Modelling and Analysing Concurrent Systems

RIO 2023 Summer School of Informatics
Rio Cuarto, Argentina; February 13–17, 2023

Lecture 1: Milner's Calculus of Communicating Systems

Thomas Noll
Software Modelling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-22-23/rio/

Outline of Lecture 1

Staff

About me

- Associate professor at the Software Modelling and Verification Group (MOVES) in the Department of Computer Science at RWTH Aachen University
- Research interests:
  - Reliability, Safety and Security of Hardware/Software Systems
  - Static Program Analysis for Software Optimisation and Verification
  - Formal Verification of Artificial Neural Networks
- Teaching activities:
  - Courses on *Concurrency Theory*
  - Courses on *Semantics and Verification of Software*
  - Courses on *Compiler Construction*
  - Courses on *Static Program Analysis*
  - Bridging courses on *Foundations of Informatics*
  - Seminars on advanced topics
  - Supervision of Bachelor's and Master's theses

Staff

About me

- Associate professor at the Software Modelling and Verification Group (MOVES) in the Department of Computer Science at RWTH Aachen University
- Research interests:
  - Reliability, Safety and Security of Hardware/Software Systems
  - Static Program Analysis for Software Optimisation and Verification
  - Formal Verification of Artificial Neural Networks
- Teaching activities:
  - Courses on *Concurrency Theory*
  - Courses on *Semantics and Verification of Software*
  - Courses on *Compiler Construction*
  - Courses on *Static Program Analysis*
  - Bridging courses on *Foundations of Informatics*
  - Seminars on advanced topics
  - Supervision of Bachelor's and Master's theses

Course Objectives

Objectives

- Understand the foundations of concurrent systems
- Understand the main semantical underpinnings of concurrency
- Model, reason about, and compare concurrent systems in a rigorous manner

Course Objectives

Objectives

- Understand the foundations of concurrent systems
- Understand the main semantical underpinnings of concurrency
- Model, reason about, and compare concurrent systems in a rigorous manner

Motivation

- Supporting the design phase of systems
  - "Programming Concurrent Systems"
  - synchronisation, scheduling, semaphores, ...

Course Objectives

Objectives

- Understand the foundations of concurrent systems
- Understand the main semantical underpinnings of concurrency
- Model, reason about, and compare concurrent systems in a rigorous manner

Motivation

- Supporting the design phase of systems
  - "Programming Concurrent Systems"
  - synchronisation, scheduling, semaphores, ...
- Verifying functional correctness properties
  - "Model Checking"
  - validation of mutual exclusion, fairness, absence of deadlocks, ...

Course Objectives

Objectives

- Understand the foundations of concurrent systems
- Understand the main semantical underpinnings of concurrency
- Model, reason about, and compare concurrent systems in a rigorous manner

Motivation

- Supporting the design phase of systems
  - "Programming Concurrent Systems"
  - synchronisation, scheduling, semaphores, ...
- Verifying functional correctness properties
  - "Model Checking"
  - validation of mutual exclusion, fairness, absence of deadlocks, ...
- Comparing expressivity of models of concurrency
  - "interleaving" vs. "true concurrency"
  - equivalence, refinement, abstraction, ...

Organisation of the Course

Organisation

- All material (slides, exercises, ...) made available via
  `https://moves.rwth-aachen.de/teaching/ws-22-23/rio/`
- Schedule: Mon Feb 13 – Thu Feb 16, 10:30 – 13:00
- Exam Fri Feb 17 morning

Outline of Lecture 1

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2)$$

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2)$$

- At first glance: x is assigned 3

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2)$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$\begin{array}{c} x := 0; \\ (x := x + 1 \parallel x := x + 2) \\ 1 \end{array} \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$\begin{array}{c} x := 0; \\ (\underset{1}{x := x + 1} \parallel \underset{2}{x := x + 2}) \end{array} \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$\begin{array}{c} x := 0; \\ (\underset{1}{x := x + 1} \parallel \underset{2}{x := x + 2}) \end{array} \qquad \text{value of } x\text{: } 1$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 2$$
$$2$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$\begin{array}{c} x := 0; \\ (x := x + 1 \parallel x := x + 2) \\ 1 \end{array} \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(\underset{1}{x := x + 1} \parallel \underset{2}{x := x + 2}) \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(\underbrace{x := x+1}_{1} \parallel \underbrace{x := x+2}_{2}) \qquad \text{value of } x\text{: } 2$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$
\begin{array}{c}
x := 0; \\
(x := x + 1 \parallel x := x + 2) \\
1
\end{array}
\qquad \text{value of } x\text{: } 1
$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 0$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 0$$
$$2$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 2$$
$$2$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 2$$
$$3$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1,

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2) \qquad \text{value of } x\text{: } 3$$
$$3$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1, or 3

Concurrency and Interaction by Example

Observation: concurrency introduces new phenomena

Example 1.1

$$x := 0;$$
$$(x := x + 1 \parallel x := x + 2)$$

- At first glance: x is assigned 3
- But: both parallel components could read x before it is written
- Thus: x is assigned 2, 1, or 3
- If exclusive access to shared memory and atomic execution of assignments guaranteed
  $\Rightarrow$ only possible outcome: 3

Concurrency and Interaction

The problem arises due to the combination of

- concurrency and
- interaction (here: via shared memory)

Concurrency and Interaction

The problem arises due to the combination of

- concurrency and
- interaction (here: via shared memory)

Conclusion

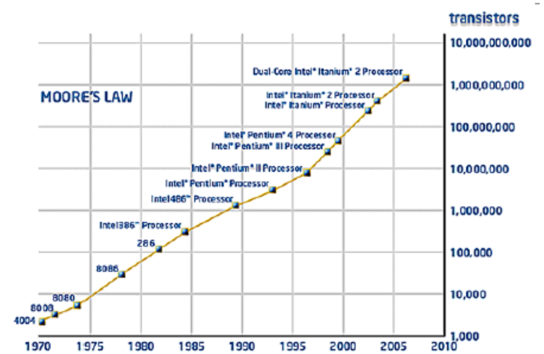
When modelling concurrent systems, the precise description of the mechanisms of both concurrency and interaction is crucially important.

Concurrency Everywhere

Herb Sutter: *The Free Lunch Is Over*, Dr. Dobb's Journal, 30(3), 2005

"The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency."

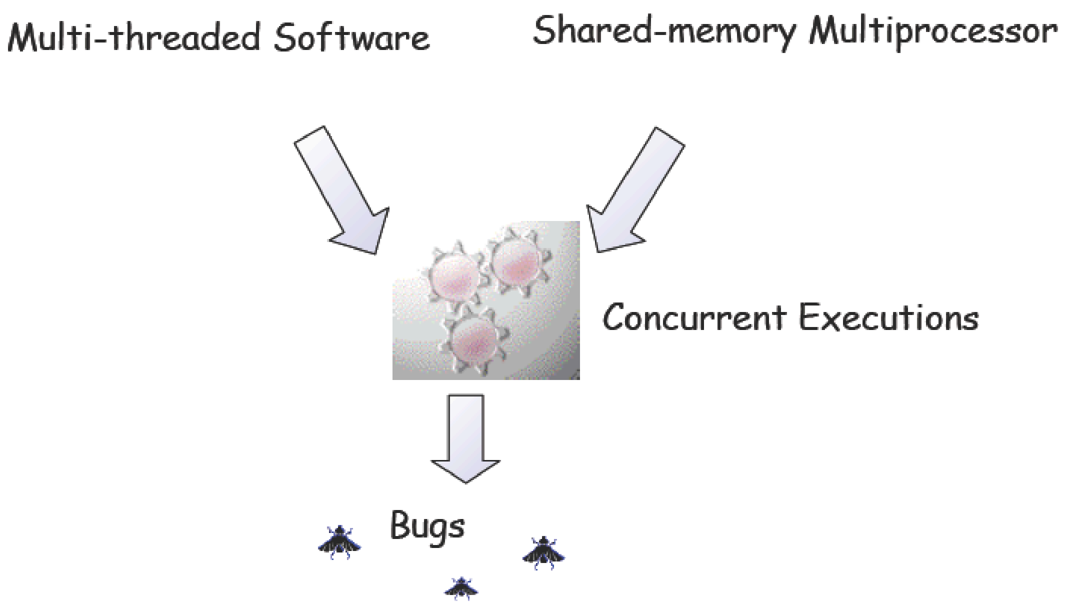
- Operating systems
- Embedded/reactive systems
  - parallelism (at least) between hardware, software, and environment
- High-end parallel hardware infrastructure:
  - high-performance computing
- Low-end parallel hardware infrastructure
  - increasing performance only achievable by parallelism
  - multi-core computers, GPGPUs, FPGAs



Moore's Law: Transistor density doubles every 2 years

Problems Everywhere

- Operating systems:
  - mutual exclusion
  - fairness (no starvation)
  - no deadlocks, ...
- Shared-memory systems:
  - memory models
  - data races
  - inconsistencies ("sequential consistency" vs. relaxed notions)
- Embedded systems:
  - safety
  - liveness, ...



Outline of Lecture 1

Memory Models

An illustrative example

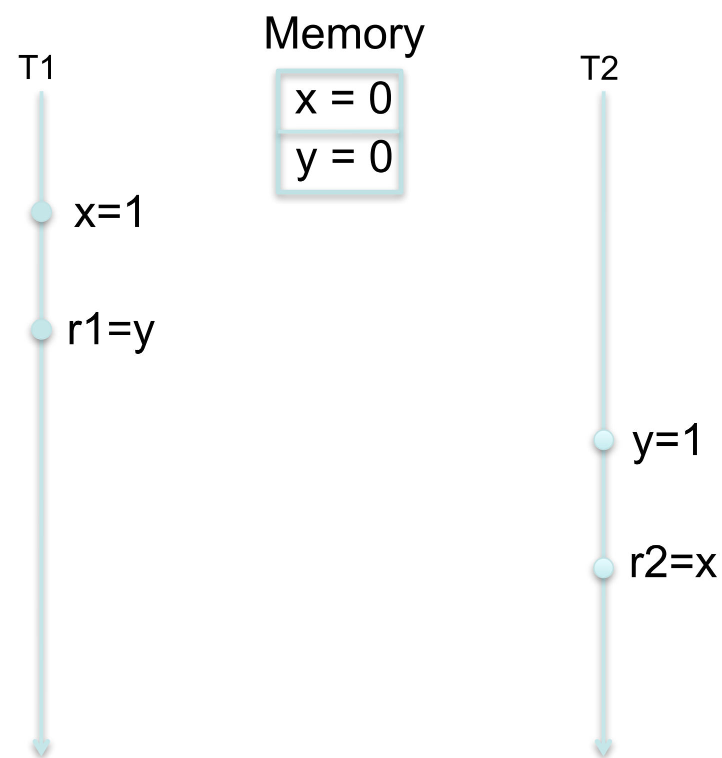
Initially: x = y = 0

thread1:	thread2:
1: x = 1	3: y = 1
2: r1 = y	4: r2 = x

(with global variables x, y and local registers r1, r2)

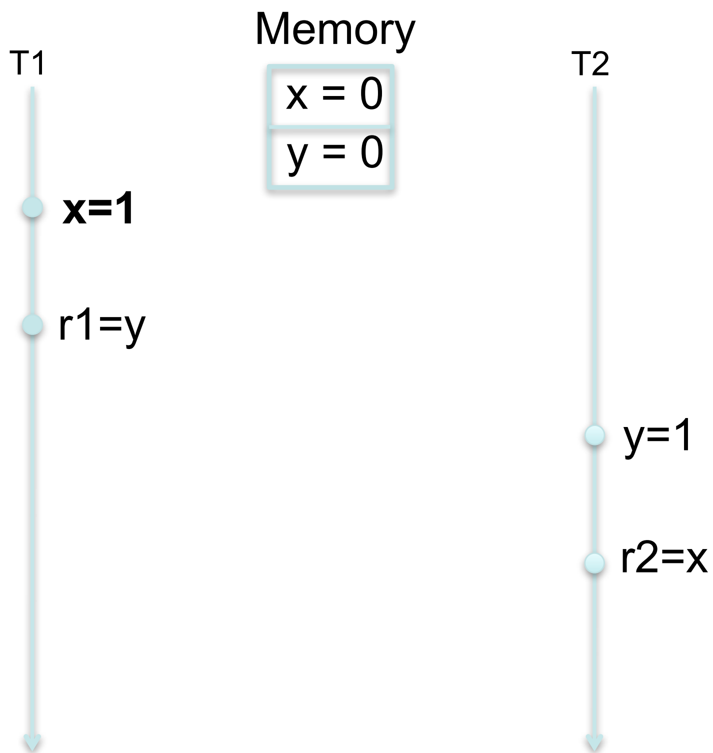
Memory Models

Sequential Consistency (SC)

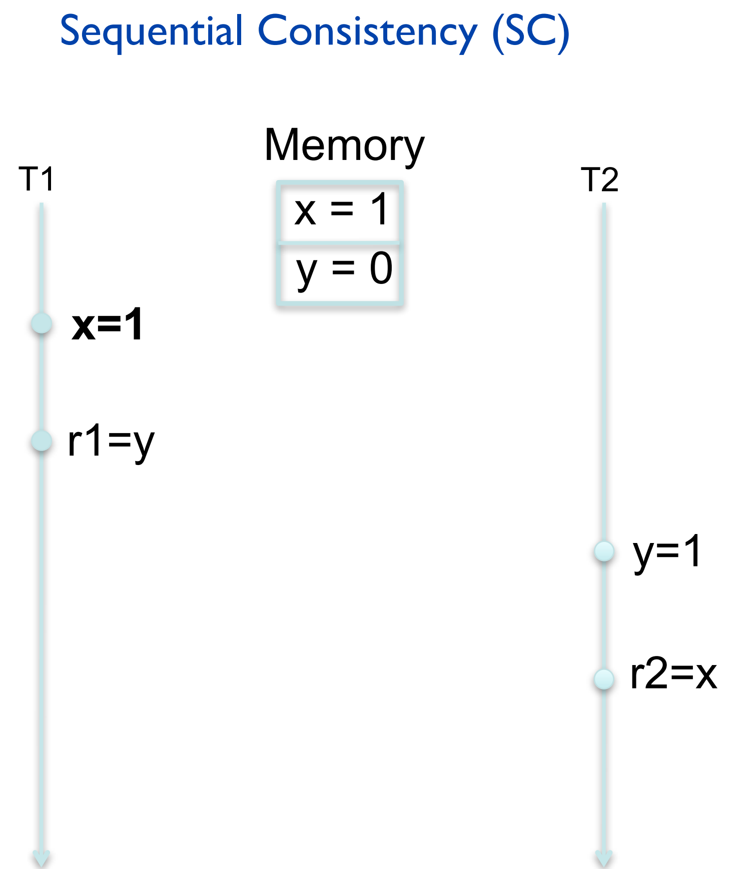


Memory Models

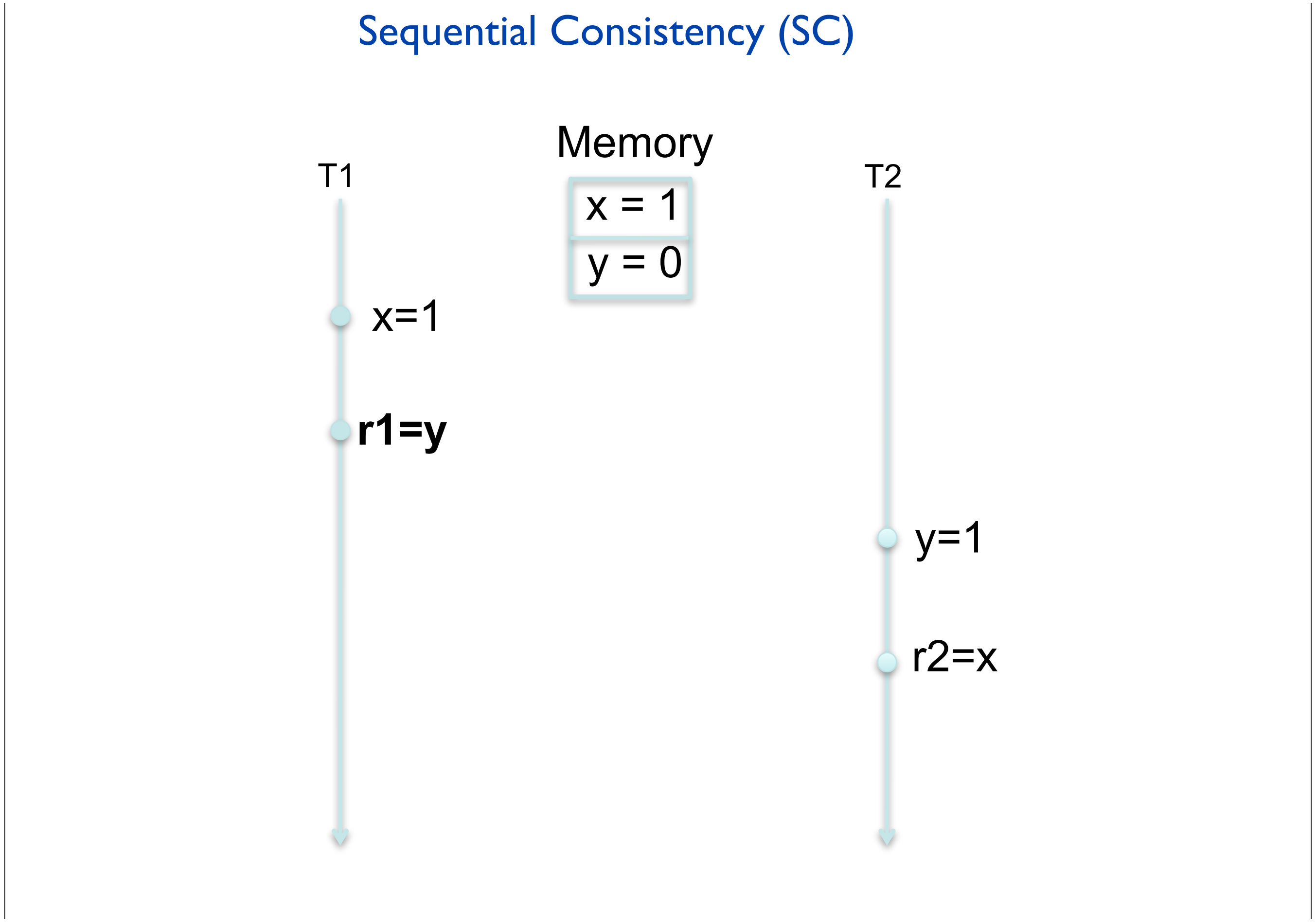
Sequential Consistency (SC)



Memory Models

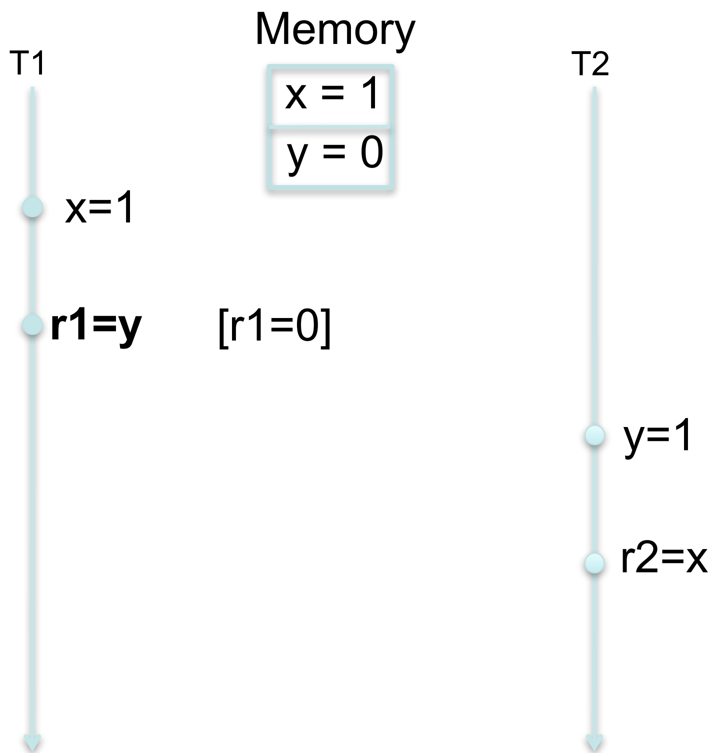


Memory Models



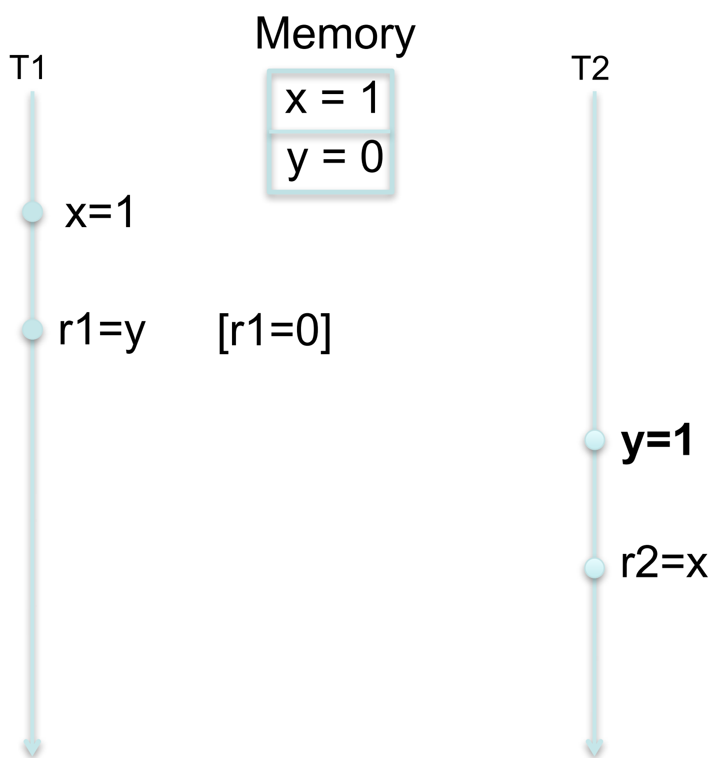
Memory Models

Sequential Consistency (SC)

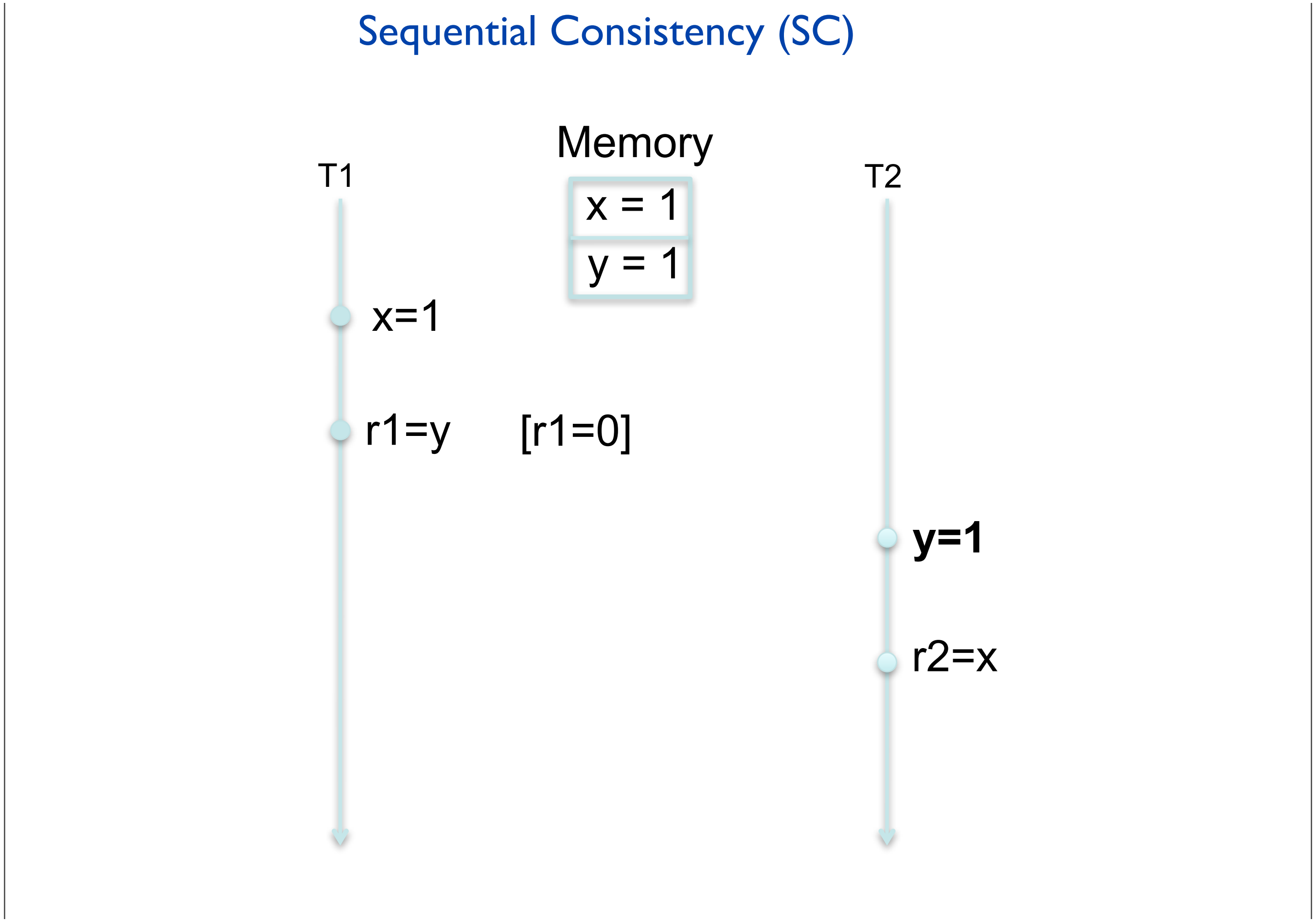


Memory Models

Sequential Consistency (SC)

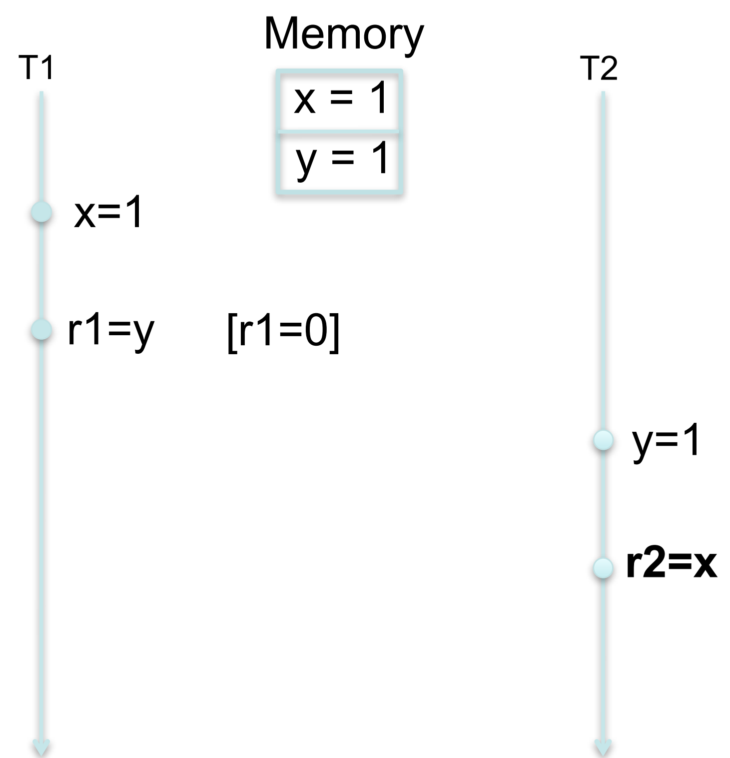


Memory Models

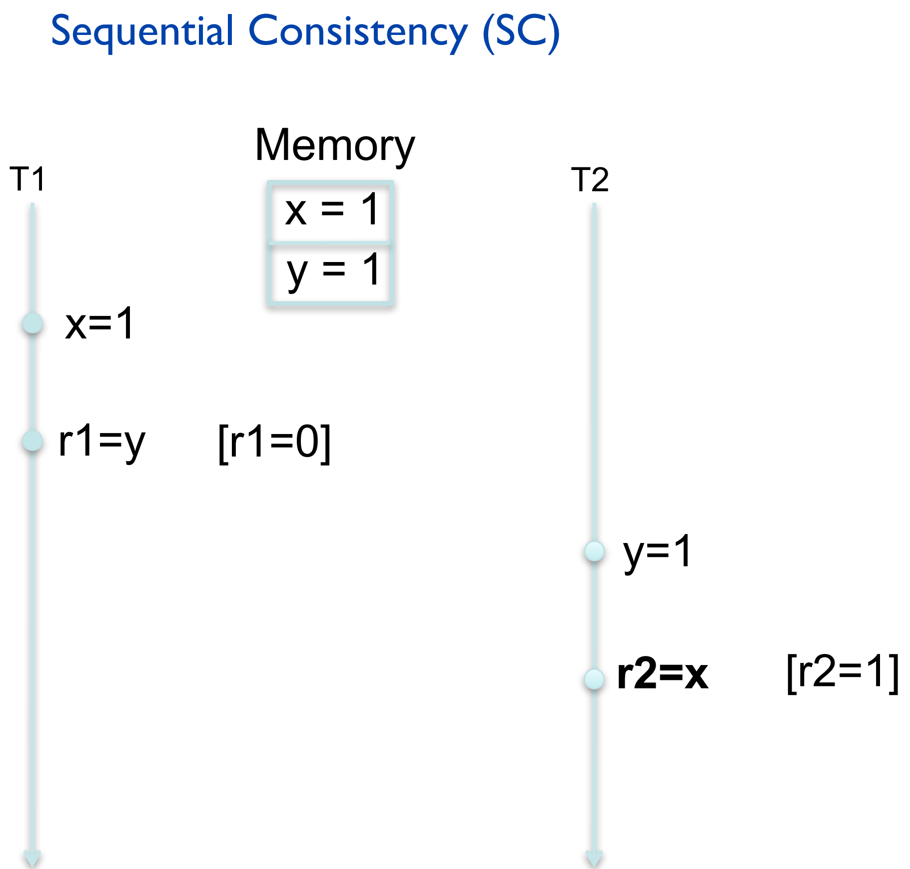


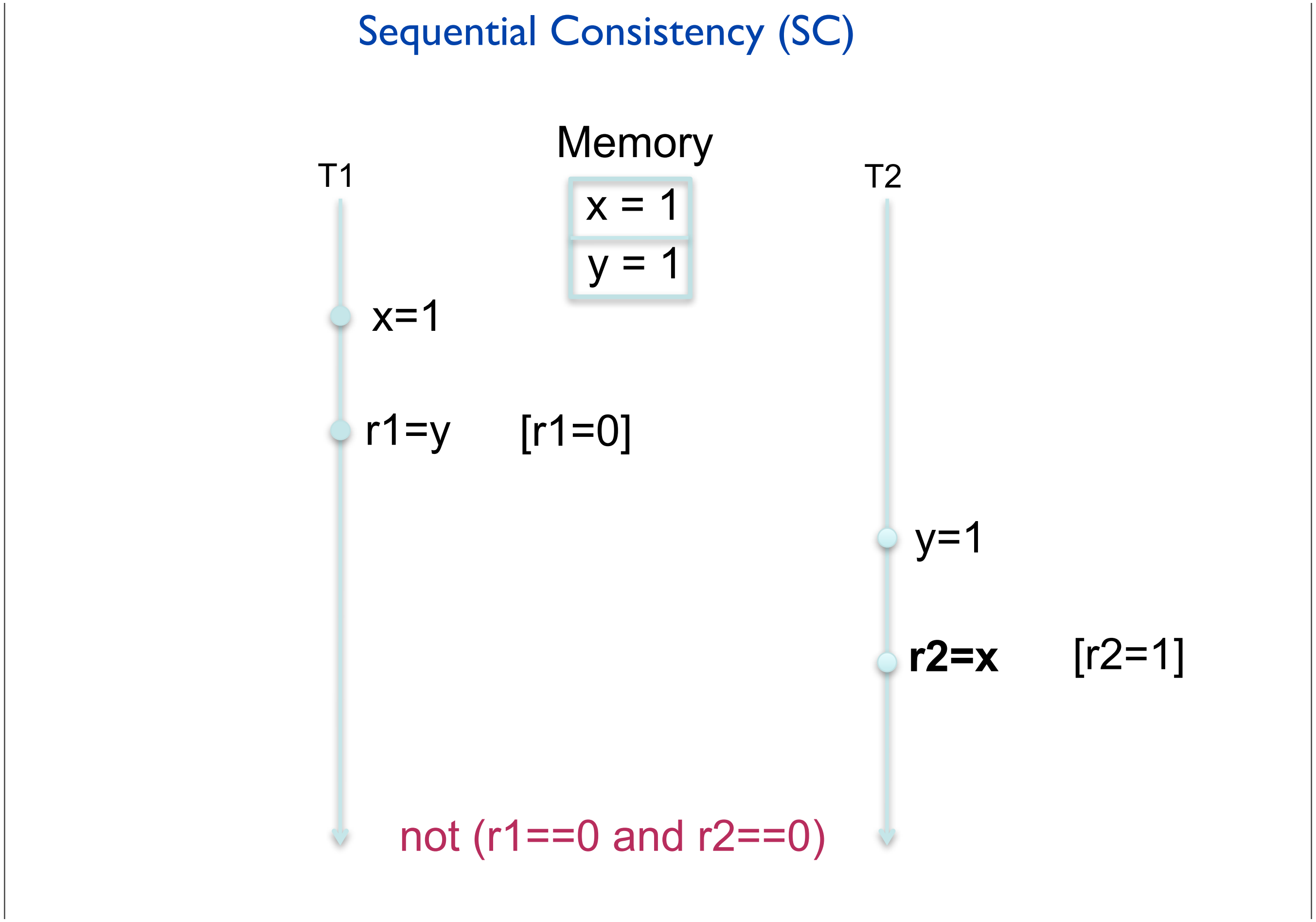
Memory Models

Sequential Consistency (SC)



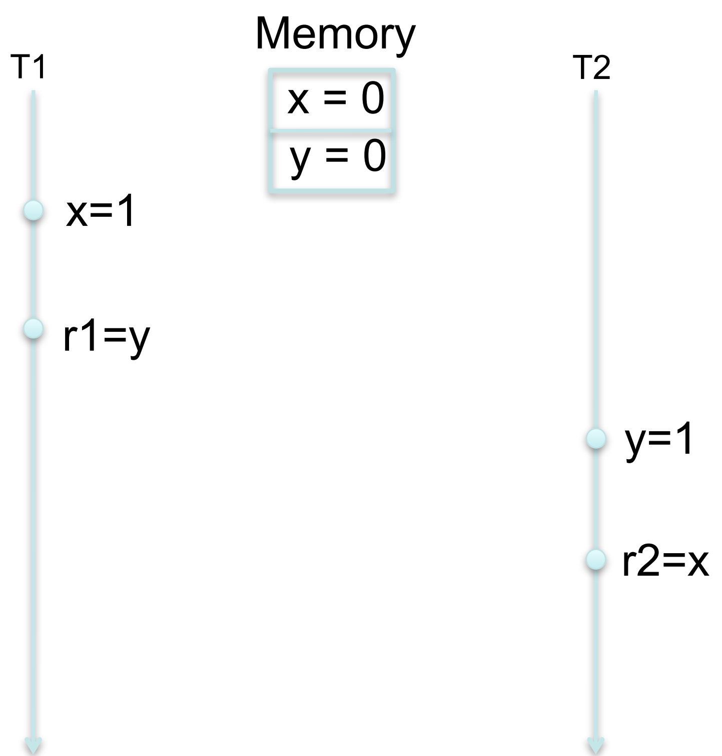
Memory Models




Sequential Consistency (SC)
Memory
x = 1
y = 1
T1
T2
x=1
r1=y
[r1=0]
y=1
r2=x
[r2=1]
not (r1==0 and r2==0)

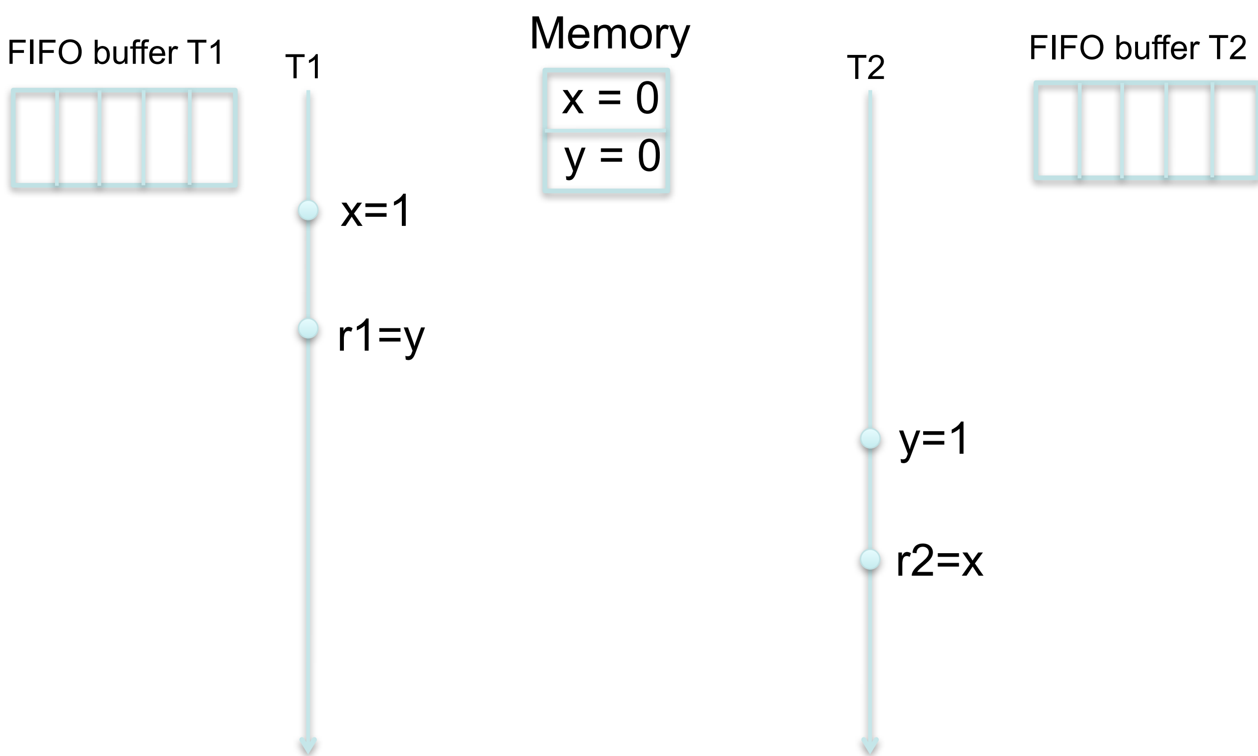
Memory Models

Total Store Ordering (TSO)



Memory Models

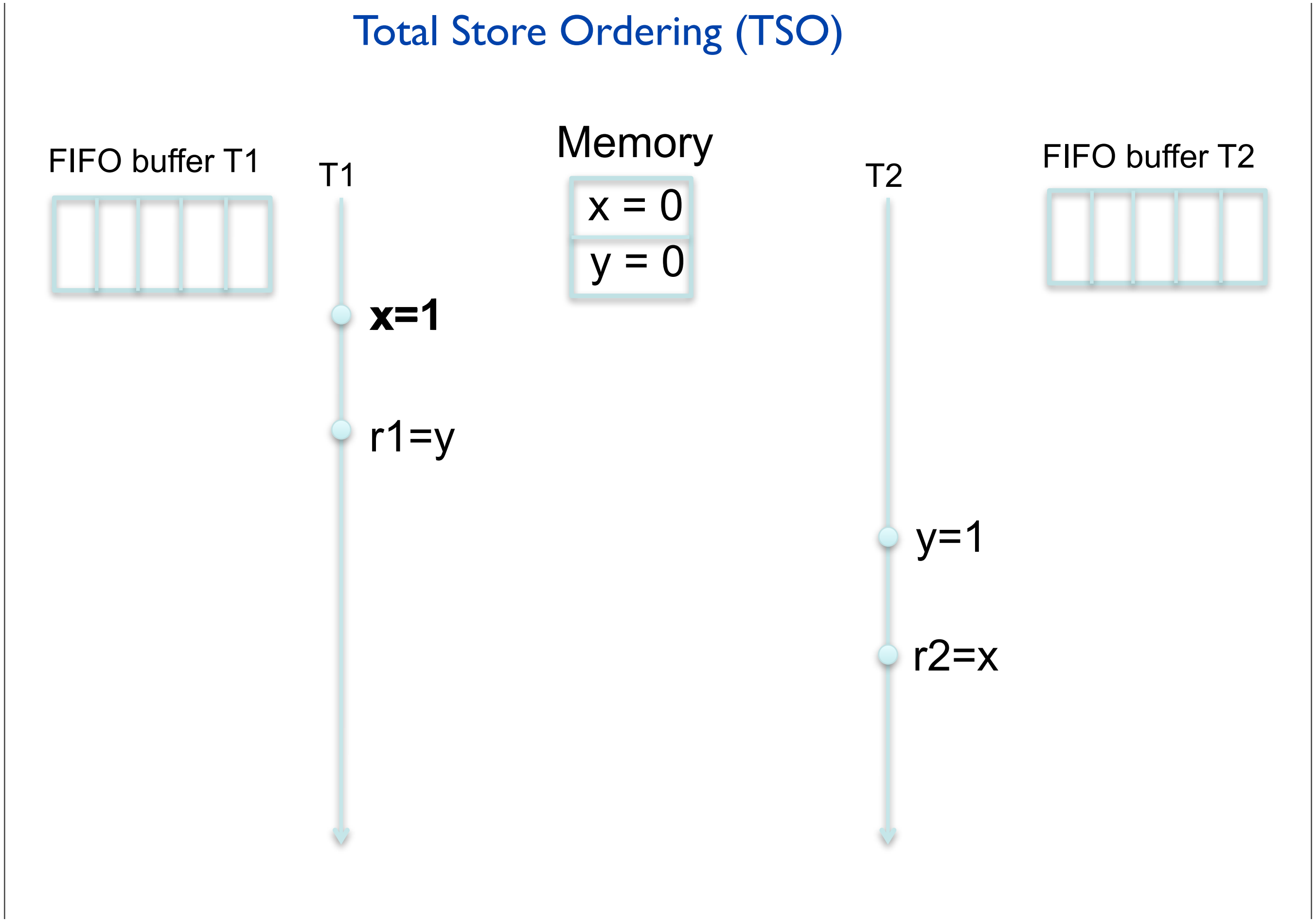
Total Store Ordering (TSO)



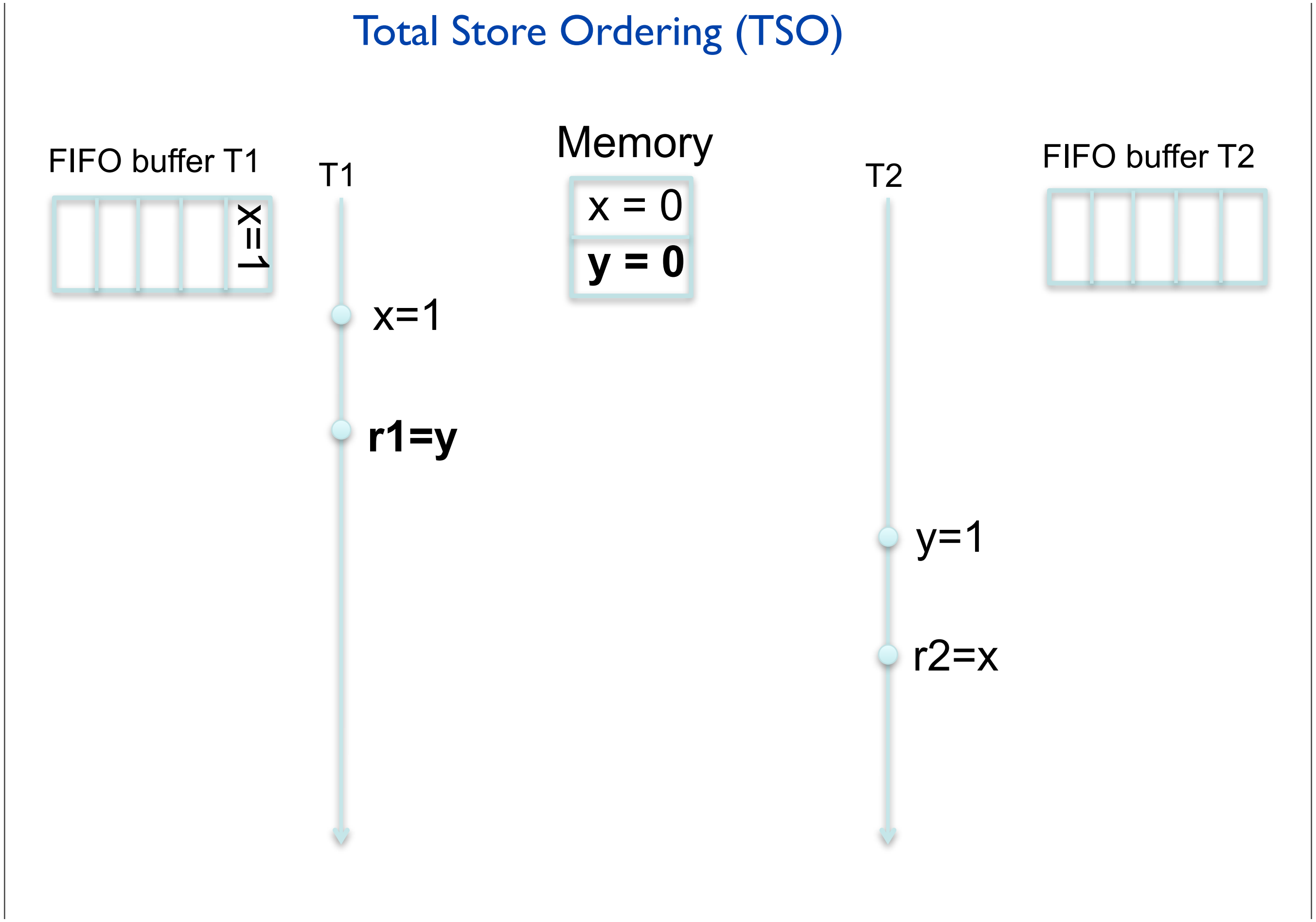
Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

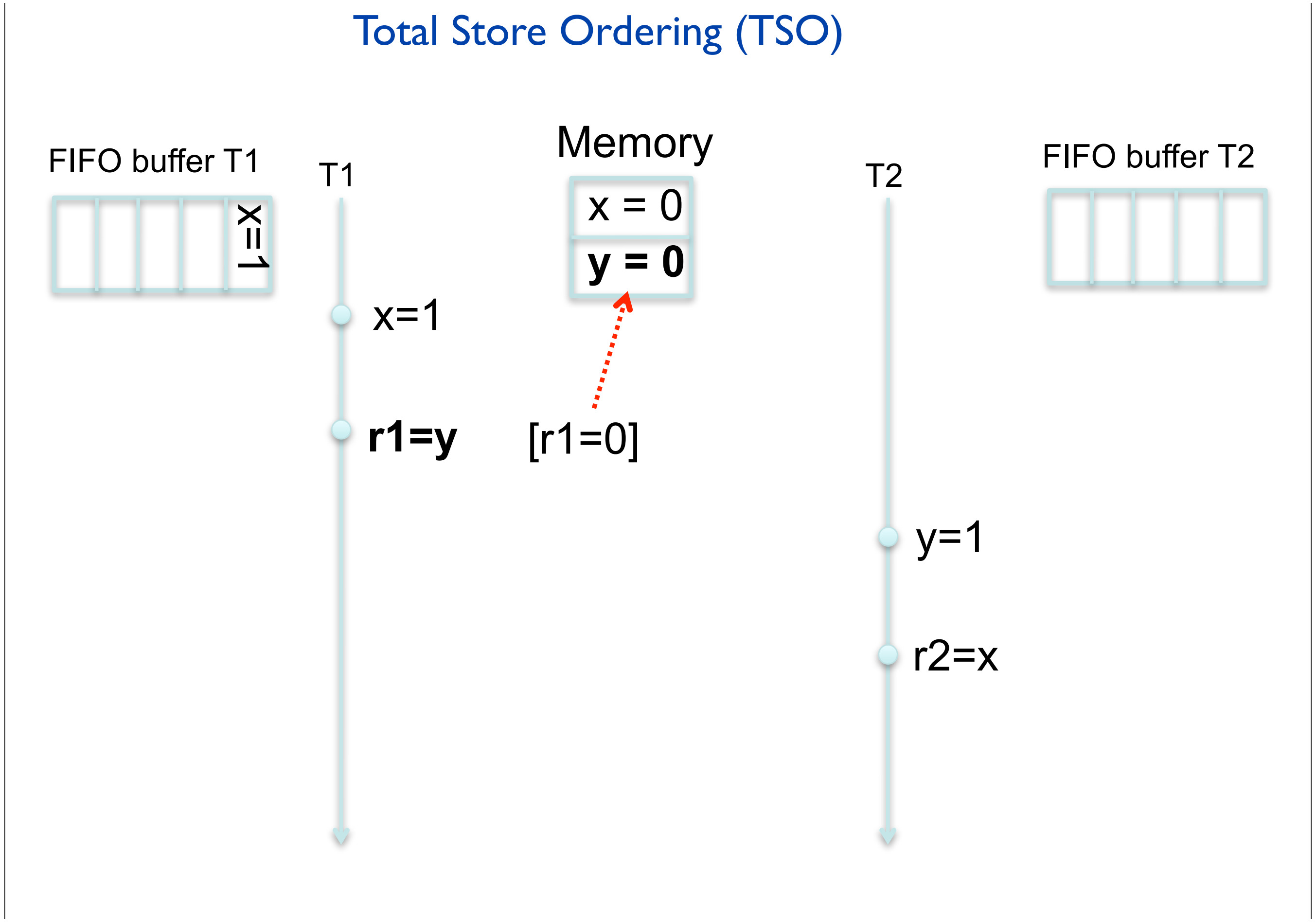
Memory Models



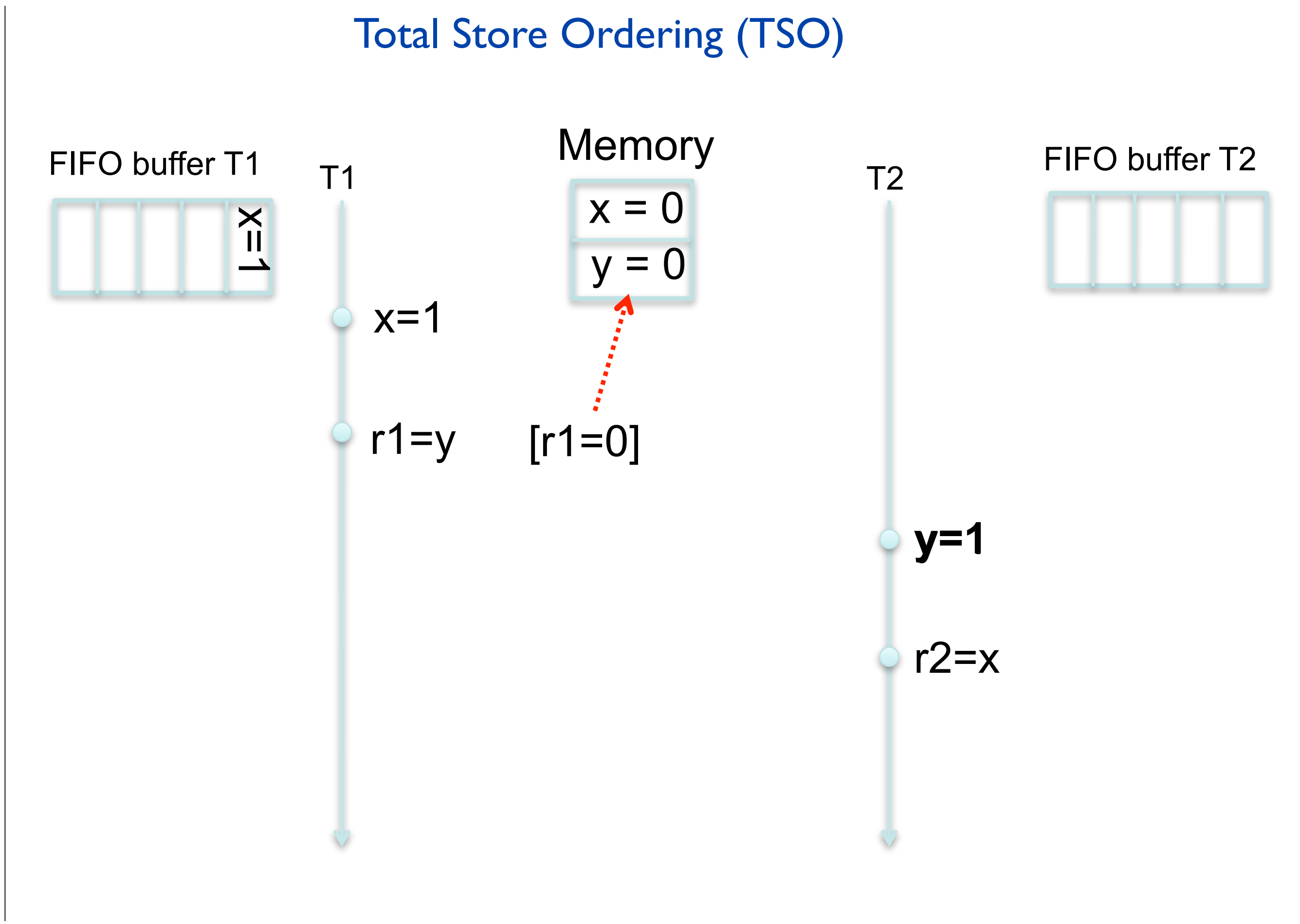
Memory Models



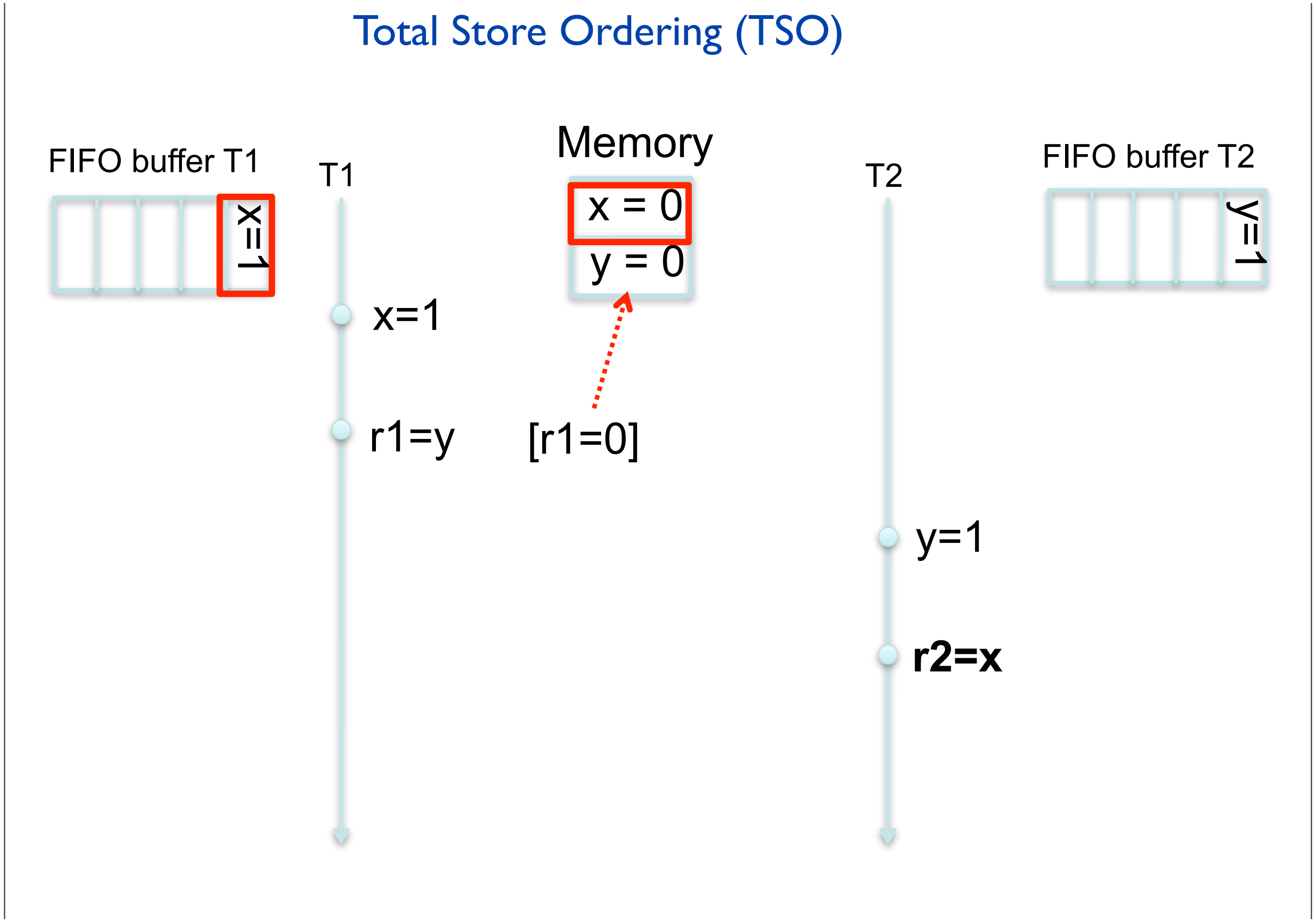
Memory Models



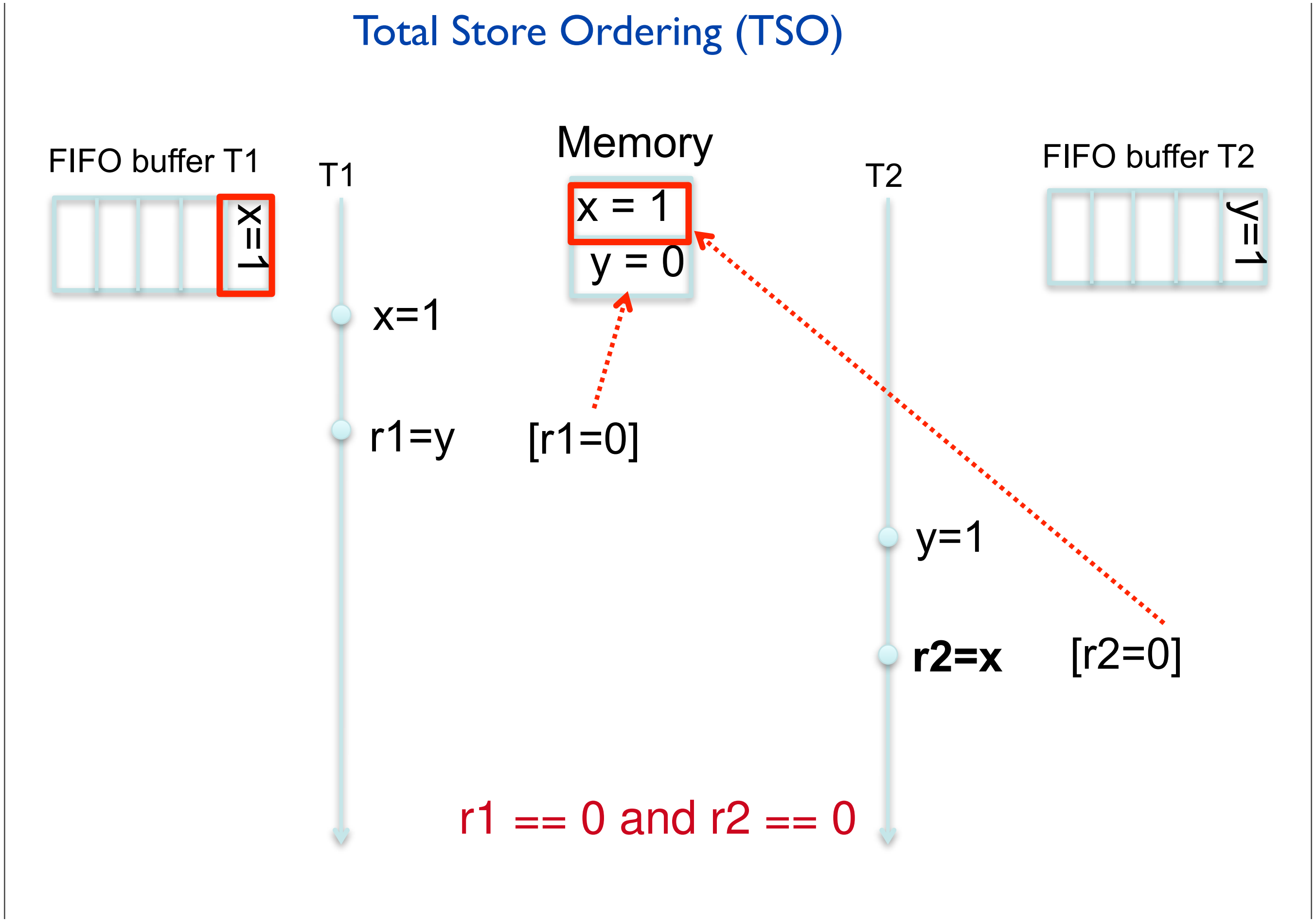
Memory Models



Memory Models



Memory Models



Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

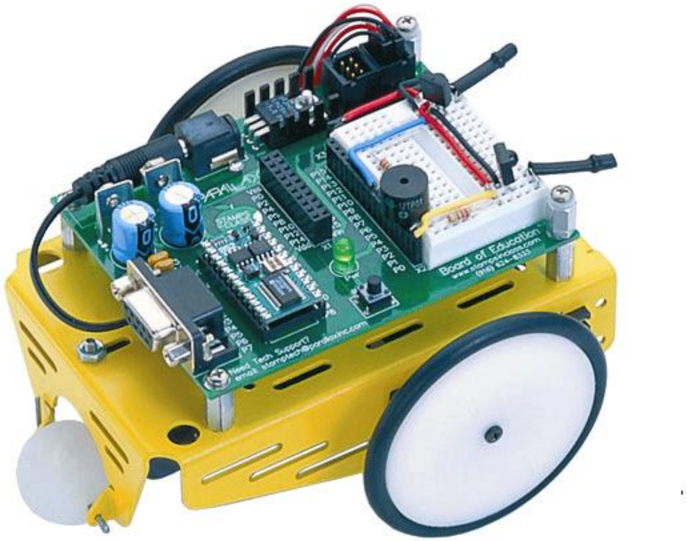
Outline of Lecture 1

Reactive Systems I

- "Classical" model for sequential systems

$$\mathit{System} : \mathit{Input} \to \mathit{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction

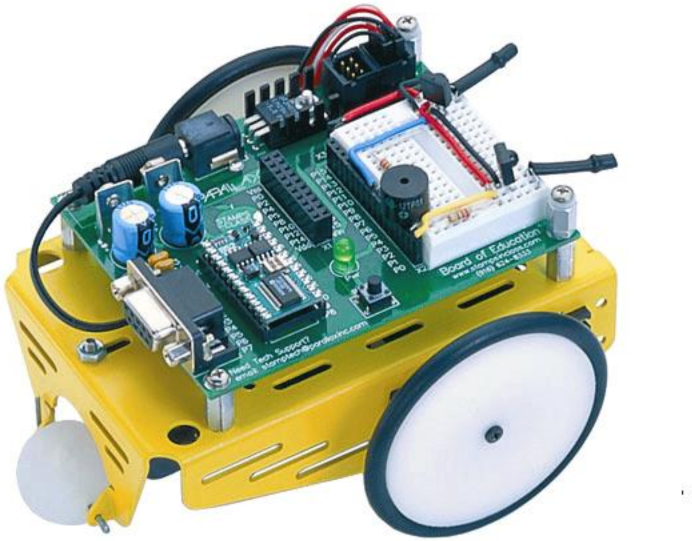


Reactive Systems I

- “Classical” model for sequential systems

$$\textit{System} : \textit{Input} \rightarrow \textit{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction
- Rather: reactive systems which interact with environment and among themselves

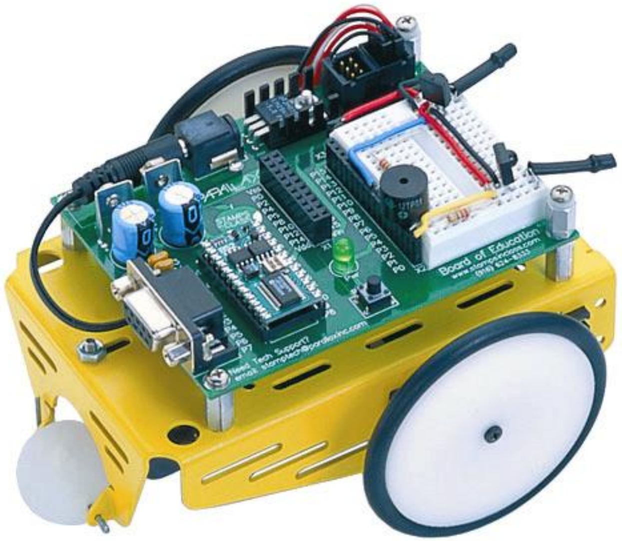


Reactive Systems I

- "Classical" model for sequential systems

$$\mathit{System} : \mathit{Input} \to \mathit{Output}$$

  (transformational systems) is not adequate
- Missing: aspect of interaction
- Rather: reactive systems which interact with environment and among themselves
- Main interest: not terminating computations but infinite behaviour (system maintains ongoing interaction with environment)
- Examples:
  - operating systems
  - embedded systems controlling mechanical or electrical devices (planes, cars, home appliances, ...)
  - power plants, production lines, ...



Reactive Systems II

Observation

Reactive systems are often safety critical, thus trustworthiness has to be ensured.

- Safety properties: “Nothing bad is ever going to happen.”
  - e.g., “at most one process in the critical section”
- Liveness properties: “Eventually something good will happen.”
  - e.g., “every request will finally be answered by the server”
- Fairness properties: “No component will starve to death.”
  - e.g., “any process requiring entry to the critical section will eventually be admitted”
- Reliability, performance, survivability, ...

Outline of Lecture 1

Overview of the Course

Overview

(1) Milner's Calculus of Communicating Systems (CCS)
- introduction and motivation
- syntax of CCS
- semantics of CCS
- the CAAL tool

(2) Behavioural Equivalences
- trace equivalence
- bisimulation
- congruence
- deadlock sensitivity

(3) Logical Specifications
- Hennessy-Milner Logic
- HML and traces
- HML and bisimulation
- adding recursion

(4) Application: Mutual-Exclusion Protocols
- modelling mutex algorithms in CCS
- verification by model checking
- verification by bisimulation checking

Overview of the Course

Overview

(1) Milner's Calculus of Communicating Systems (CCS)
- introduction and motivation
- syntax of CCS
- semantics of CCS
- the CAAL tool

(2) Behavioural Equivalences
- trace equivalence
- bisimulation
- congruence
- deadlock sensitivity

(3) Logical Specifications
- Hennessy-Milner Logic
- HML and traces
- HML and bisimulation
- adding recursion

(4) Application: Mutual-Exclusion Protocols
- modelling mutex algorithms in CCS
- verification by model checking
- verification by bisimulation checking

Literature

Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen and Jiří Srba: *Reactive Systems: Modelling, Specification and Verification*, Cambridge Univ. Press, 2007

Outline of Lecture 1

The Calculus of Communicating Systems

History

- First development:
  Robin Milner: *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980
- Elaboration and larger case studies:
  Robin Milner: *Communication and Concurrency*, Prentice-Hall, 1989
- Extension to mobile systems:
  Robin Milner: *Communicating and Mobile Systems: the π-calculus*, Cambridge University Press, 1999

The Calculus of Communicating Systems

History

- First development:
  Robin Milner: *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980
- Elaboration and larger case studies:
  Robin Milner: *Communication and Concurrency*, Prentice-Hall, 1989
- Extension to mobile systems:
  Robin Milner: *Communicating and Mobile Systems: the π-calculus*, Cambridge University Press, 1999

Approach

Description of concurrency on a simple and abstract level, using only a few basic primitives

- no explicit storage (variables)
- no explicit representation of values (numbers, Booleans, ...) or data structures

$\Rightarrow$ Concurrent system reduced to communication potential

Outline of Lecture 1

Syntax of CCS I

Definition 1.2 (Syntax of CCS)

- Let A be a set of (action) names.

Syntax of CCS I

Definition 1.2 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.

Syntax of CCS I

Definition 1.2 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.

Syntax of CCS I

Definition 1.2 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.

Syntax of CCS I

Definition 1.2 (Syntax of CCS)

- Let A be a set of (action) names.
- $\overline{A} := \{\overline{a} \mid a \in A\}$ denotes the set of co-names.
- $Act := A \cup \overline{A} \cup \{\tau\}$ is the set of actions with the silent (or: unobservable) action τ.
- Let Pid be a set of process identifiers.
- The set Prc of process expressions is defined by the following syntax:

$$
\begin{array}{rlll}
P ::= & \mathrm{nil} & & \text{(inaction)} \\
\mid & \alpha.P & & \text{(prefixing)} \\
\mid & P_1 + P_2 & & \text{(choice)} \\
\mid & P_1 \parallel P_2 & & \text{(parallel composition)} \\
\mid & P \setminus L & & \text{(restriction)} \\
\mid & P[f] & & \text{(relabelling)} \\
\mid & C & & \text{(process call)}
\end{array}
$$

where $\alpha \in Act$, $\emptyset \neq L \subseteq A$, $C \in Pid$, and $f : Act \to Act$ such that $f(\tau) = \tau$ and $f(\overline{a}) = \overline{f(a)}$ for each $a \in A$.

Syntax of CCS II

Definition 1.2 (continued)

- A (recursive) process definition is an equation system of the form

$$(C_i = P_i \mid 1 \leq i \leq k)$$

  where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Syntax of CCS II

Definition 1.2 (continued)

- A (recursive) process definition is an equation system of the form

$$(C_i = P_i \mid 1 \leq i \leq k)$$

where $k \geq 1$, $C_i \in Pid$ (pairwise distinct), and $P_i \in Prc$ (with identifiers from $\{C_1, \ldots, C_k\}$).

Notational Conventions:

- $\overline{\overline{a}}$ means a
- $\sum_{i=1}^{n} P_i$ ($n \in \mathbb{N}$) means $P_1 + \ldots + P_n$ (where $\sum_{i=1}^{0} P_i := \text{nil}$)
- $P \setminus a$ abbreviates $P \setminus \{a\}$
- $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$ stands for $f : Act \to Act$ with $f(a_i) = b_i$ for $i \in [n]$ and $f(\alpha) = \alpha$ otherwise
- Restriction and relabelling bind stronger than prefixing, prefixing stronger than composition, composition stronger than choice:

$$P \setminus a + b.Q \parallel R \quad \text{means} \quad (P \setminus a) + ((b.Q) \parallel R)$$

Outline of Lecture 1

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2.
- $P_1 \parallel P_2$ denotes the parallel execution of P_1 and P_2, involving interleaving or communication.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2.
- $P_1 \parallel P_2$ denotes the parallel execution of P_1 and P_2, involving interleaving or communication.
- The restriction $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2.
- $P_1 \parallel P_2$ denotes the parallel execution of P_1 and P_2, involving interleaving or communication.
- The restriction $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.
- The relabelling $P[f]$ allows to adapt the naming of actions.

Meaning of CCS Constructs

- nil is an inactive process that can do nothing.
- $\alpha.P$ can execute α and then behaves as P.
- An action $a \in A$ ($\overline{a} \in \overline{A}$) is interpreted as an input (output, resp.) operation. Both are complementary: if performed in parallel (i.e., in $P_1 \parallel P_2$), they are merged into a τ-action.
- $P_1 + P_2$ represents the nondeterministic choice between P_1 and P_2.
- $P_1 \parallel P_2$ denotes the parallel execution of P_1 and P_2, involving interleaving or communication.
- The restriction $P \setminus L$ declares each $a \in L$ as a local name which is only known within P.
- The relabelling $P[f]$ allows to adapt the naming of actions.
- The behaviour of a process call C is given by the right-hand side of the corresponding equation.

CCS Examples

Example 1.3

(1) One-place buffer:

$$B = in.\overline{out}.B$$

Software Modeling and Verification Chair

CCS Examples

Example 1.3

(1) One-place buffer:

$$B = in.\overline{out}.B$$

(2) Two-place buffer:

$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

CCS Examples

Example 1.3

(1) One-place buffer:

$$B = in.\overline{out}.B$$

(2) Two-place buffer:

$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

(3) Parallel two-place buffer:

$$\begin{aligned} B_\parallel &= (B[out \mapsto com] \parallel B[in \mapsto com]) \setminus com \\ B &= in.\overline{out}.B \end{aligned}$$

"Interaction diagram":

$$\xrightarrow{in} \boxed{\xrightarrow{in} \boxed{\xrightarrow{in} B \xrightarrow{out}} \xrightarrow{com} \boxed{\xrightarrow{in} B \xrightarrow{out}} \xrightarrow{out}} \xrightarrow{out}$$

Outline of Lecture 1

Labelled Transition Systems

Goal: represent system behaviour by (infinite) graph

- nodes = system states
- edges = transitions between states

Labelled Transition Systems

Goal: represent system behaviour by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 1.4 (Labelled transition system)

A labelled transition system (LTS) is a triple $(S, Act, \longrightarrow)$ consisting of

- a set S of states
- a set Act of (action) labels
- a transition relation $\longrightarrow \subseteq S \times Act \times S$

For $(s, \alpha, s') \in \longrightarrow$ we write $s \xrightarrow{\alpha} s'$. An LTS is called finite if S is so.

Labelled Transition Systems

Goal: represent system behaviour by (infinite) graph

- nodes = system states
- edges = transitions between states

Definition 1.4 (Labelled transition system)

A labelled transition system (LTS) is a triple $(S, Act, \longrightarrow)$ consisting of

- a set S of states
- a set Act of (action) labels
- a transition relation $\longrightarrow \subseteq S \times Act \times S$

For $(s, \alpha, s') \in \longrightarrow$ we write $s \xrightarrow{\alpha} s'$. An LTS is called finite if S is so.

Remarks:

- Sometimes an initial state $s_0 \in S$ is distinguished ("$LTS(s_0)$").
- (Finite) LTSs correspond to (finite) automata without final states.

Semantics of CCS I

We define the assignment

$$\begin{aligned} \text{syntax} &\longrightarrow \text{semantics} \\ \text{process definition} &\longmapsto \text{LTS} \end{aligned}$$

by induction over the syntactic structure of process expressions. Here we employ derivation rules of the form

$$\text{(rule name)}\,\frac{\text{premise(s)}}{\text{conclusion}}$$

which are composed to form derivation trees (where axioms, i.e., rules without premises, correspond to leaves).

Semantics of CCS II

Reminder: $P ::= \mathrm{nil} \mid \alpha.P \mid P_1 + P_2 \mid P_1 \parallel P_2 \mid P \setminus L \mid P[f] \mid C$

Definition 1.5 (Semantics of CCS)

A process definition $(C_i = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc, Act, \longrightarrow)$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc$, $\alpha \in Act$, $\lambda \in A \cup \overline{A}$, $a \in A$):

$$\text{(Act)}\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{(Sum}_1\text{)}\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(Sum}_2\text{)}\frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(Par}_1\text{)}\frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Par}_2\text{)}\frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \text{(Com)}\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\text{(Res)}\frac{P \xrightarrow{\alpha} P' \quad (\alpha, \overline{\alpha} \notin L)}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \qquad \text{(Rel)}\frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Call)}\frac{P \xrightarrow{\alpha} P' \quad (C = P)}{C \xrightarrow{\alpha} P'}$$

Semantics of CCS III

Example 1.6

(1) One-place buffer: $B = in.\overline{out}.B$

– First step:

$$
\text{(Call)}\;\dfrac{\text{(Act)}\;\dfrac{}{in.\overline{out}.B \xrightarrow{in} \overline{out}.B}}{B \xrightarrow{in} \overline{out}.B}
$$

Semantics of CCS III

Example 1.6

(1) One-place buffer: $B = in.\overline{out}.B$

– First step:

$$\text{(Call)}\ \dfrac{\text{(Act)}\ \dfrac{}{in.\overline{out}.B \xrightarrow{in} \overline{out}.B}}{B \xrightarrow{in} \overline{out}.B}$$

– Second step:

$$\text{(Act)}\ \dfrac{}{\overline{out}.B \xrightarrow{\overline{out}} B}$$

Semantics of CCS III

Example 1.6

(1) One-place buffer: $B = in.\overline{out}.B$

– First step:

$$\text{(Call)}\ \dfrac{\text{(Act)}\ \dfrac{}{in.\overline{out}.B \xrightarrow{in} \overline{out}.B}}{B \xrightarrow{in} \overline{out}.B}$$

– Second step:

$$\text{(Act)}\ \dfrac{}{\overline{out}.B \xrightarrow{\overline{out}} B}$$

$\Rightarrow$ Complete LTS:

$$\begin{tikzcd} {} \arrow[r] & \boxed{B} \arrow[rr, bend left, "in"] & & \boxed{\overline{out}.B} \arrow[ll, bend left, "\overline{out}"] \end{tikzcd}$$

Semantics of CCS IV

Example 1.6 (continued)

(2) Sequential two-place buffer:
$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

– First step:

$$\dfrac{\dfrac{}{in.B_1 \xrightarrow{in} B_1}\,(\mathrm{Act})}{B_0 \xrightarrow{in} B_1}\,(\mathrm{Call})$$

Example 1.6 (continued)

(2) Sequential two-place buffer:
$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

– First step:

$$\text{(Call)}\,\dfrac{\text{(Act)}\,\dfrac{}{in.B_1 \xrightarrow{in} B_1}}{B_0 \xrightarrow{in} B_1}$$

– Second step:

$$\text{(Call)}\,\dfrac{\text{(Sum}_1\text{)}\,\dfrac{\text{(Act)}\,\dfrac{}{\overline{out}.B_0 \xrightarrow{\overline{out}} B_0}}{\overline{out}.B_0 + in.B_2 \xrightarrow{\overline{out}} B_0}}{B_1 \xrightarrow{\overline{out}} B_0}$$

Semantics of CCS IV

Example 1.6 (continued)

(2) Sequential two-place buffer:
$$B_0 = in.B_1$$
$$B_1 = \overline{out}.B_0 + in.B_2$$
$$B_2 = \overline{out}.B_1$$

- First step:

$$\text{(Call)}\dfrac{\text{(Act)}\dfrac{}{in.B_1 \xrightarrow{in} B_1}}{B_0 \xrightarrow{in} B_1}$$

- Second step:

$$\text{(Call)}\dfrac{\text{(Sum}_1\text{)}\dfrac{\text{(Act)}\dfrac{}{\overline{out}.B_0 \xrightarrow{\overline{out}} B_0}}{\overline{out}.B_0 + in.B_2 \xrightarrow{\overline{out}} B_0}}{B_1 \xrightarrow{\overline{out}} B_0}$$

- Like second step (with (Sum_2)): $B_1 \xrightarrow{in} B_2$
- Like first step: $B_2 \xrightarrow{\overline{out}} B_1$

Example 1.6 (continued)

(2) Sequential two-place buffer:

$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

– First step:

$$\text{(Call)}\ \dfrac{\text{(Act)}\ \dfrac{}{in.B_1 \xrightarrow{in} B_1}}{B_0 \xrightarrow{in} B_1}$$

– Second step:

$$\text{(Call)}\ \dfrac{\text{(Sum}_1\text{)}\ \dfrac{\text{(Act)}\ \dfrac{}{\overline{out}.B_0 \xrightarrow{\overline{out}} B_0}}{\overline{out}.B_0 + in.B_2 \xrightarrow{\overline{out}} B_0}}{B_1 \xrightarrow{\overline{out}} B_0}$$

– Like second step (with (Sum_2)): $B_1 \xrightarrow{in} B_2$

– Like first step: $B_2 \xrightarrow{\overline{out}} B_1$

– Complete LTS:

$$\begin{tikzcd} {} \arrow[r] & \boxed{B_0} \arrow[r, bend left, "in"] & \boxed{B_1} \arrow[l, bend left, "\overline{out}"] \arrow[r, bend left, "in"] & \boxed{B_2} \arrow[l, bend left, "\overline{out}"] \end{tikzcd}$$

empty one entry full

Semantics of CCS V

Example 1.6 (continued)

(3) Parallel two-place buffer:

$$
\begin{aligned}
B_{\|} &= (B[f] \parallel B[g]) \setminus com \\
B &= in.\overline{out}.B
\end{aligned}
$$

$(f := [out \mapsto com], g := [in \mapsto com])$

First step:

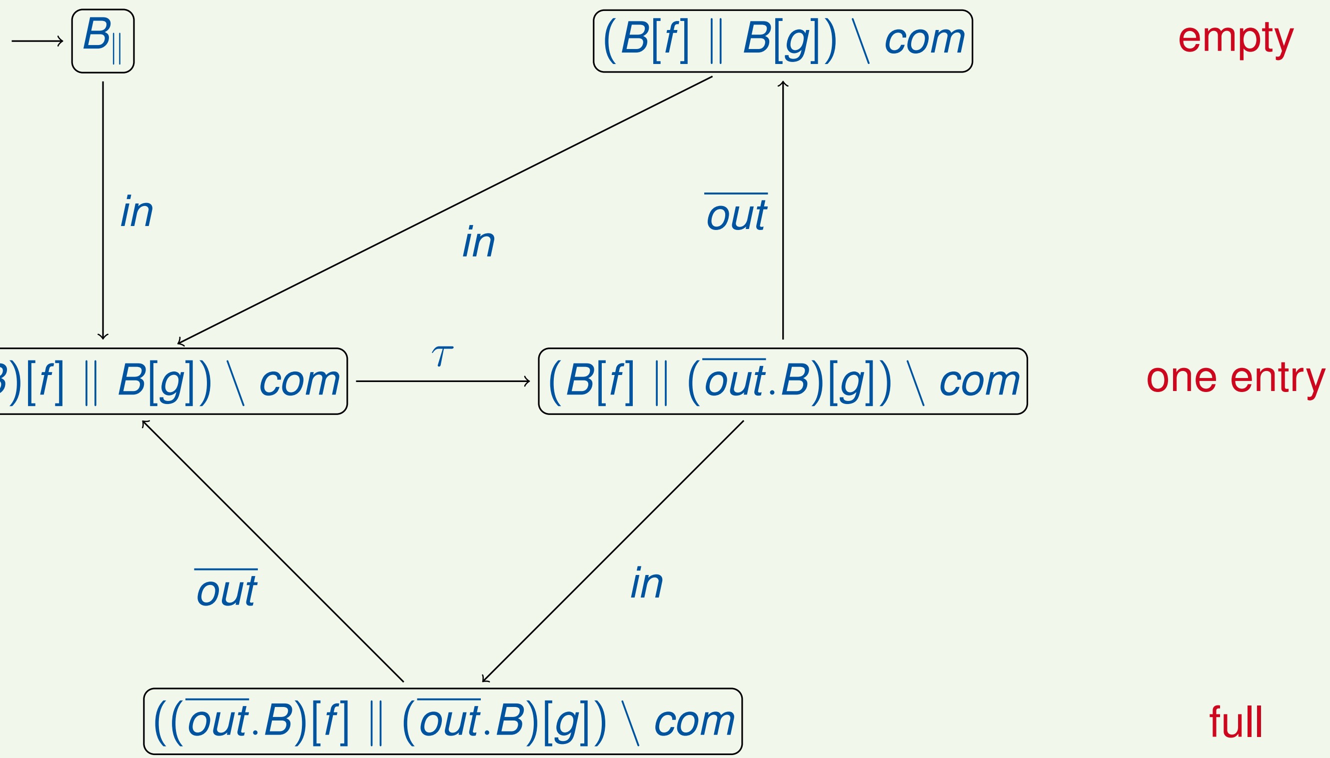
$$
\text{(Call)}\,\dfrac{\text{(Res)}\,\dfrac{\text{(Par}_1\text{)}\,\dfrac{\text{(Rel)}\,\dfrac{\text{(Call)}\,\dfrac{\text{(Act)}\,\dfrac{}{in.\overline{out}.B \xrightarrow{in} \overline{out}.B}}{B \xrightarrow{in} \overline{out}.B}}{B[f] \xrightarrow{in} (\overline{out}.B)[f]}}{B[f] \parallel B[g] \xrightarrow{in} (\overline{out}.B)[f] \parallel B[g]}}{(B[f] \parallel B[g]) \setminus com \xrightarrow{in} ((\overline{out}.B)[f] \parallel B[g]) \setminus com}}{B_{\|} \xrightarrow{in} ((\overline{out}.B)[f] \parallel B[g]) \setminus com}
$$

Semantics of CCS VI

Example 1.6 (continued)

(3) Parallel two-place buffer: $B_{\|} = (B[f] \parallel B[g]) \setminus com \quad (f := [out \mapsto com], g := [in \mapsto com])$
$B = in.\overline{out}.B$

Complete LTS:



Outline of Lecture 1

The Power of Recursive Definitions

So far: only finite state spaces – not necessarily true!

The Power of Recursive Definitions

So far: only finite state spaces – not necessarily true!

Example 1.7 (Counter)

$$C = up.(C \parallel down.\mathrm{nil})$$

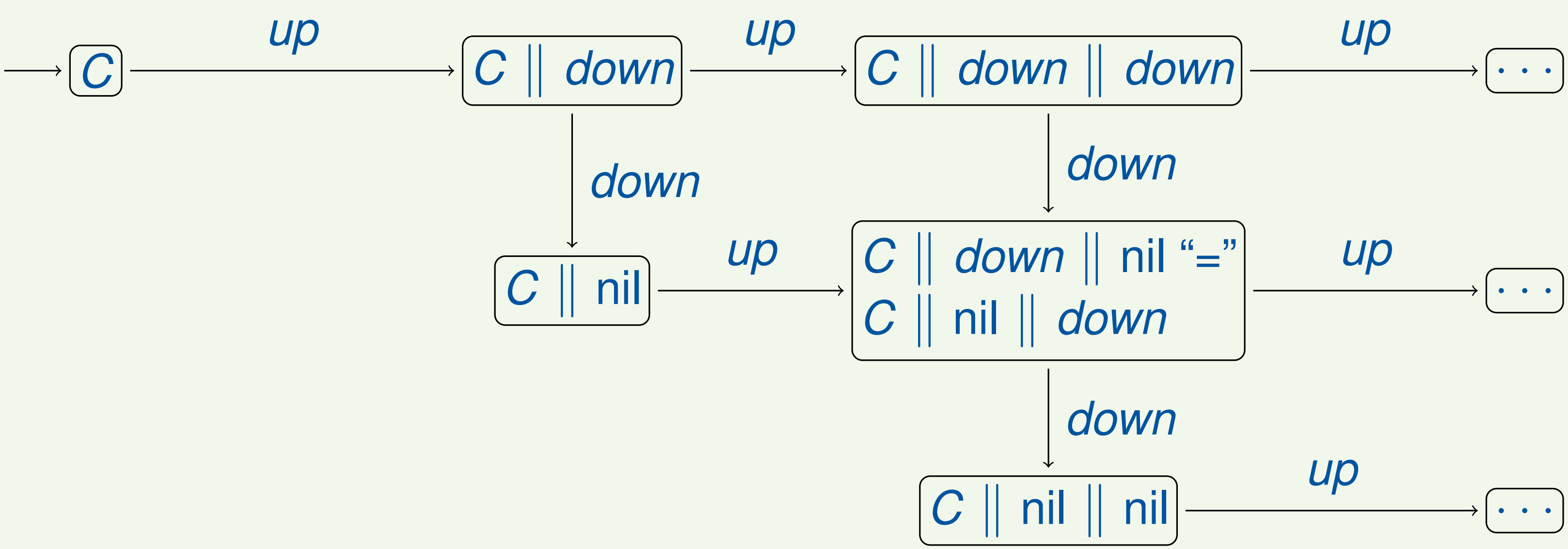
The Power of Recursive Definitions

So far: only finite state spaces – not necessarily true!

Example 1.7 (Counter)

$$C = up.(C \parallel down.\mathrm{nil})$$

gives rise to infinite LTS (abbreviating $down := down.\mathrm{nil}$):



The Power of Recursive Definitions

So far: only finite state spaces – not necessarily true!

Example 1.7 (Counter)

$$C = up.(C \parallel down.\mathrm{nil})$$

gives rise to infinite LTS (abbreviating $down := down.\mathrm{nil}$):

$$\begin{tikzcd}
{} \arrow[r] & \boxed{C} \arrow[r, "up"] & \boxed{C \parallel down} \arrow[r, "up"] \arrow[d, "down"] & \boxed{C \parallel down \parallel down} \arrow[r, "up"] \arrow[d, "down"] & \boxed{\cdots} \\
& & \boxed{C \parallel \mathrm{nil}} \arrow[r, "up"] & \boxed{\begin{array}{l} C \parallel down \parallel \mathrm{nil} \text{ “=”} \\ C \parallel \mathrm{nil} \parallel down \end{array}} \arrow[r, "up"] \arrow[d, "down"] & \boxed{\cdots} \\
& & & \boxed{C \parallel \mathrm{nil} \parallel \mathrm{nil}} \arrow[r, "up"] & \boxed{\cdots}
\end{tikzcd}$$

Sequential “specification”:

$$C_0 = up.C_1$$
$$C_n = up.C_{n+1} + down.C_{n-1} \qquad (n > 0)$$

Outline of Lecture 1

Preliminaries

Concurrency and Interaction

A Closer Look at Memory Models

A Closer Look at Reactive Systems

Overview of the Course

The Approach

Syntax of CCS

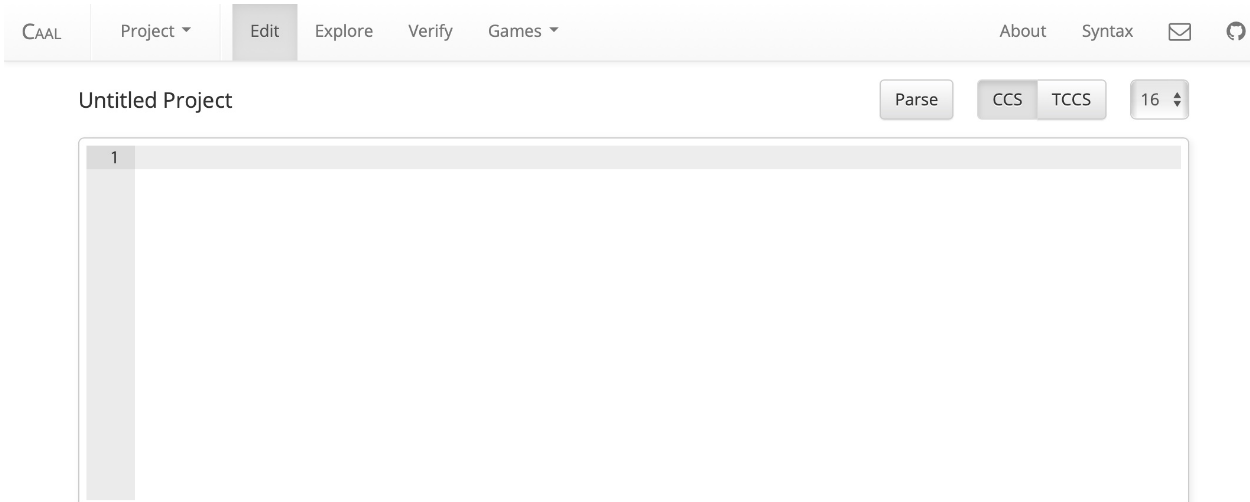
Intuitive Meaning and Examples

Formal Semantics of CCS

Infinite State Spaces

The CAAL Tool

The CAAL Tool



CAAL (Concurrency Workbench, Aalborg Edition; `https://caal.cs.aau.dk/`)

- Smart editor
- Visualisation of generated LTS
- Equivalence checking w.r.t. several bisimulation, simulation and trace equivalences
- Generation of distinguishing formulae for non-equivalent processes
- Model checking of recursive HML formulae
- (Bi)simulation and model checking games.