Concurrent systems

- Transition systems
  - suited for modeling sequential data-dependent systems
  - and for modeling sequential hardware circuits

- How about concurrent systems?
  - multi-threading
  - distributed algorithms and communication protocols

- Can we model:
  - multi-threading with shared variables?
  - synchronous communication?
  - synchronous composition of hardware?

Overview Lecture #3

⇒ Concurrency
  - The interleaving paradigm

- Communication principles
  - Shared variable "communication"
  - Handshaking
  - Synchronous communication

- Channel systems

- The state-space explosion problem

Interleaving

- Abstract from decomposition of system in components

- Actions of independent components are merged or "interleaved"
  - a single processor is available
  - on which the actions of the processes are interlocked

- No assumptions are made on the order of processes
  - possible orders for non-terminating independent processes \( P \) and \( Q \):
    \[
    \begin{align*}
    P & Q P P Q P Q Q P \ldots \\
    P & P Q P P Q P P Q \ldots \\
    P & Q P P Q P P Q \ldots 
    \end{align*}
    \]

  - assumption: there is a scheduler with an a priori unknown strategy
Interleaving

• Justification for interleaving:
  the effect of concurrently executed, independent actions \( \alpha \) and \( \beta \) equals
  the effect when \( \alpha \) and \( \beta \) are successively executed in arbitrary order

• Symbolically this is stated as:

\[
\text{Effect}(\alpha ||| \beta, \eta) = \text{Effect}((\alpha ; \beta) + (\beta ; \alpha), \eta)
\]

- \( ||| \) stands for the (binary) interleaving operator
- \( ; \) stands for sequential execution, and \( + \) for non-deterministic choice

Interleaving of transition systems

Let \( TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i) \quad i = 1, 2 \) be two transition systems

Transition system

\[ TS_1 ||| TS_2 = (S_1 \times S_2, Act_1 \uplus Act_2, \rightarrow, I_1 \times I_2, AP_1 \uplus AP_2, L) \]

where \( L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2) \) and the transition relation \( \rightarrow \) is defined by the rules:

- \( s_1 \xrightarrow{\alpha} s_1' \) and \( s_2 \xrightarrow{\alpha} s_2' \)

What are program graphs?

A program graph \( PG \) over a set \( \text{Var} \) of typed variables is a tuple

\( (\text{Loc}, \text{Act}, \text{Effect}, \rightarrow, \text{Loc}_0, g_0) \) where

- \( \text{Loc} \) is a set of locations with initial locations \( \text{Loc}_0 \subseteq \text{Loc} \)
- \( \text{Effect} : \text{Act} \times \text{Eval}(\text{Var}) \rightarrow \text{Eval}(\text{Var}) \) is the effect function
- \( \rightarrow \subseteq \text{Loc} \times \text{Cond}(\text{Var}) \times \text{Act} \times \text{Loc} \), transition relation
  Boolean conditions over \( \text{Var} \)
- \( g_0 \in \text{Cond}(\text{Var}) \) is the initial condition.
### From program graphs to transition systems

- **Basic strategy:** unfolding
  - state = location (current control) $\ell$ + data valuation $\eta$
  - initial state = initial location satisfying the initial condition $g_0$

- **Propositions and labeling**
  - propositions: "at $\ell$" and "$x \in D$" for $D \subseteq \text{dom}(x)$
  - $\langle \ell, \eta \rangle$ is labeled with "at $\ell$" and all conditions that hold in $\eta$

- if $\ell \xrightarrow{\alpha} \ell'$ and $g$ holds in $\eta$, then $\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', \text{Effect}(\alpha, \eta) \rangle$

### Interleaving of program graphs

For program graphs $PG_1$ (on $\text{Var}_1$) and $PG_2$ (on $\text{Var}_2$) **without** shared variables, i.e., $\text{Var}_1 \cap \text{Var}_2 = \varnothing$,

$$TS(PG_1) \ ||| \ TS(PG_2)$$

faithfully describes the concurrent behavior of $PG_1$ and $PG_2$.

*What if they have variables in common?*
#3: Concurrency

## Shared variable communication

\[ x := 2x \ ||\ |\ x := x + 1 \]

\[ \text{action } \alpha \quad |\quad \text{action } \beta \]

with initially \( x = 3 \)

\[
\begin{array}{c}
\downarrow \alpha \\
\downarrow \beta \\
\hline
\begin{array}{c}
\alpha \\
\beta \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\downarrow \beta \\
\downarrow \alpha \\
\hline
\begin{array}{c}
\beta \\
\alpha \\
\end{array}
\end{array}
\]

\( \langle x=6, x=4 \rangle \) is an \textit{inconsistent} state!

\( \Rightarrow \) no faithful model of the concurrent execution of \( \alpha \) and \( \beta \)

## Modeling concurrent program graphs

- If \( PG_1 \) and \( PG_2 \) share no variables:
  \[ TS(PG_1) || TS(PG_2) \]
  - interleaving of transition systems

- If \( PG_1 \) and \( PG_2 \) share some variables:
  \[ TS(PG_1 ||| PG_2) \]
  - interleaving of program graphs

- In general: \( TS(PG_1) || TS(PG_2) \neq TS(PG_1 ||| PG_2) \)

## Interleaving of program graphs

Let \( PG_i = (Loc_i, Act_i, Effect_i, \longrightarrow_i, Loc_{0,i}, g_{0,i}) \) over variables \( Var_i \).

Program graph \( PG_1 ||| PG_2 \) over \( Var_1 \cup Var_2 \) is defined by:

\[
(\text{Loc}_1 \times \text{Loc}_2, \text{Act}_1 \cup \text{Act}_2, \text{Effect}, \longrightarrow, \text{Loc}_{0,1} \times \text{Loc}_{0,2}, g_{0,1} \land g_{0,2})
\]

where \( \longrightarrow \) is defined by the inference rules:

\[
\begin{array}{c}
\ell_1 \xrightarrow{g^{\alpha}_1} \ell'_1 \\
\ell_2 \xrightarrow{g^{\alpha}_2} \ell'_2
\end{array}
\]

and \( Effect(\alpha, \eta) = Effect_i(\alpha, \eta) \) if \( \alpha \in \text{Act}_i \).

## Example

\[ x := 2x \ ||\ |\ x := x + 1 \]

\[ \text{action } \alpha \quad |\quad \text{action } \beta \]

with initially \( x = 3 \)

\[ \langle x=6, x=4 \rangle \]
#3: Concurrency

## On atomicity

\[
x := x + 1; y := 2x + 1; z := \text{y div } x \quad || \quad x := 0
\]

### Possible execution fragment:

\[
\langle x = 11 \rangle \xrightarrow{x := x + 1} \langle x = 12 \rangle \xrightarrow{y := 2x + 1} \langle x = 12 \rangle \xrightarrow{x := 0} \langle x = 0 \rangle \xrightarrow{z := y \div x} \ldots
\]

### Atomic

\[
\langle x := x + 1; y := 2x + 1; z := y \div x \rangle \quad || \quad x := 0
\]

Either the left process or the right process is completed first:

\[
\langle x = 11 \rangle \xrightarrow{x := x + 1} \langle x = 12 \rangle \xrightarrow{y := 2x + 1} \langle x = 12 \rangle \xrightarrow{z := y \div x} \langle x = 12 \rangle \xrightarrow{x := 0} \langle x = 0 \rangle
\]

---

#3: Concurrency

## Peterson’s mutual exclusion algorithm

\[
P_1\quad \text{loop forever}
\]

\[
\begin{align*}
&: \quad (* \text{ non-critical actions } *) \\
&\langle b_1 := \text{true}; x := 2 \rangle; \quad (* \text{ request } *) \\
&\text{wait until } (x = 1 \lor \neg b_2) \\
&\text{do critical section } \text{od} \\
&b_1 := \text{false} \quad (* \text{ release } *) \\
&:\quad (* \text{ non-critical actions } *)
\end{align*}
\]

\[b_i\text{ is true if and only if process } P_i\text{ is waiting or in critical section}
\]

if both processes want to enter their critical section, \( x \) decides who gets access

---

#3: Concurrency

## Banking system

Person Left behaves as follows:

```plaintext
while true {
    .......
    nc : \langle b_1, x = \text{true}, 2 \rangle
    wt : \text{wait until}(x == 1 \lor \neg b_2) \{ \\
    cs : \ldots \text{@account} \ldots \\
    b_1 = \text{false}; \\
    .......
}
```

Person Right behaves as follows:

```plaintext
while true {
    .......
    nc : \langle b_2, x = \text{true}, 1 \rangle
    wt : \text{wait until}(x == 2 \lor \neg b_1) \{ \\
    cs : \ldots \text{@account} \ldots \\
    b_2 = \text{false}; \\
    .......
}
```

Can we guarantee that only one person at a time has access to the bank account?

---

#3: Concurrency

## Program graph representation

---
Is the banking system safe?

Banking system with non-atomic assignment

Person Left behaves as follows:

```
while true {
    ....
    nc : x = 2;
    rq : b1 = true;
    wt : wait until(x == 1 || ¬b2) {
        cs : ...@account...
        b1 = false;
        ....
    }
    ....
}
```

Person Right behaves as follows:

```
while true {
    ....
    nc : x = 1;
    rq : b2 = true;
    wt : wait until(x == 2 || ¬b1) {
        cs : ...@account...
        b2 = false;
        ....
    }
    ....
}
```

On atomicity again

Assume that the location inbetween the assignments \( x := \ldots \) and \( b_i := \) true in program graph \( PG \), is called \( rq_i \). Possible state sequence:

\[
\langle nc_1, nc_2, x = 1, b_1 = false, b_2 = false \rangle
\]

\[
\langle nc_1, rq_2, x = 1, b_1 = false, b_2 = false \rangle
\]

\[
\langle rq_1, rq_2, x = 2, b_1 = false, b_2 = false \rangle
\]

\[
\langle wt_1, rq_2, x = 2, b_1 = true, b_2 = false \rangle
\]

\[
\langle cs_1, rq_2, x = 2, b_1 = true, b_2 = false \rangle
\]

\[
\langle cs_1, wt_2, x = 2, b_1 = true, b_2 = true \rangle
\]

\[
\langle cs_1, cs_2, x = 2, b_1 = true, b_2 = true \rangle
\]

Manually inspect whether two may have access to the account simultaneously: No

Parallelism and handshaking

- Concurrent processes run truly in parallel
- To obtain cooperation, some interaction mechanism is needed
- If processes are distributed there is no shared memory
  \( \Rightarrow \) Message passing
  - synchronous message passing (= handshaking)
  - asynchronous message passing (= channel communication)
Handshaking

- Concurrent processes interact by *synchronous message passing*
  - processes execute synchronized actions together
  - that is, in interaction both processes need to participate at the same time
  - the interacting processes “shake hands”

- Abstract from information that is exchanged
  - $H$ is a set of *handshake actions*
    - actions outside $H$ are independent and are interleaved
    - actions in $H$ need to be synchronized

Let $TS_i = (S_i, Act_i, I_i, AP_i, L_i), i=1,2$ and $H \subseteq Act_1 \cap Act_2$

$TS_1 \parallel_H TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, I_1 \times I_2, AP_1 \cup AP_2, L)$

where $L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$ and with $\rightarrow$ defined by:

- $s_1 \xrightarrow{\alpha_{1}} s'_1 \land s_2 \xrightarrow{\alpha_{2}} s'_2$ handshaking for $\alpha \in H$

$\parallel_{H}$ is a shorthand for $\parallel$ with $H = Act_1 \cap Act_2$
Pairwise handshaking

$TS_1 \parallel \ldots \parallel TS_n$ for $H_{i,j} = \text{Act}_i \cap \text{Act}_j$ with $H_{i,j} \cap \text{Act}_k = \emptyset$ for $k \notin \{i, j\}$

State space of $TS_1 \parallel \ldots \parallel TS_n$ is the Cartesian product of those of $TS_i$

- for $\alpha \in \text{Act} \setminus \left( \bigcup_{i \neq j} H_{i,j} \right)$ and $0 < i \leq n$:
  
  $s_i \xrightarrow{\alpha} s'_i$
  
  $(s_1, \ldots, s_i, \ldots, s_n) \xrightarrow{\alpha} (s_1, \ldots, s'_i, \ldots, s_n)$

- for $\alpha \in H_{i,j}$ and $0 < i < j \leq n$:
  
  $s_i \xrightarrow{\alpha} s'_i \land s_j \xrightarrow{\alpha} s'_j$
  
  $(s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n) \xrightarrow{\alpha} (s_1, \ldots, s'_i, \ldots, s'_j, \ldots, s_n)$