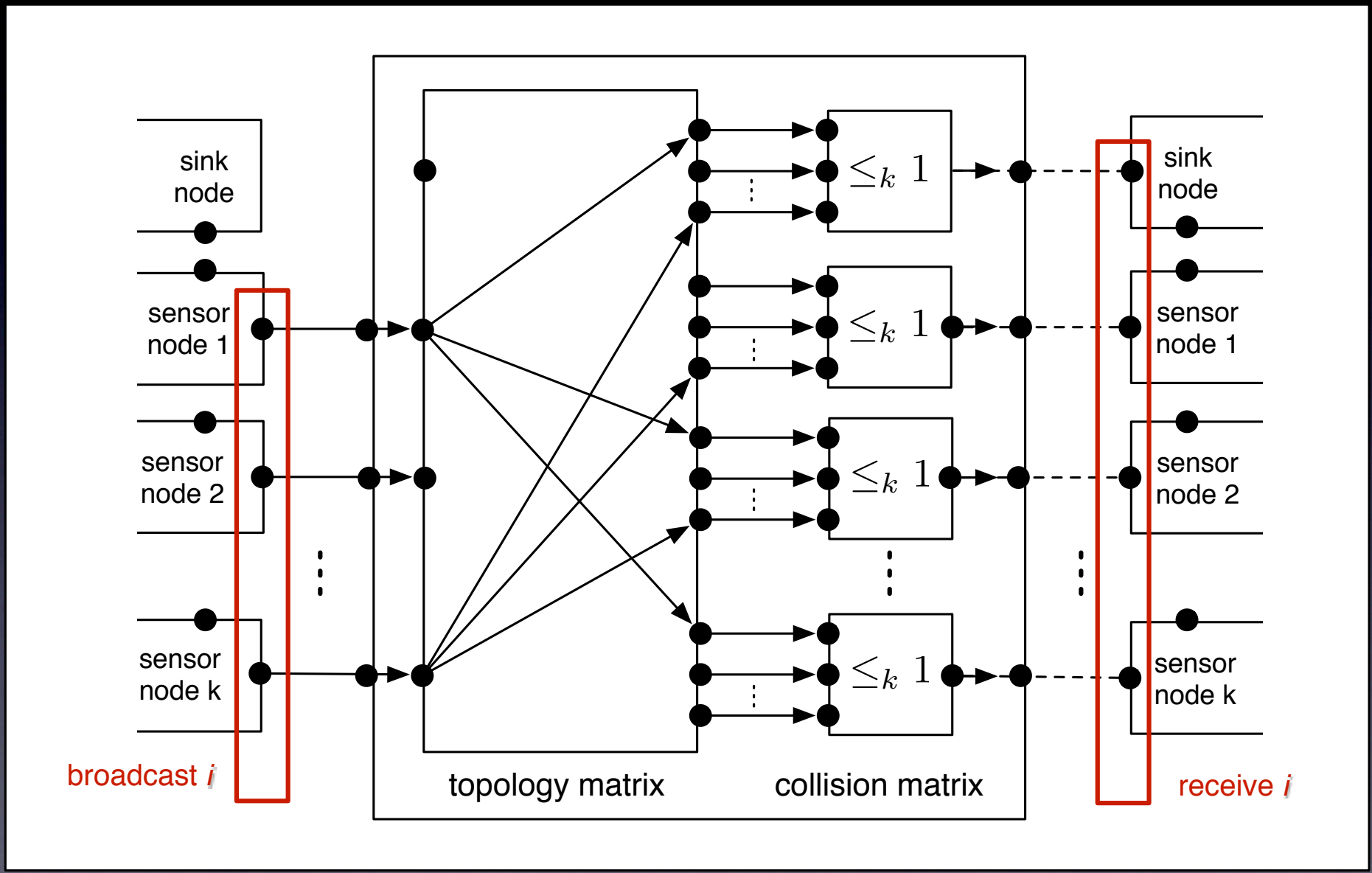
Vereofy - Dynamic Circuits

Formal Verification for Dynamic Coordinations Models

```
CIRCUIT main{
  dc = new dynamic_circuit;
  dr = new my_driver;

  join(dr.sink[0], dc.reconf_port);
}
```

Vereofy - BSN

Formal Verification for Dynamic Coordinations Models



TECHNISCHE UNIVERSITÄT DRESDEN

Vereofy - Release

Formal Verification for Dynamic Coordinations Models

Vereofy V1.0 release date: Nov. 15, 2008

Vereofy - Release

Formal Verification for Dynamic Coordinations Models

Vereofy V1.0 release date: Nov. 15, 2008

www.vereofy.de

Vereofy - Release

Formal Verification for Dynamic Coordinations Models

Vereofy V1.0 release date: Nov. 15, 2008

www.vereofy.de

vereofy@tcs.inf.tu-dresden.de

The Final Slide

Formal Verification for Dynamic Coordinations Models

The Final Slide

Thank you for your attention!