Model Checking I
alias
Reactive Systems Verification

Luca Tesei

MSc in Computer Science, University of Camerino

Topics

- Synchronous Product
- Examples
- The state explosion problem

Material

Reading:

Chapter 2 of the book, pages 75–80.

More:

The slides in the following pages are taken from the material of the course “Introduction to Model Checking” held by Prof. Dr. Ir. Joost-Pieter Katoen at Aachen University.

Summary: parallel operators

PC2.2-39

(pure) interleaving for TS $\mathcal{T}_1 \;|||\; \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

(pure) interleaving for TS $\mathcal{T}_1 \; ||| \; \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \, \|_{Syn} \, \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in Syn

Summary: parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \ ||| \ \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \, \|_{Syn} \, \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in Syn

interleaving for program graphs $\mathcal{P}_1 \ ||| \ \mathcal{P}_2$

- interleaving for concurrent actions
- communication via shared variables

Summary: parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \; ||| \; \mathcal{T}_2$

- only concurrency, no communication
- not applicable for competing systems

synchronous message passing for TS $\mathcal{T}_1 \, \|_{Syn} \, \mathcal{T}_2$

- interleaving for concurrent actions
- synchronization via actions in *Syn*

interleaving for program graphs $\mathcal{P}_1 \; ||| \; \mathcal{P}_2$

- interleaving for concurrent actions
- communication via shared variables

channel systems: open $\mathcal{P}_1 | \ldots | \mathcal{P}_n$ or closed $[\mathcal{P}_1 | \ldots | \mathcal{P}_n]$

- interleaving, shared variables, message passing

Parallel operators

(pure) interleaving for TS $\mathcal{T}_1 \,|||\, \mathcal{T}_2$

- only concurrency, no communication

synchronous message passing for TS $\mathcal{T}_1 \|_{Syn} \mathcal{T}_2$

- interleaving, synchronization via actions in Syn

interleaving for program graphs $\mathcal{P}_1 \,|||\, \mathcal{P}_2$

- interleaving, shared variables

channel systems: open $\mathcal{P}_1 | \ldots | \mathcal{P}_n$ or closed $[\mathcal{P}_1 | \ldots | \mathcal{P}_n]$

- interleaving, shared variables
- synchronous and asynchronous message passing

synchronous product for TS $\mathcal{T}_1 \otimes \mathcal{T}_2$

- no interleaving, "pure" synchronization

Synchronous product

for parallel systems with fully synchronized processes

$$\left.\begin{array}{l} \mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \ldots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \ldots) \end{array}\right\} \text{ two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \; = \; (S_1 \times S_2, Act, \longrightarrow, \ldots)$$

for parallel systems with fully synchronized processes

$$\left.\begin{array}{l}\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \ldots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \ldots)\end{array}\right\} \text{two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \; = \; (S_1 \times S_2, Act, \longrightarrow, \ldots)$$

where the action set Act is given by a function

$$Act_1 \times Act_2 \longrightarrow Act, \quad (\alpha, \beta) \mapsto \alpha * \beta$$

action name for the concurrent execution of α and β

for parallel systems with fully synchronized processes

$$\left.\begin{array}{l} \mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \ldots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \ldots) \end{array}\right\} \text{two TS}$$

synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \;=\; (S_1 \times S_2, Act, \longrightarrow, \ldots)$$

where the action set Act is given by a function

$$Act_1 \times Act_2 \longrightarrow Act, \quad (\alpha, \beta) \mapsto \alpha * \beta$$

action name for the concurrent execution of α and β

if action names are irrelevant: $Act_1 = Act_2 = Act = \{\tau\}$

Synchronous product

for parallel systems with fully synchronized processes

$$\left.\begin{array}{l}\mathcal{T}_1 = (S_1, Act_1, \longrightarrow_1, \ldots) \\ \mathcal{T}_2 = (S_2, Act_2, \longrightarrow_2, \ldots)\end{array}\right\} \text{ two TS}$$

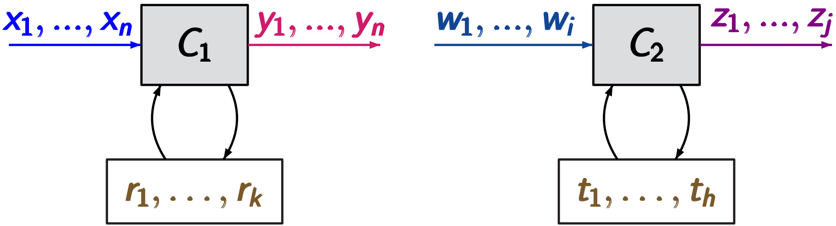
synchronous product:

$$\mathcal{T}_1 \otimes \mathcal{T}_2 \; = \; (S_1 \times S_2, Act, \longrightarrow, \ldots)$$

transition relation $\longrightarrow$:

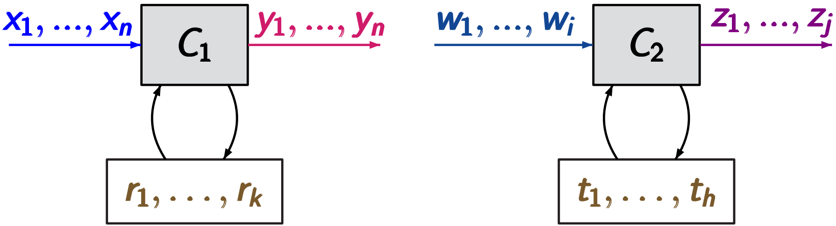
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \;\wedge\; s_2 \xrightarrow{\beta}_2 s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha * \beta} \langle s_1', s_2' \rangle}$$

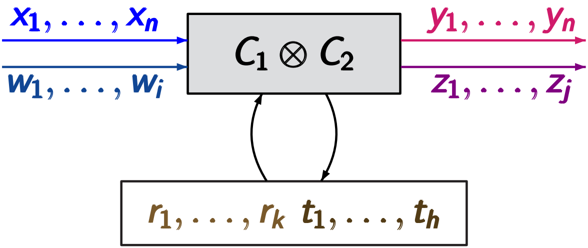
Synchronous product for composing circuits



2 sequential circuits

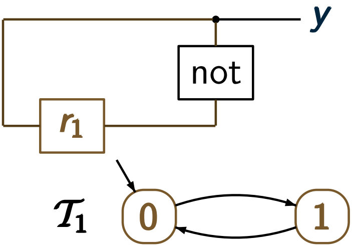
Synchronous product for composing circuits

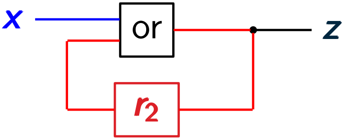




Synchronous product: example

Synchronous product: example

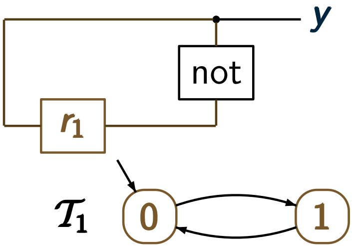


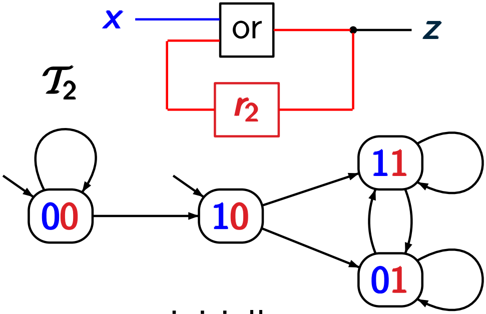


initially:
$r_1 = 0$

transition function:
$\delta_{r_1} = \neg r_1$

Synchronous product: example





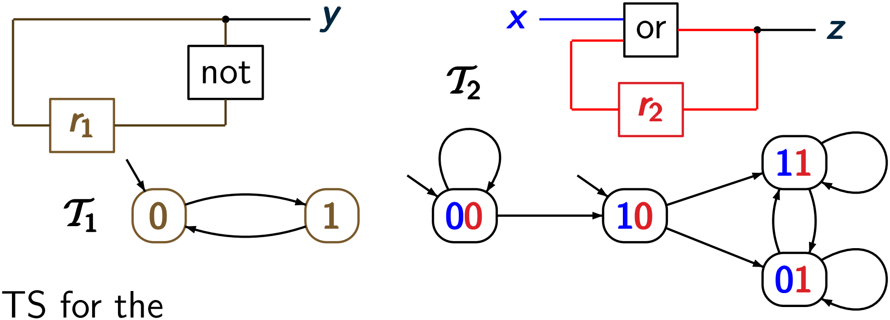
initially:
$r_1 = 0$

initially:
$r_2 = 0$

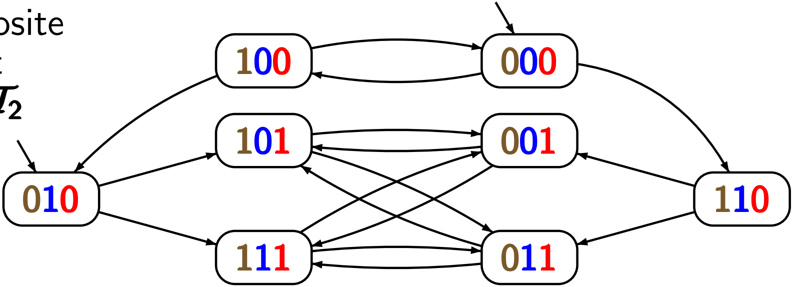
transition function:
$\delta_{r_1} = \neg r_1$

transition function:
$\delta_{r_2} = r_2 \vee x$

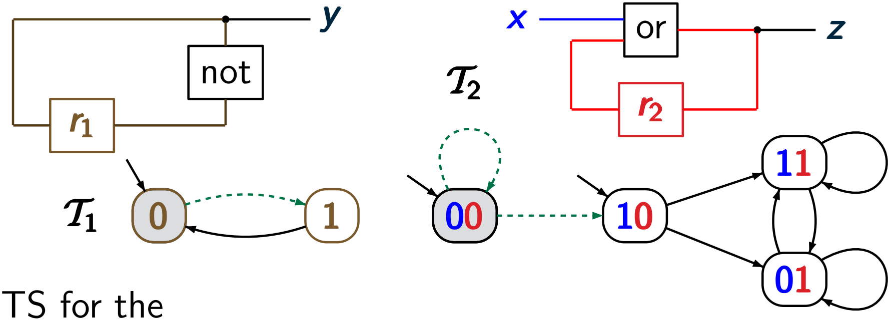
Synchronous product: example



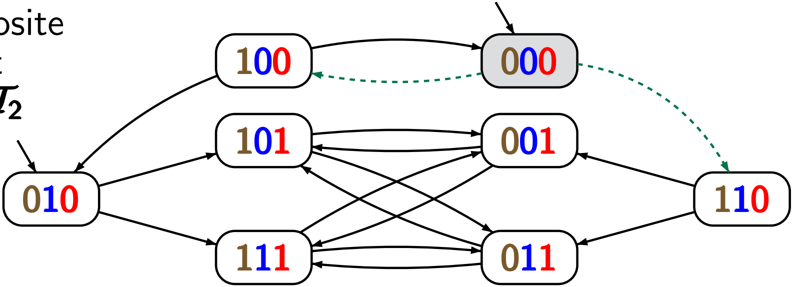
TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$



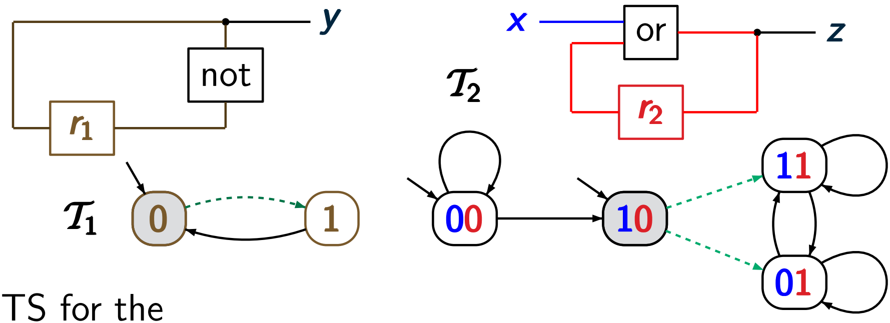
Synchronous product: example



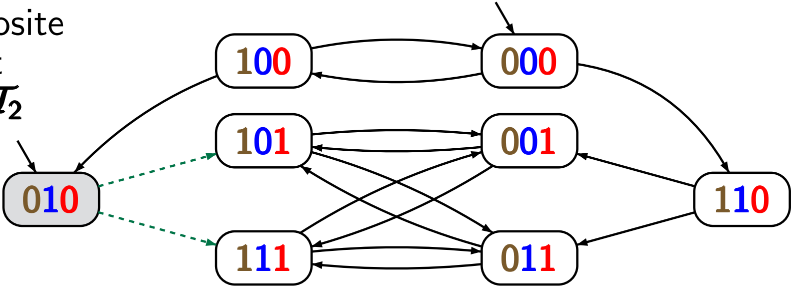
TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$



Synchronous product: example



TS for the composite circuit $\mathcal{T}_1 \otimes \mathcal{T}_2$



State explosion problem

TS for reactive systems can be enormously large

State explosion problem

TS for reactive systems can be enormously large

- infinite for systems with
  - ∗ variables of infinite domains, e.g., $\mathbb{N}$
  - ∗ infinite data structures, e.g., stacks, queues, lists,...

State explosion problem

TS for reactive systems can be enormously large

- infinite for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...
- if finite: exponential growth in

State explosion problem

TS for reactive systems can be enormously large

- infinite for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...
- if finite: exponential growth in
  - number of parallel components, e.g., state space of $\mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$ is $S_1 \times \ldots \times S_n$

State explosion problem

TS for reactive systems can be enormously large

- infinite for systems with
  - variables of infinite domains, e.g., $\mathbb{N}$
  - infinite data structures, e.g., stacks, queues, lists,...
- if finite: exponential growth in
  - number of parallel components,
    e.g., state space of $\mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$ is $S_1 \times \ldots \times S_n$
  - number of variables and channels

State explosion problem

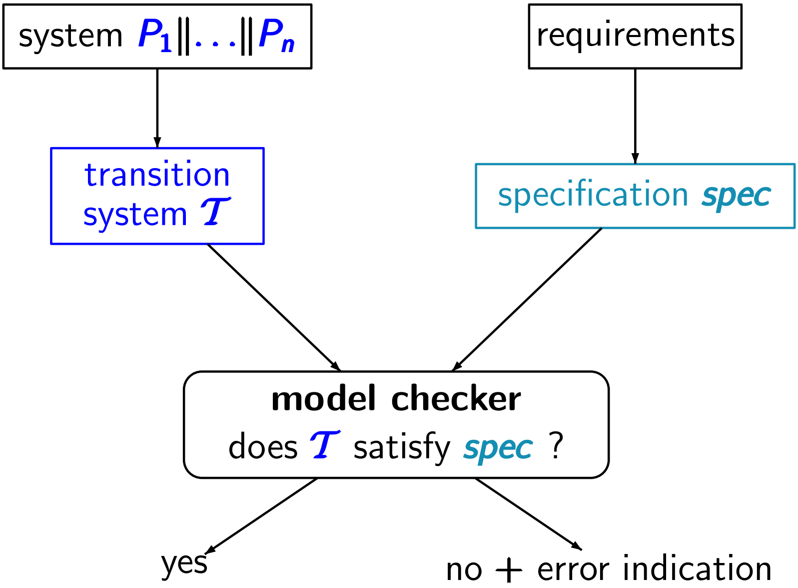
TS for reactive systems can be enormously large

- infinite for systems with
  - ∗ variables of infinite domains, e.g., $\mathbb{N}$
  - ∗ infinite data structures, e.g., stacks, queues, lists,...
- if finite: exponential growth in
  - ∗ number of parallel components,
    e.g., state space of $\mathcal{T}_1 \parallel \ldots \parallel \mathcal{T}_n$ is $S_1 \times \ldots \times S_n$
  - ∗ number of variables and channels

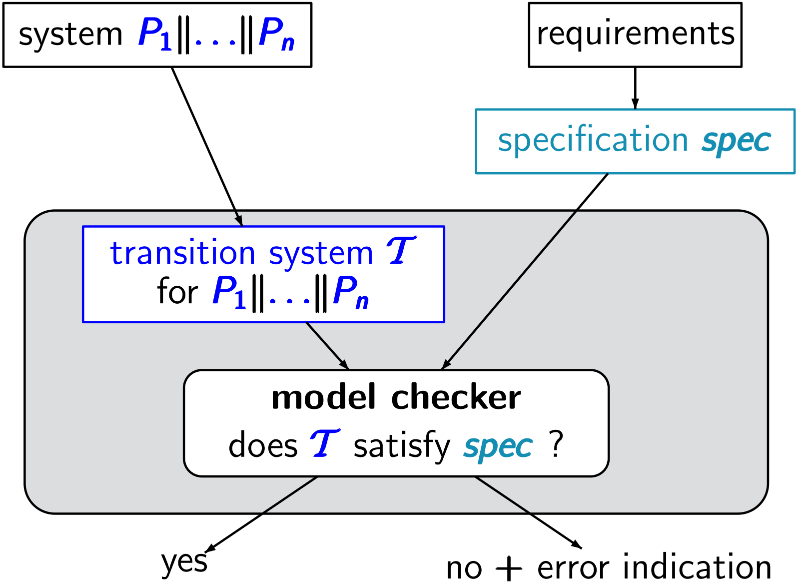
e.g., for channel systems: size of the state space is

$$|Loc_1| \cdot \ldots \cdot |Loc_n| \cdot \prod_{x \in Var} |Dom(x)| \cdot \prod_{c \in Chan} |Dom(c)|^{cap(c)}$$

Model checking



Model checking



Model checking