Topics in Structural Operational Semantics

Arnar Birgisson
Reykjavik University
May 2009

Overview

Practical

Abstract

0. Introduction to SOS
1. Using SOS to design semantics of authorization framework
2. Decompositional reasoning for a process calculus with history
3. SOS rule formats for determinism and idempotency

Structural Operational Semantics

- Syntax of languages is specified e.g. with EBNF or similar
- Tells us that

  ```
  print(“Hello”); print(“World!”)
  ```

  is a legal program.
- ...but not what it *does*.

Structural Operational Semantics

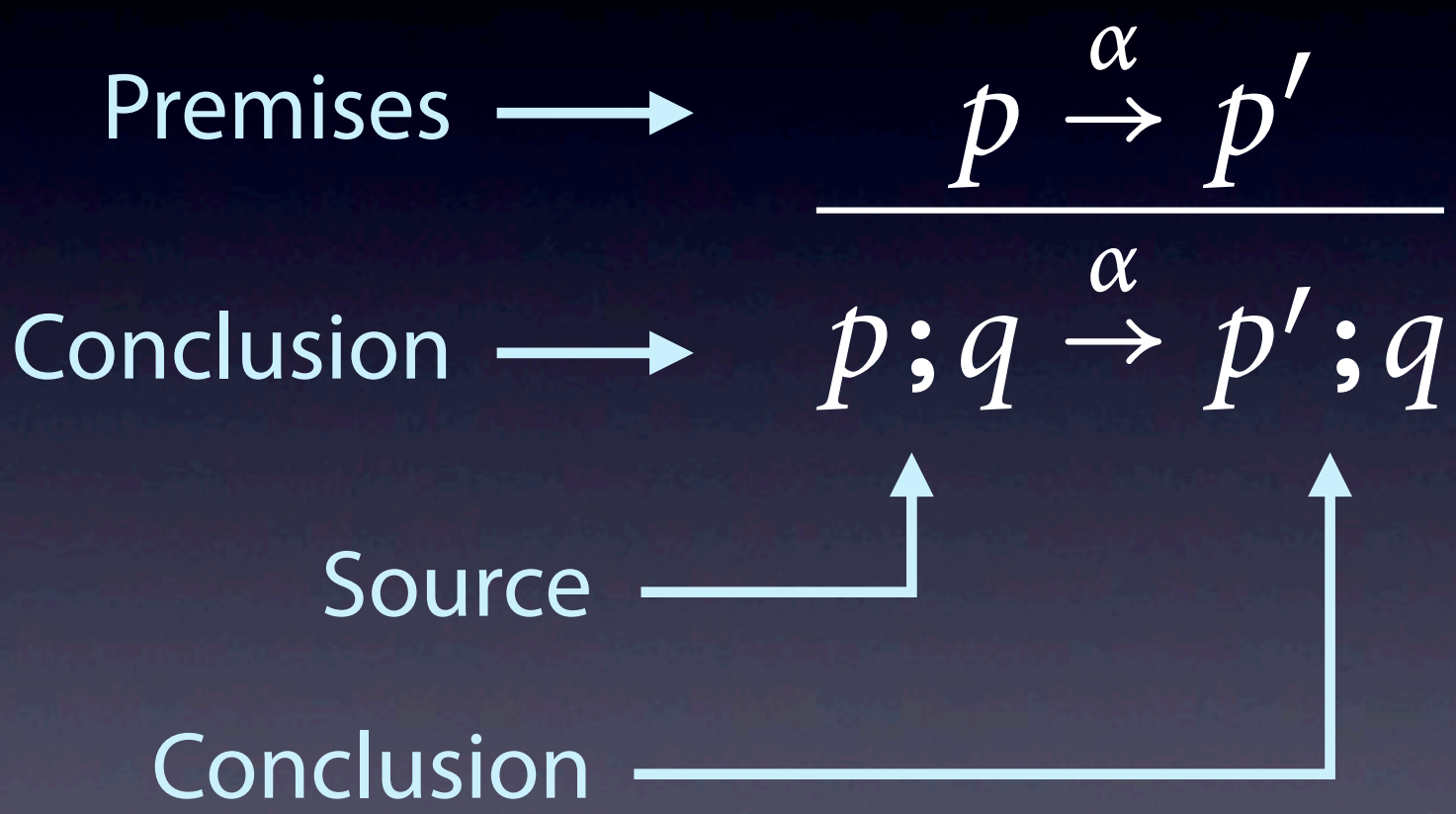
- SOS specifies the semantics of a language with a collection of rules
- Rules specify the operations of programs,

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

- ... in terms of their structure

$$\frac{p \xrightarrow{\alpha} p'}{p;q \xrightarrow{\alpha} p';q}$$

Anatomy of an SOS rule



Example

```
print("Hi"); print("Mom!")
```

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print(“Hi”); print(“Mom!”)
```

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \qquad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```

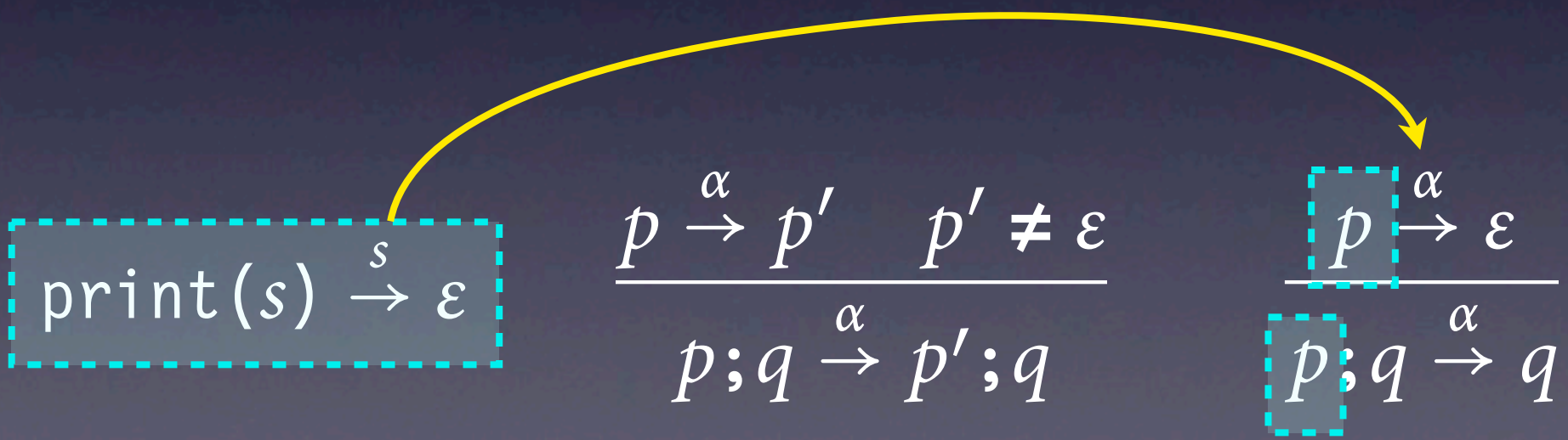
$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

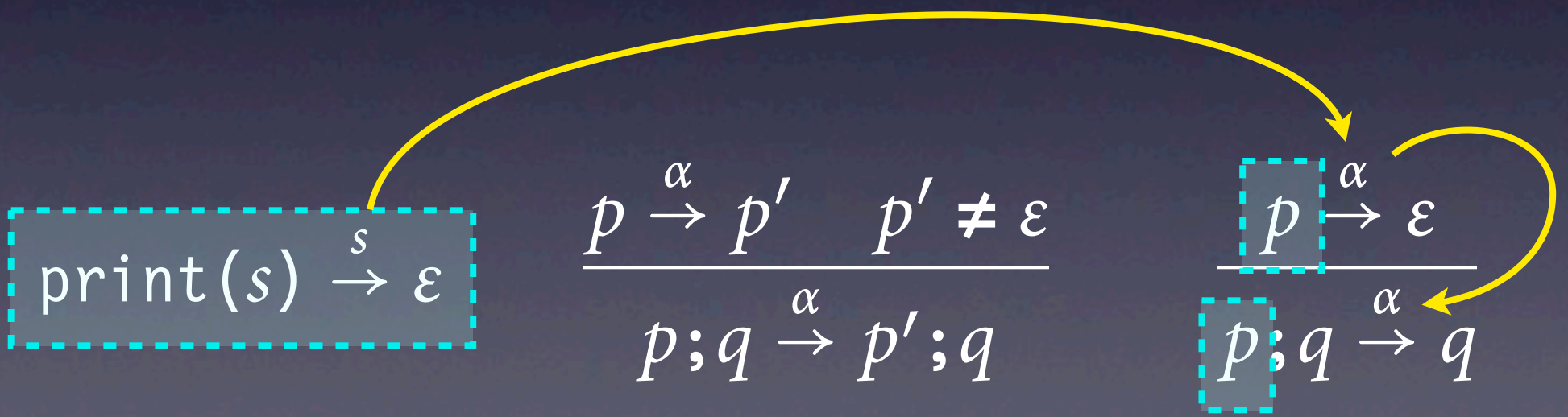
Example

```
print("Hi"); print("Mom!")
```



Example

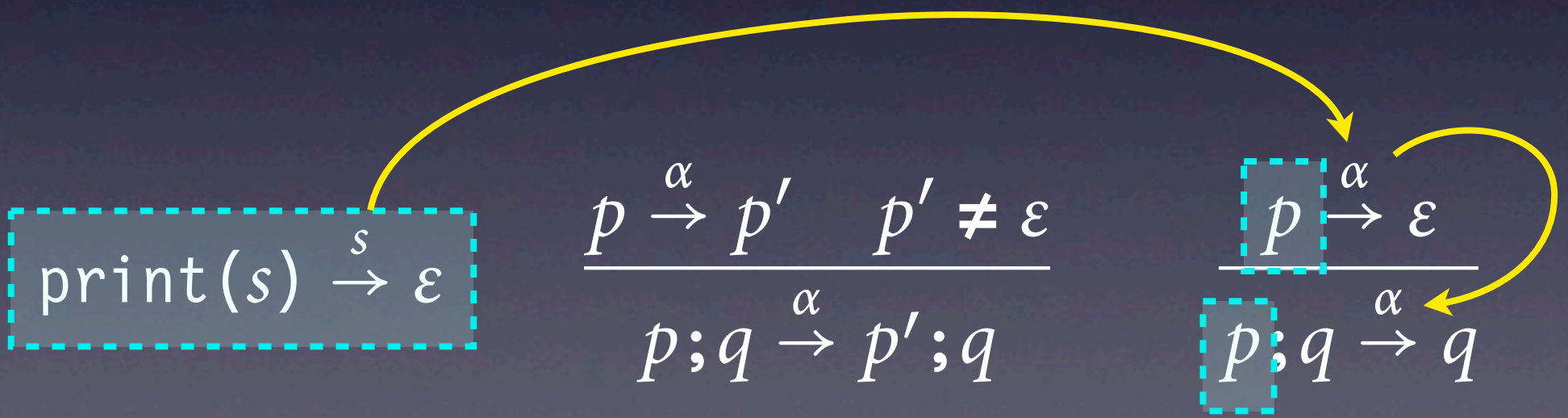
```
print("Hi"); print("Mom!")
```



Example

```
print("Hi"); print("Mom!")
```

$$\xrightarrow{\text{"Hi"}} \texttt{print("Mom!");}$$



Example

```
print("Hi"); print("Mom!")
```

$$\xrightarrow{\text{"Hi"}} \texttt{print("Mom!");}$$

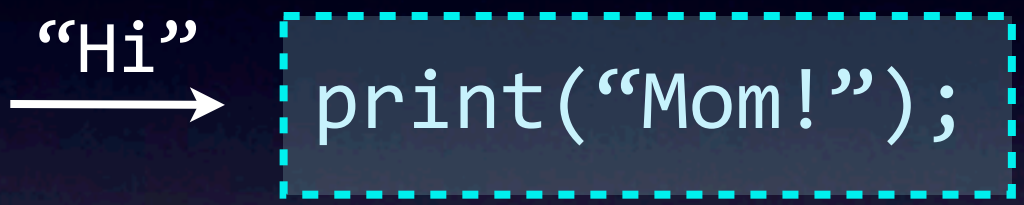
$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```



$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Example

```
print("Hi"); print("Mom!")
```

$$\xrightarrow{\text{"Hi"}} \texttt{print("Mom!");}$$

$$\xrightarrow{\text{"Mom!"}} \varepsilon$$

$$\texttt{print}(s) \xrightarrow{s} \varepsilon$$

$$\frac{p \xrightarrow{\alpha} p' \quad p' \neq \varepsilon}{p;q \xrightarrow{\alpha} p';q}$$

$$\frac{p \xrightarrow{\alpha} \varepsilon}{p;q \xrightarrow{\alpha} q}$$

Uses of SOS

- Rigourous specification of semantics (Part I)
- Reasoning about program behaviour (Part II)
- Proving properties of languages (Part III)
- Proving two programs are equivalent, *e.g.*
  `parallel(f,g)` *behaves like* `parallel(g,f)`
- .... and more

Part I

Transactional Memory Introspection

Joint work with Úlfar Erlingsson

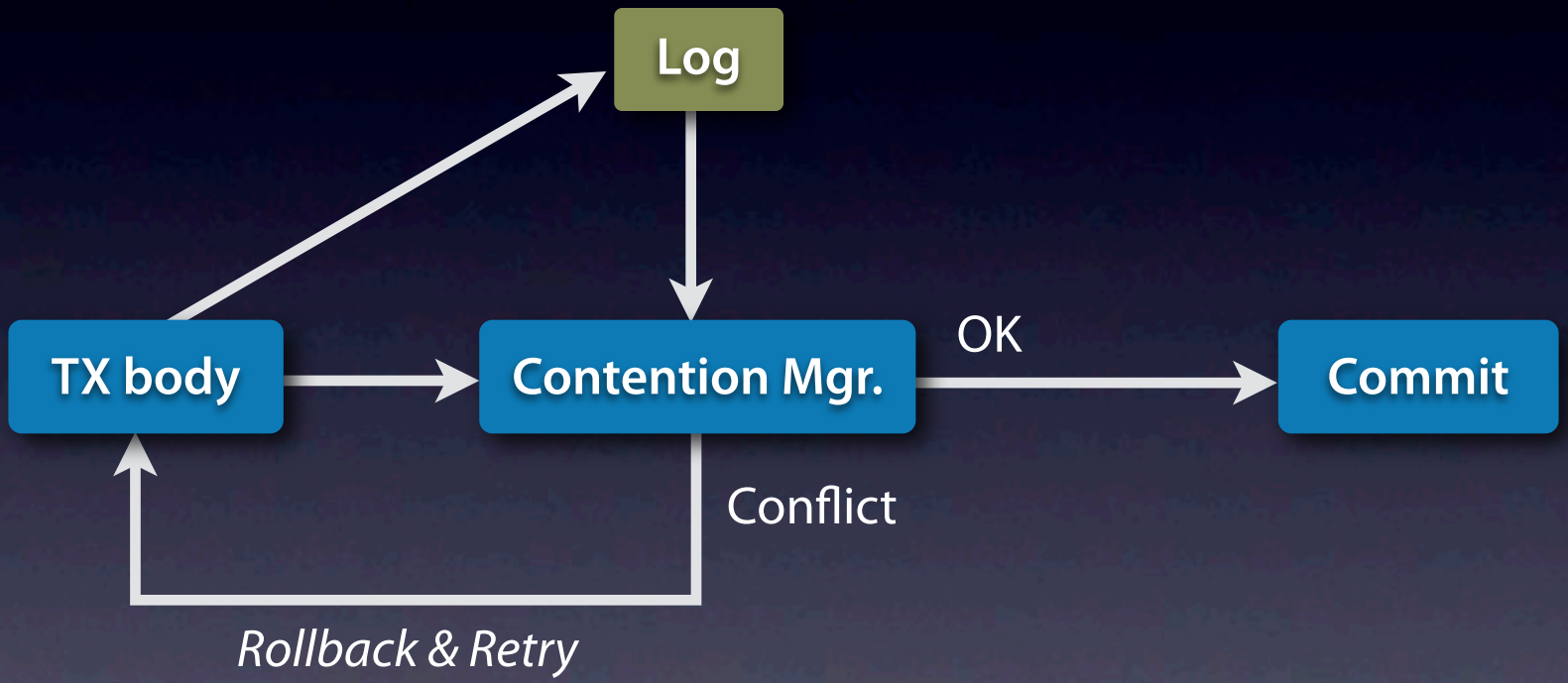
Reykjavik University

(published in PLAS'09 in June '09)

Software Transactional Memory

- STM provides synchronization for threads
- “Atomic” blocks of code are executed in parallel, optimistically but isolated
- The STM system monitors execution to detect conflicting memory accesses
- In case of conflict, an atomic block is rolled back and retried at a later time

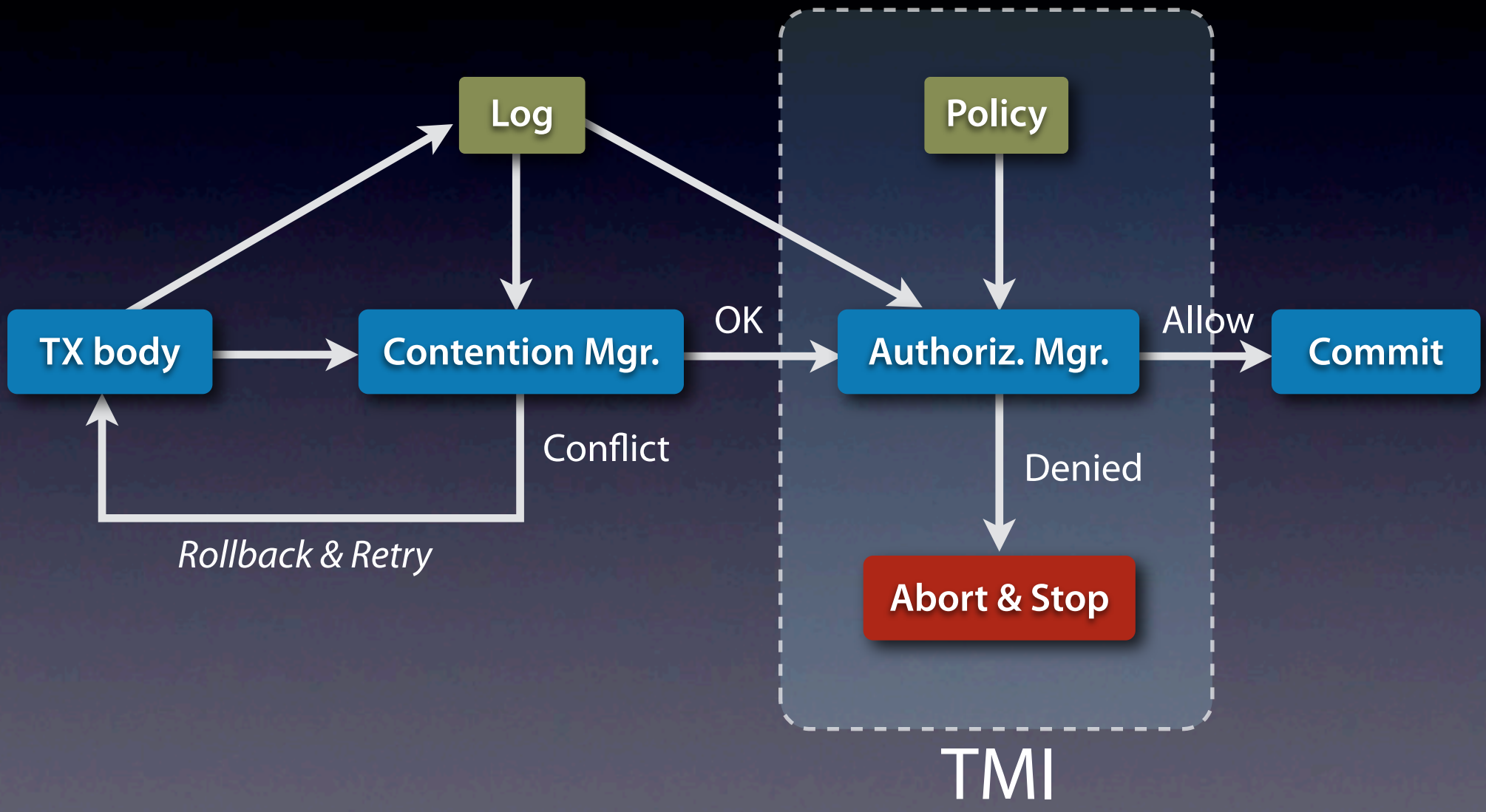
Software Transactional Memory



Transactional Memory Introspection

- A policy enforcement architecture
- Uses the existing mechanisms of STM
- Code, possibly violating policy, lives inside isolated transactions
- STM monitoring used to detect policy violations as well as race conditions
- Transaction containing violation is rolled back, but not retried

Where TMI fits in with STM



Benefits of TMI

- Prevents time of check to time of use bugs
- Ensures complete mediation
- Easier error handling for the programmer
- Decouples policy enforcement code from application code → easier to verify policy enforcement

STM Haskell

```
inc :: TVar Int -> STM ()
inc var = do value <- readTVar var
             writeTVar var (value + 1)

main :: IO ()
main = do x <- newTVarIO 0
          forkIO (atomically (inc x))
          forkIO (atomically (inc x))
          forkIO (atomically (inc x))
```

Value of x is guaranteed to be 3

STM Haskell Semantics

- Main execution context (IO) and STM context handled in separate transition systems
- Creating/reading/writing TVars only possible in the STM transition system
- The "atomically" operator lifts a sequence of STM operations to a single IO operation

$$\frac{M; \Theta \rightarrow^{*}_{STM} \texttt{return}\, N; \Theta'}{\texttt{atomically}\, M; \Theta \rightarrow_{IO} \texttt{return}\, N; \Theta'}$$

TMI Haskell

```
data Label = High | Low

copy :: TMIVar Label t -> TMIVar Label t -> TMI ()
copy x y = do tmp <- readTMIVar x  ;; logged
              writeTMIVar y tmp    ;; logged

policy :: TMILog Label -> Bool
policy log = <(read, High) never precedes
              (write, Low) in log>

main :: IO ()
main = do h <- newTMIVarIO High 10
          l <- newTMIVarIO Low  0
          atomically $ do
             auth policy (copy l h)  ;; OK
             auth policy (copy h l)  ;; fail, TX aborts!
```

TMI semantics

- Yet another context, for TMI protected code
- Creating/reading/writing security sensitive variables only allowed in the TMI context
- “auth” operator lifts TMI operations to STM operations, but clears it with the policy first
- TMI context maintains log of operations, this is the input to the policy decision term

TMI semantics (simplified)

$$\frac{M; \Theta, [] \rightarrow^{*}_{TMI} \texttt{return}\, N; \Theta', \Sigma \qquad \Sigma \vdash P \Rightarrow \mathrm{OK}}{\texttt{auth}\, P\, M; \Theta \rightarrow_{STM} \texttt{return}\, N; \Theta'}$$

$$\frac{M; \Theta, [] \rightarrow^{*}_{TMI} \texttt{return}\, N; \Theta', \Sigma \qquad \Sigma \vdash P \Rightarrow \mathrm{FAIL}}{\texttt{auth}\, P\, M; \Theta \rightarrow_{STM} \texttt{raise AuthError}; \Theta}$$

...not simplified

I/O transitions $P; \Theta \xrightarrow{a} Q; \Theta'$

$$\mathsf{P}[\texttt{putChar}\ c]; \Theta \xrightarrow{!c} \mathsf{P}[\texttt{return}\ ()]; \Theta \quad \textit{(PUTC)}$$
$$\mathsf{P}[\texttt{getChar}\ ]; \Theta \xrightarrow{?c} \mathsf{P}[\texttt{return}\ c]; \Theta \quad \textit{(GETC)}$$
$$\mathsf{P}[\texttt{forkIO}\ M]; \Theta \to (\mathsf{P}[\texttt{return}\ t] \mid M_t); \Theta \quad t \notin \mathsf{P}, \Theta, M \quad \textit{(FORK)}$$
$$\mathsf{P}[\texttt{catch}\ (\texttt{return}\ M)\ N]; \Theta \to \mathsf{P}[\texttt{return}\ M]; \Theta \quad \textit{(CATCH1)}$$
$$\mathsf{P}[\texttt{catch}\ (\texttt{throw}\ M)\ N]; \Theta \to \mathsf{P}[N\ M]; \Theta \quad \textit{(CATCH2)}$$

$$\frac{M \to N}{\mathsf{P}[M]; \Theta \to \mathsf{P}[N]; \Theta} \quad \textit{(ADMIN)}$$

$$\frac{M; \Theta, \{\} \overset{*}{\Rightarrow} \texttt{return}\ N; \Theta', \Delta'}{\mathsf{P}[\texttt{atomic}\ M]; \Theta \to \mathsf{P}[\texttt{return}\ N]; \Theta'} \quad \textit{(ARET1)} \qquad \frac{M; \Theta, \{\} \overset{*}{\Rightarrow} \texttt{throw}\ N; \Theta', \Delta'}{\mathsf{P}[\texttt{atomic}\ M]; \Theta \to \mathsf{P}[\texttt{throw}\ N]; \Theta \cup \Delta'} \quad \textit{(ATHROW1)}$$

Administrative transitions $M \to N$

$$M \to V \quad \text{if } \mathcal{V}[\![M]\!] = V \text{ and } M \notin V \quad \textit{(EVAL)}$$
$$\texttt{return}\ N \gg\!= M \to M\ N \quad \textit{(BIND)}$$
$$\texttt{throw}\ N \gg\!= M \to \texttt{throw}\ N \quad \textit{(THROW)}$$
$$\texttt{retry} \gg\!= M \to \texttt{retry} \quad \textit{(RETRY)}$$

STM transitions $M; \Theta, \Delta, \Sigma \Rightarrow N; \Theta', \Delta', \Sigma'$

$$\mathsf{S}[\texttt{readTVar}\ r]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ \Theta_1(r)]; \Theta, \Delta, \Sigma \quad \text{if } r \in \mathrm{dom}(\Theta) \text{ and } \Theta_2(s) = \bot \quad \textit{(READ)}$$
$$\mathsf{S}[\texttt{writeTVar}\ r\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ ()]; \Theta[r \mapsto (M, \bot)], \Delta, \Sigma \quad \text{if } r \in \mathrm{dom}(\Theta) \text{ and } \Theta_2(s) = \bot \quad \textit{(WRITE)}$$
$$\mathsf{S}[\texttt{newTVar}\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ r]; \Theta[r \mapsto (M, \bot)], \Delta[r \mapsto (M, \bot)], \Sigma \quad r \notin \mathrm{dom}(\Theta) \quad \textit{(NEW)}$$

$$\frac{M \to N}{\mathsf{S}[M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[N]; \Theta, \Delta, \Sigma} \quad \textit{(AADMIN)}$$

$$\frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\Rightarrow} \texttt{return}\ N; \Theta', \Delta', \Sigma'}{\mathsf{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ N]; \Theta', \Delta', \Sigma'} \quad \textit{(OR1)} \qquad \frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\Rightarrow} \texttt{throw}\ N; \Theta', \Delta', \Sigma'}{\mathsf{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{throw}\ N]; \Theta', \Delta', \Sigma'} \quad \textit{(OR2)}$$

$$\frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\Rightarrow} \texttt{retry}; \Theta', \Delta', \Sigma'}{\mathsf{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[M_2]; \Theta, \Delta, \Sigma} \quad \textit{(OR3)} \qquad \frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{return}\ M'; \Theta', \Delta', \Sigma'}{\mathsf{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ M']; \Theta', \Delta \cup \Delta', \Sigma \oplus \Sigma'} \quad \textit{(XSTM1)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{throw}\ M'; \Theta', \Delta', \Sigma'}{\mathsf{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[N\ M']; \Theta \cup \Delta', \Delta \cup \Delta', \Sigma \oplus (\Sigma'|_{\Delta'})} \quad \textit{(XSTM2)} \qquad \frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{retry}; \Theta', \Delta', \Sigma'}{\mathsf{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{retry}]; \Theta, \Delta, \Sigma} \quad \textit{(XSTM3)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{return}\ M'; \Theta', \Delta', \Sigma' \qquad \Sigma' \vdash N \overset{*}{\rightsquigarrow} \texttt{return}\ N'}{\mathsf{S}[\texttt{authorized}\ N\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{return}\ M']; \Theta', \Delta', \Sigma} \quad \textit{(AURET1)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{throw}\ M'; \Theta', \Delta', \Sigma' \qquad \Sigma' \vdash N \overset{*}{\rightsquigarrow} \texttt{return}\ N'}{\mathsf{S}[\texttt{authorized}\ N\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{throw}\ M']; \Theta', \Delta', \Sigma} \quad \textit{(AUTHROW1)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{return}\ M'; \Theta', \Delta', \Sigma' \qquad \Sigma' \vdash N \overset{*}{\rightsquigarrow} \texttt{throw}\ N'}{\mathsf{S}[\texttt{authorized}\ N\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{throw UnathorizedError}]; \Theta', \Delta', \Sigma} \quad \textit{(AURET2)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{throw}\ M'; \Theta', \Delta', \Sigma' \qquad \Sigma' \vdash N \overset{*}{\rightsquigarrow} \texttt{throw}\ N'}{\mathsf{S}[\texttt{authorized}\ N\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{throw UnauthorizedError}]; \Theta', \Delta', \Sigma} \quad \textit{(AUTHROW1)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\Rightarrow} \texttt{retry}; \Theta', \Delta', \Sigma'}{\mathsf{S}[\texttt{authorized}\ N\ M]; \Theta, \Delta, \Sigma \Rightarrow \mathsf{S}[\texttt{retry}]; \Theta', \Delta' \Sigma} \quad \textit{(AURETRY)}$$

Figure 2: Operational semantics (part 1)

TMI transitions $M; \Theta, \Delta, \Sigma \rightharpoonup N; \Theta', \Delta', \Sigma'$

$$\mathbb{S}[\texttt{readTMIVar}\ r]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{return}\ \Theta_1(r)]; \Theta, \Delta, \Sigma \oplus [(\textsc{read}, \Theta_2(r))] \quad \text{if } r \in \mathrm{dom}(\Theta) \text{ and } \Theta_2(r) \neq \bot \quad \textit{(TMIREAD)}$$
$$\mathbb{S}[\texttt{writeTMIVar}\ r\ M]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{return}\ ()]; \Theta[r \mapsto (M, \Theta_2(r))], \Delta, \Sigma \oplus [(\textsc{write}, \Theta_2(r))] \quad \text{if } r \in \mathrm{dom}(\Theta) \text{ and } \Theta_2(r) \neq \bot \quad \textit{(TMIWRITE)}$$
$$\mathbb{S}[\texttt{newTMIVar}\ N\ M]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{return}\ r]; \Theta[r \mapsto (M, N)], \Delta[r \mapsto (M, N)], \Sigma \oplus [(\textsc{create}, N)] \quad r \notin \mathrm{dom}(\Theta) \quad \textit{(TMINEW)}$$

$$\frac{M \to N}{\mathbb{S}[M]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[N]; \Theta, \Delta, \Sigma} \quad \textit{(TADMIN)} \qquad \frac{M; \Theta, \Delta, \Sigma \overset{*}{\Rightarrow} N; \Theta', \Delta', \Sigma'}{\mathbb{S}[\texttt{liftSTM}\ M]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[N]; \Theta', \Delta', \Sigma'} \quad \textit{(LIFTSTM)}$$

$$\frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\rightharpoonup} \texttt{return}\ N; \Theta', \Delta', \Sigma'}{\mathbb{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{return}\ N]; \Theta', \Delta', \Sigma'} \quad \textit{(TOR1)} \qquad \frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\rightharpoonup} \texttt{throw}\ N; \Theta', \Delta', \Sigma'}{\mathbb{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{throw}\ N]; \Theta', \Delta', \Sigma'} \quad \textit{(TOR2)}$$

$$\frac{M_1; \Theta, \Delta, \Sigma \overset{*}{\rightharpoonup} \texttt{retry}; \Theta', \Delta', \Sigma'}{\mathbb{S}[M_1\ \texttt{`orElse`}\ M_2]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[M_2]; \Theta, \Delta, \Sigma} \quad \textit{(TOR3)} \qquad \frac{M; \Theta, \{\}, [] \overset{*}{\rightharpoonup} \texttt{return}\ M'; \Theta', \Delta', \Sigma'}{\mathbb{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{return}\ M']; \Theta', \Delta \cup \Delta', \Sigma \oplus \Sigma'} \quad \textit{(XTMI1)}$$

$$\frac{M; \Theta, \{\}, [] \overset{*}{\rightharpoonup} \texttt{throw}\ M'; \Theta', \Delta', \Sigma'}{\mathbb{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[N\ M']; \Theta \cup \Delta', \Delta \cup \Delta', \Sigma \oplus (\Sigma'|_{\Delta'})} \quad \textit{(XTMI2)} \qquad \frac{M; \Theta, \{\}, [] \overset{*}{\rightharpoonup} \texttt{retry}; \Theta', \Delta', \Sigma'}{\mathbb{S}[\texttt{catch}\ M\ N]; \Theta, \Delta, \Sigma \rightharpoonup \mathbb{S}[\texttt{retry}]; \Theta, \Delta, \Sigma} \quad \textit{(XTMI3)}$$

Authorization transitions $\Sigma \vdash M \rightsquigarrow N$

$$\frac{M \to N}{\Sigma \vdash \mathbb{E}[M] \rightsquigarrow \mathbb{E}[N]} \quad \textit{(AUADMIN)} \qquad \Sigma \vdash \mathbb{E}[\texttt{getlog}] \rightsquigarrow \mathbb{E}[\texttt{return}\ hs(\Sigma)] \quad \textit{(GETLOG)}$$

$$\Sigma \vdash \mathbb{E}[\texttt{catch}\ (\texttt{return}\ M)\ N] \rightsquigarrow \mathbb{E}[\texttt{return}\ M] \quad \textit{(ACATCH1)}$$
$$\Sigma \vdash \mathbb{E}[\texttt{catch}\ (\texttt{throw}\ M)\ N] \rightsquigarrow \mathbb{E}[N\ M] \quad \textit{(ACATCH2)}$$

Figure 3: Operational semantics (part 2)

Gain of using SOS

- We had already defined TMI, implemented it in Java and published the results
- Despite this, it took several attempts to design acceptable semantics with SOS
- Revealed ill-understood edge cases
- Basis for more formal future work

Part II

Decompositional reasoning about the past

Joint, ongoing work with Luca Aceto,
Anna Ingólfsdóttir and MohammadReza Mousavi

Reykjavik University &
TU/e - Eindhoven Technical University

Decompositional reasoning

- Model checking has to deal with state space explosion, esp. for parallel processes
- Transforming global properties into local properties can potentially help
- Check local properties on the components instead
- Dates back to early ‘90s (Larsen and Xinxin)

Example

CM = <unknown specification>

CS = $\overline{coin}.coffe.\overline{pub}.CS$

Uni = (CM | CS) \ coin, coffee

We want to verify that Uni satisfies $\langle\overline{pub}\rangle tt$

By looking at CS, we see this holds if and only if

CM satisfies $\langle\overline{pub}\rangle tt \vee \langle coin\rangle\langle\overline{coffe}\rangle tt$

Example

CM = <unknown specification>

CS = $\overline{coin}.coffe.\overline{pub}.CS$

Uni = (CM | CS) \ coin, coffee

We want to verify that Uni satisfies $\langle\overline{pub}\rangle tt$

By looking at CS, we see this holds if and only if

CM satisfies $\langle\overline{pub}\rangle tt \vee \langle coin\rangle\langle\overline{coffe}\rangle tt$

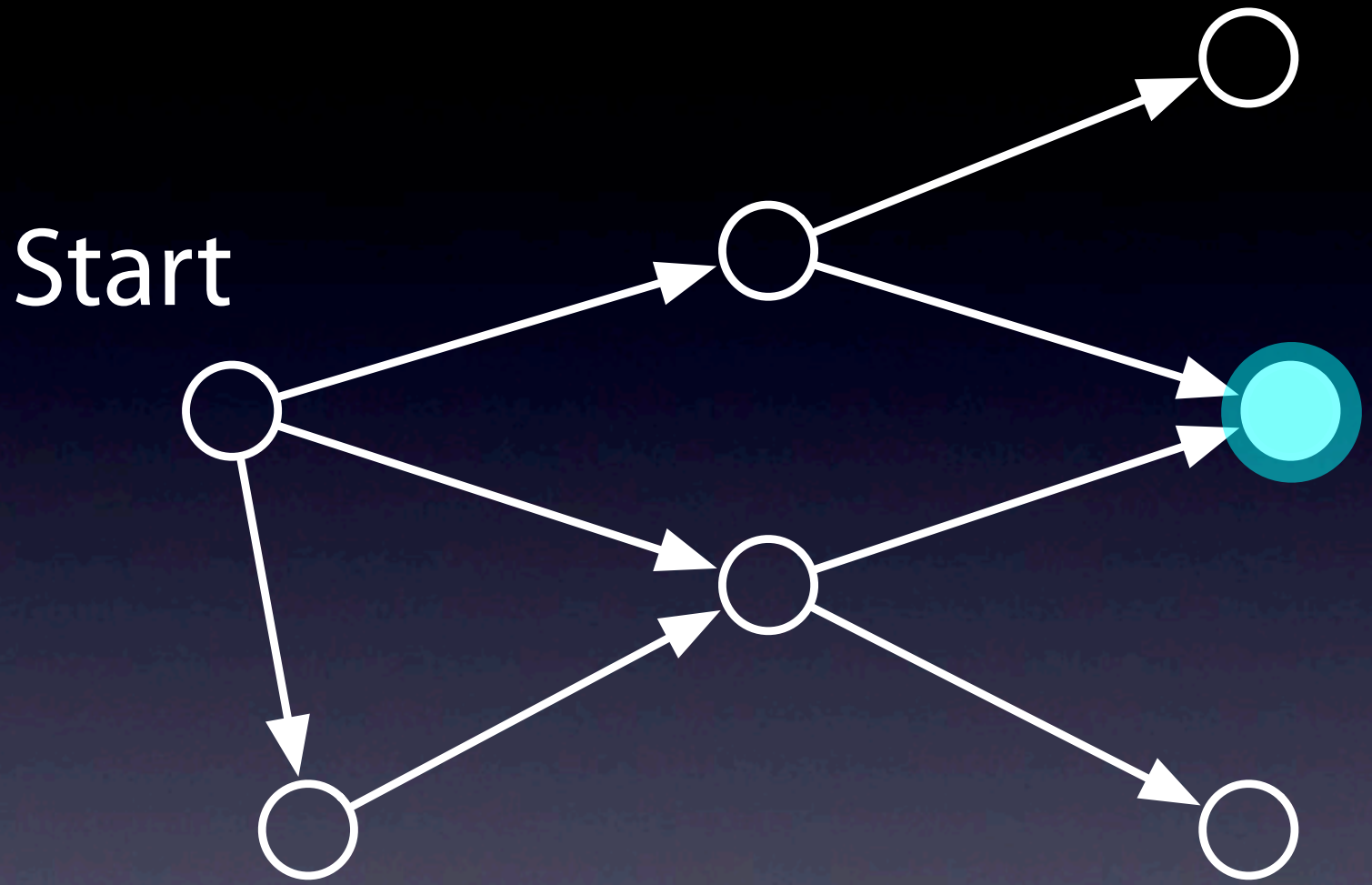
$\langle\overline{pub}\rangle tt$ / CS

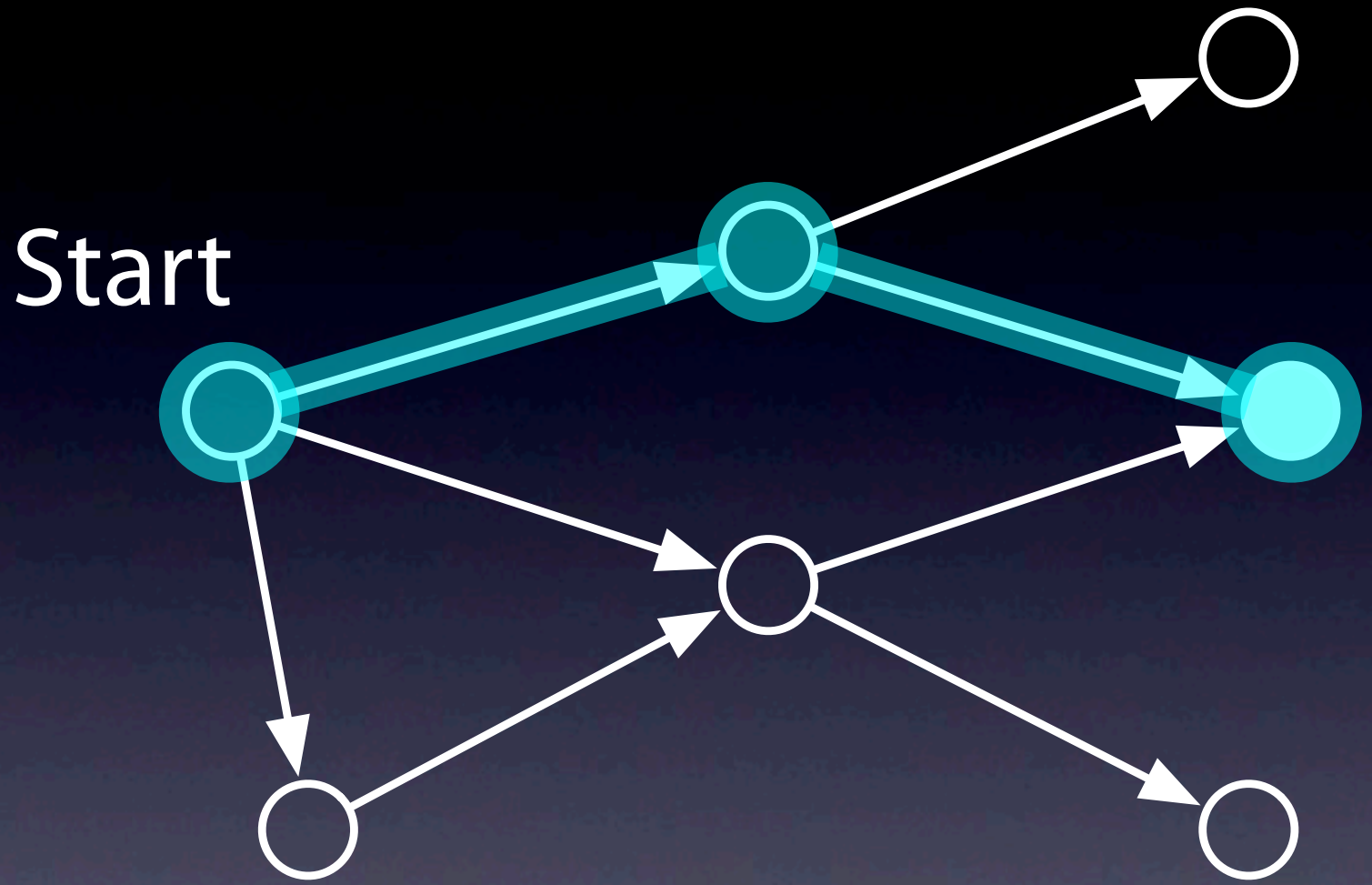
Process algebra with history

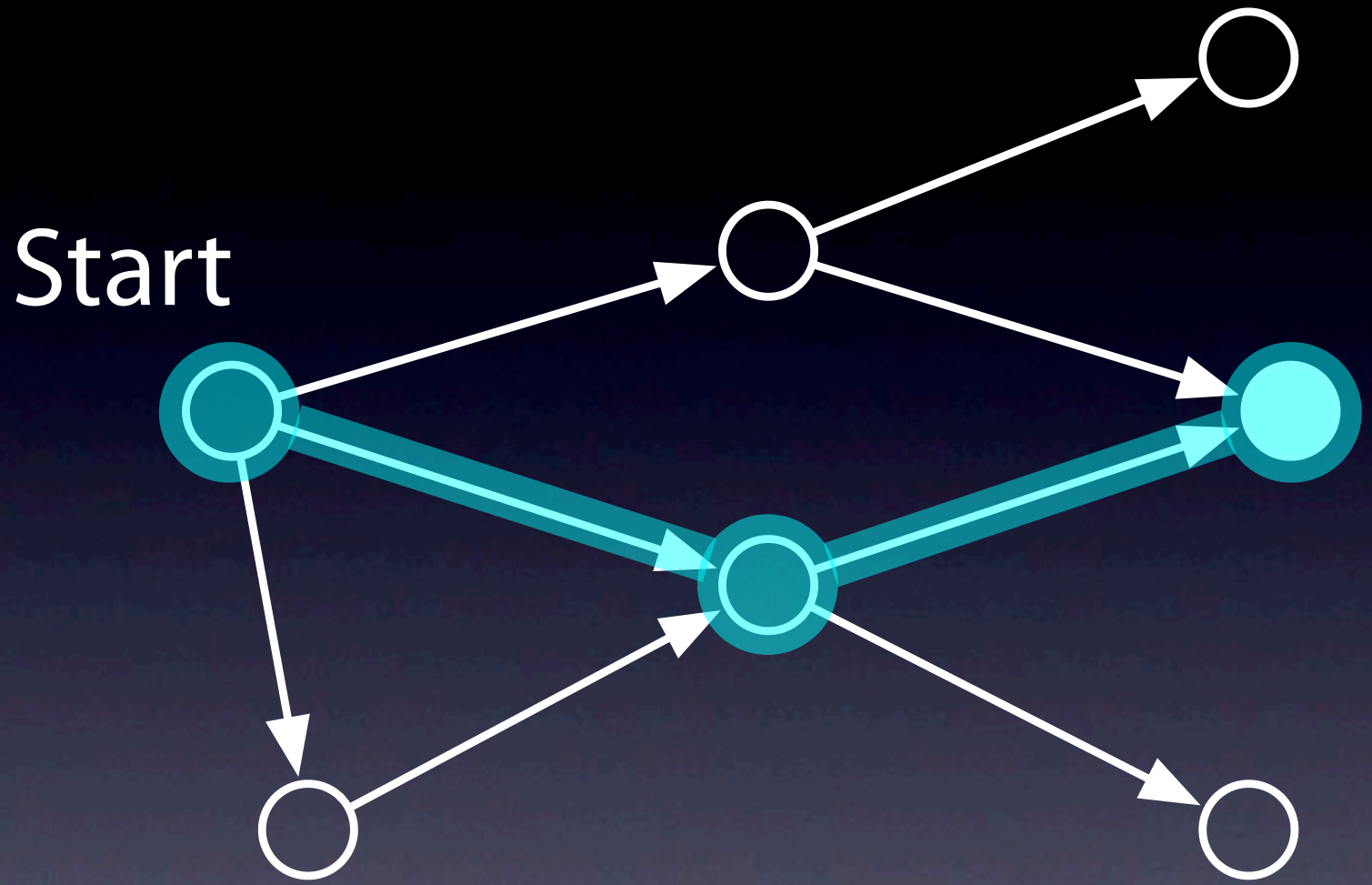
- Instead of states being processes, states are computations
- Computations contain the past history of a process

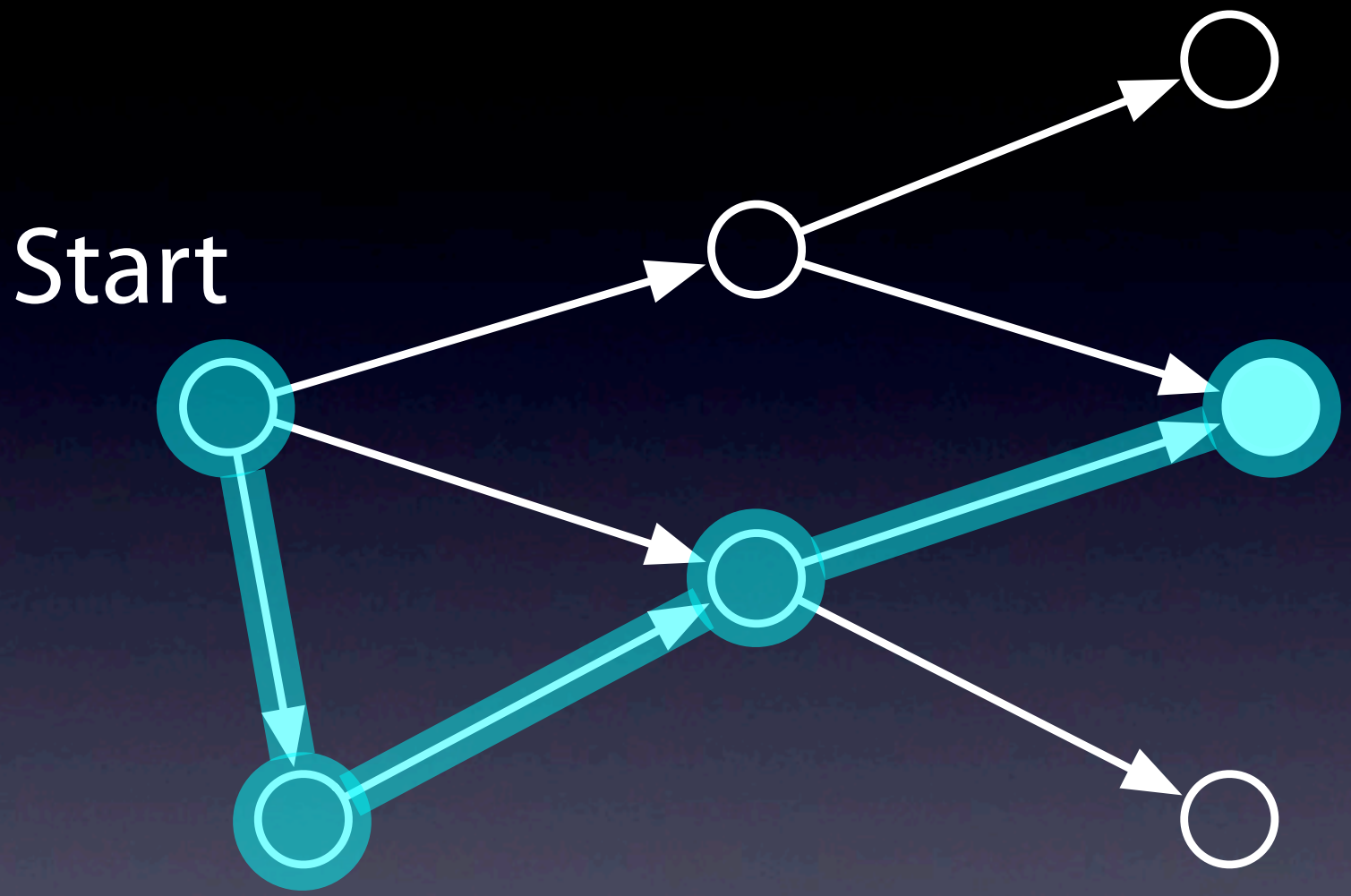
$$(a.b.0 \xrightarrow{a} b.0) \xrightarrow{b} (a.b.0 \xrightarrow{a} b.0 \xrightarrow{b} 0)$$

- Applications in verification of security protocols, epistemic properties


Start


Start


Start

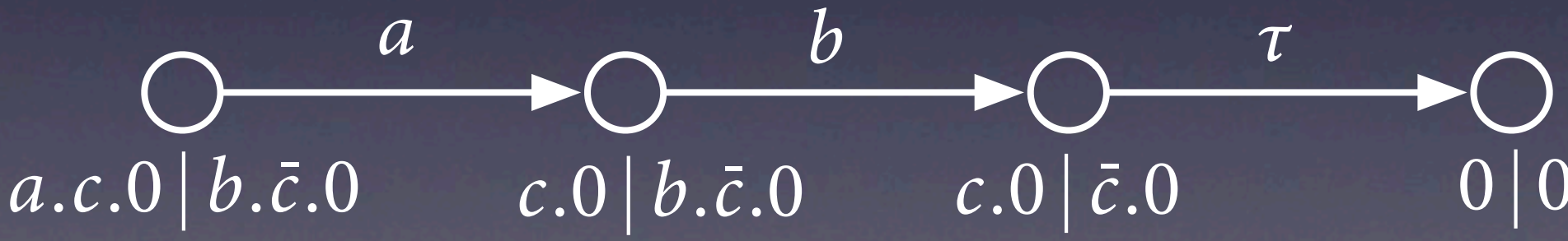

Start

HML with past

- Truth, conjunction, negation, diamond and backwards diamond
- $C \models \langle \leftarrow a \rangle \varphi$ means that the last step of the computation C was labelled with a and the computation with that last step removed satisfies φ

Decomposing a parallel computation

- Each step in a composed computation is either
  - a step of one of the components, or
  - a communication between two components



Decomposing a parallel computation

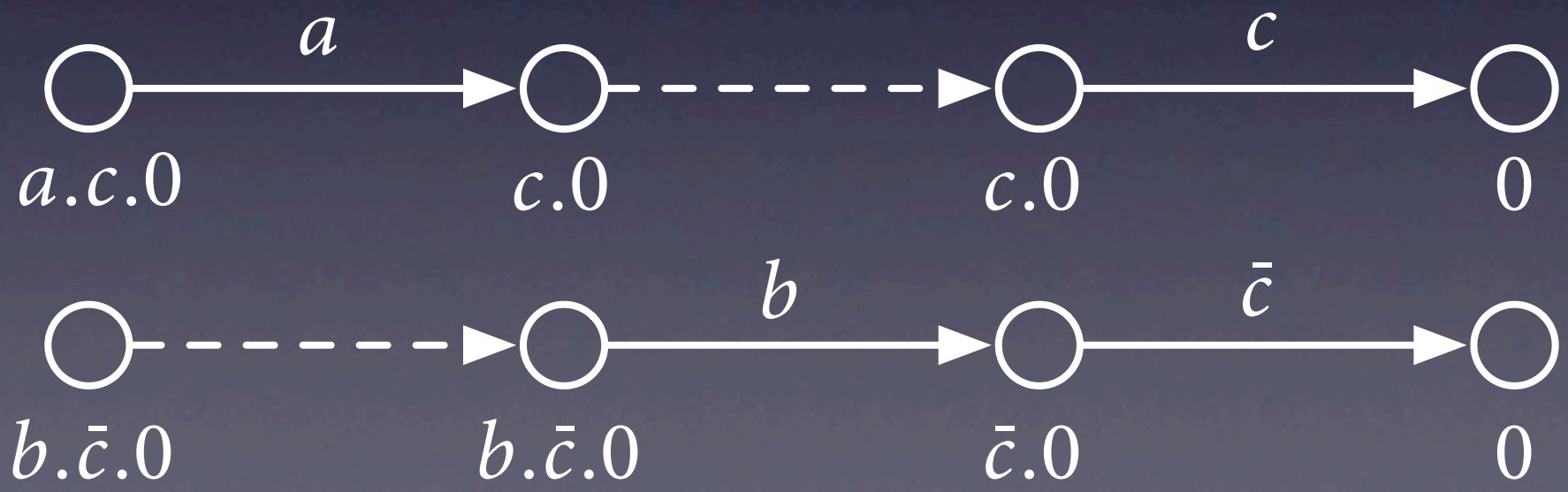
- Each step in a composed computation is either
  - a step of one of the components, or
  - a communication between two components

Decomposing a parallel computation

- Each step in a composed computation is either
  - a step of one of the components, or
  - a communication between two components

Decomposing a parallel computation

- Each step in a composed computation is either
  - a step of one of the components, or
  - a communication between two components



Decomposition

- Decomposition is not unique, e.g.

$$(a.0 + b.0) \mid (\bar{a}.0 + \bar{b}.0) \xrightarrow{\tau} 0 \mid 0$$

- For a computation C of two parallel components, $D(C)$ is the set of decompositions of C, where each decomposition is a pair of computations.

Decomposition theorem

- We define a transformation of HML+past formula φ w.r.t. a computation C, written φ/C

Theorem: For a computation C and formula φ

$$C \vDash \varphi \quad \Rightarrow \quad \forall (C', C'') \in D(C) : C' \vDash \varphi/C$$

$$C \vDash \varphi \quad \Leftarrow \quad \exists (C', C'') \in D(C) : C' \vDash \varphi/C$$

Future (and current) work

- Extend the theorem to handle fixed points
- Generalize with rule formats (currently based on CCS specific syntax and semantics)
- Extend to include epistemic operators in the logic, i.e. knowledge, common knowledge

Part III

SOS rule formats for determinism and idempotency

Joint work with Luca Aceto, Anna Ingólfsdóttir,
MohammadReza Mousavi and Michel Reniers

Reykjavik University &
TU/e - Eindhoven Technical University

(published in FSEN'09 in April '09)

Determinism

- In a transition system, a transition relation $\xrightarrow{l}$ is said to be deterministic if

$$p \xrightarrow{l} p' \text{ and } p \xrightarrow{l} p'' \quad \Rightarrow \quad p' \equiv p''$$

- Holds for sublanguages of many process calculi and programming languages
- Important, e.g. in timed systems where passage of time must not resolve any choice.

Idempotency

- An operator binary f is idempotent w.r.t. $\sim$, if for all terms x it holds that

$$f(x, x) \sim x$$

  where $\sim$ is some suitable equivalence.

- Example: The non-deterministic choice operator + (e.g. in CCS) is idempotent w.r.t. to bisimulation:

$$\forall p : p + p \approx p$$

SOS rule formats

- Semantic properties such as determinism and idempotency had to be proven for each transition system specification
- Rule formats are meta-theorems, properties can be inferred from the syntactic structure of rules - no need for complex proofs
- Rule formats exist for many properties

Source dependency

- A variable in a rule is *source dependent* via a set of labels L if

  1. It appears in the source of the conclusion, or
  2. It appears in the target of a premise, whose source variables are all source dependent via L

- Example:

$$\frac{x \xrightarrow{a} z \quad z \xrightarrow{b} y}{f(x) \xrightarrow{c} y}$$

y is source dependent via $\{a, b\}$

Determinism format

- A transition system specification T is in the *determinism format* w.r.t. a set of labels L if for each l in L

  1. In each rule defining an l-transition, all variables in the conclusion are source dependent via (a subset of) L, and
  2. In each two, distinct rules defining l-transitions with the *same source*, either the conclusion is also the same or their predicates contradict

Determinism theorem

If a TSS is in determinism format with respect to a set of labels L,
then for all of the transition relations with labels in L are deterministic.

Example

$$\frac{}{\epsilon \downarrow} \qquad \frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \downarrow}{x \mp y \downarrow} \qquad \frac{y \downarrow}{x \mp y \downarrow}$$

$$\frac{x \xrightarrow{a} x' \qquad y \xrightarrow{a} y'}{x \mp y \xrightarrow{a} x' \mp y'} \qquad \frac{x \xrightarrow{a} x' \qquad y \overset{a}{\nrightarrow}}{x \mp y \xrightarrow{a} x'} \qquad \frac{x \overset{a}{\nrightarrow} \qquad y \xrightarrow{a} y'}{x \mp y \xrightarrow{a} y'}$$

Semantic vs. syntactic

- The reason for quoting *"same"*: there are subtleties in abstracting from variable names
- The general format quantifies over substitutions, i.e. not syntactic and hard to check
- Restricting rules to be *normalized* gives a completely syntactic, but less general format

Normalized rules

- Non-normalized, not covered by syntactic format, but deterministic:

$$\frac{x \xrightarrow{a} x'}{f(x,y) \xrightarrow{a} x'} \qquad \frac{y \overset{a}{\not\to} \quad x \xrightarrow{b} x'}{f(y,x) \xrightarrow{a} x'}$$

- Normalized, covered by syntactic format:

$$\frac{x \xrightarrow{a} x'}{f(x,y) \xrightarrow{a} x'} \qquad \frac{x \overset{a}{\not\to} \quad y \xrightarrow{b} y'}{f(x,y) \xrightarrow{a} y'}$$

Idempotency format

- The idempotency format describes certain types of rules which are allowed
- Relies on the determinism format
- Example: ATP time-deterministic choice, idempotency of $\oplus$ depends on χ being deterministic

$$\frac{x_0 \xrightarrow{a} x_0'}{x_0 \oplus x_1 \xrightarrow{a} x_0'} \qquad \frac{x_1 \xrightarrow{a} x_1'}{x_0 \oplus x_1 \xrightarrow{a} x_1'} \qquad \frac{x_0 \xrightarrow{\chi} x_0' \quad x_1 \xrightarrow{\chi} x_1'}{x_0 \oplus x_1 \xrightarrow{\chi} x_0' \oplus x_1'}$$

Meta

- Computer science is about abstractions
- Formal methods are the way to reason rigorously about those abstractions
- Sound abstractions are key to building reliable, correct systems - which are becoming ever more important
- Theory is nothing without practical applications and vice versa

Thanks!