Topics in Concurrency

Jonathan Hayman

11 October 2016
Concurrency and distribution

- Computation is becoming increasingly distributed, concurrent and interactive
  - boundaries of computation becoming increasingly unclear,
  - behaviour of systems increasingly difficult to reproduce
- Problems such as how to structure and understand distributed computation, how to ensure correctness (e.g. security) of processes in an uncontrolled environment
- Concurrency theory is a broad and active field for research, but
- Present ideas of process and logics for distributed computation are too crude to address all problems...
Concurrency and distribution

- Computation is becoming increasingly distributed, concurrent and interactive
  - boundaries of computation becoming increasingly unclear,
  - behaviour of systems increasingly difficult to reproduce
- Problems such as how to structure and understand distributed computation, how to ensure correctness (e.g. security) of processes in an uncontrolled environment
- Concurrency theory is a broad and active field for research, but
- Present ideas of process and logics for distributed computation are too crude to address all problems . . . However there are attempts:
  - **topics in concurrency**
- Theories of processes, logics & model checking, security, mobility
Topics in Concurrency

- Simple parallelism and non-determinism
- Communicating processes
  - Milner’s CCS (Calculus of Communicating Systems)
  - Bisimulation
- Specification logics for processes
  - modal $\mu$-calculus
  - CTL
  - model checking
- Petri nets
  - events, causal dependence, independence
- Mobile processes
  - Higher-order processes: process passing, location
- Security protocols
  - SPL (Security Protocol Language)
  - Petri net semantics
  - Proofs of secrecy and authentication

Chapter 1 in the lecture notes revises relevant topics from Discrete Mathematics (well-founded induction and Tarski’s fixed point theorem).
While programs

Similar to $L1$ from *Semantics of Programming Languages*:

$$c:: = \text{skip} | X := a | \text{if } b \text{ then } c_1 \text{ else } c_2 | c_0;c_1 | \text{while } b \text{ do } c$$

- States $\sigma \in \Sigma$ are functions from locations to values
- Configurations: $\langle c, \sigma \rangle$ and $\sigma$
- Rules describe a single step of execution:
  
  \[
  \frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c'_0; c_1, \sigma' \rangle}
  \quad
  \frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c_1, \sigma' \rangle}
  \]

  \[
  \frac{\langle b, \sigma \rangle \rightarrow \text{true}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle}
  \]

  \[
  \vdots
  \]
Parallel commands

Syntax extended with parallel composition:

\[ c ::= \ldots \mid c_0 \parallel c_1 \]

Rules:

\[
\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c'_0 \parallel c_1, \sigma' \rangle \\
\langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0 \parallel c'_1, \sigma' \rangle
\]

(\text{\textit{+rules for termination of } } c_0, c_1)
Parallel commands

Syntax extended with parallel composition:

\[ c :: = \ldots \mid c_0 \parallel c_1 \]

Rules:

\[ \langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle \]
\[ \langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c'_0 \parallel c_1, \sigma' \rangle \]
\[ \langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle \]
\[ \langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0 \parallel c'_1, \sigma' \rangle \]

(\text{+rules for termination of } c_0, c_1)

- Parallelism \(\rightsquigarrow\) Non-determinism
- Behaviour of \(\parallel\)-commands not a partial function from states to states; when are two \(\parallel\)-commands equivalent? [Congruence?]
- Parallelism by non-deterministic interleaving
- “communication by shared variables”
Study of parallelism (or concurrency) includes study of non-determinism.
*Study of parallelism (or concurrency)*
  
  *includes*
  
  *study of non-determinism*

What about the converse?

*Can we explain parallelism (or concurrency)*

*in terms of non-determinism?*
**The language of Guarded Commands (Dijkstra)**

- Boolean expressions: \( b \)
- Arithmetic expressions: \( a \)
- Commands:

\[
c ::= \text{skip} \mid \text{abort} \mid X := a \mid c_0; c_1 \mid \text{if } gc \text{ fi} \mid \text{do } gc \text{ od}
\]

- Guarded commands:

\[
\text{gc} ::= b \rightarrow c \quad \text{guard} \\
\mid gc_0 \parallel gc_1 \quad \text{alternative}
\]
Operational semantics

- Assume given rules for evaluating Booleans and assignments.
- **Guarded commands:**

\[
\begin{align*}
\langle b, \sigma \rangle &\rightarrow true \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow \langle c, \sigma \rangle
\end{align*}
\]

\[\langle gc \rangle_{i} \rightarrow fail \quad (i = 0, 1)\]
Operational semantics

- Assume given rules for evaluating Booleans and assignments.
- **Guarded commands:**

\[
\begin{align*}
\langle b, \sigma \rangle &\rightarrow true \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow \langle c, \sigma \rangle \\
\langle gc_0, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_0 \parallel gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle
\end{align*}
\]

\[\langle gc_0 \parallel gc_1, \sigma \rangle \rightarrow \langle c, \sigma' \rangle\] introduces non-determinism
Operational semantics

- Assume given rules for evaluating Booleans and assignments.
- **Guarded commands:**

\[
\begin{align*}
\langle b, \sigma \rangle &\rightarrow true \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow \langle c, \sigma \rangle \\
\langle gc_0, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_0 \parallel gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_0 \parallel gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle b, \sigma \rangle &\rightarrow false \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow fail \quad \text{fail is a new configuration} \\
\langle gc_0, \sigma \rangle &\rightarrow fail \\
\langle gc_1, \sigma \rangle &\rightarrow fail \\
\langle gc_0 \parallel gc_1, \sigma \rangle &\rightarrow fail
\end{align*}
\]
Operational semantics

- Assume given rules for evaluating Booleans and assignments.
- **Guarded commands:**

\[
\begin{align*}
\langle b, \sigma \rangle & \rightarrow true \\
\langle b \rightarrow c, \sigma \rangle & \rightarrow \langle c, \sigma \rangle \\
\langle gc_0, \sigma \rangle & \rightarrow \langle c, \sigma' \rangle \\
\langle gc_1, \sigma \rangle & \rightarrow \langle c, \sigma' \rangle \\
\langle gc_0 \parallel gc_1, \sigma \rangle & \rightarrow \langle c, \sigma' \rangle \\
\langle b, \sigma \rangle & \rightarrow false \\
\langle b \rightarrow c, \sigma \rangle & \rightarrow fail \\
\langle gc_0, \sigma \rangle & \rightarrow fail \\
\langle gc_1, \sigma \rangle & \rightarrow fail \\
\langle gc_0 \parallel gc_1, \sigma \rangle & \rightarrow fail
\end{align*}
\]
- **Commands:**
  
  abort has no rules

- **Conditional:**

  \[
  \langle gc, \sigma \rangle \rightarrow \langle c, \sigma' \rangle \\
  \langle \text{if} \ gc \ \text{fi}, \sigma \rangle \rightarrow \langle c, \sigma' \rangle
  \]

  no rule in case \( \langle gc, \sigma \rangle \rightarrow \text{fail} \); then conditional behaves like abort

- **Loop:**

  \[
  \langle gc, \sigma \rangle \rightarrow \text{fail} \\
  \langle \text{do} \ gc \ \text{od}, \sigma \rangle \rightarrow \sigma
  \]

  \[
  \langle gc, \sigma \rangle \rightarrow \langle c, \sigma' \rangle \\
  \langle \text{do} \ gc \ \text{od}, \sigma \rangle \rightarrow \langle c; \text{do} \ gc \ \text{od}, \sigma' \rangle
  \]

  in case \( \langle gc, \sigma \rangle \rightarrow \text{fail} \), the loop behaves like skip:

  \( \langle \text{skip}, \sigma \rangle \rightarrow \sigma \)
The process

\[
d \begin{array}{c}
  b_1 \rightarrow c_1 \mid \ldots \mid b_n \rightarrow c_n
\end{array}
\]

is a form of (non-deterministically interleaved) parallel composition

\[
\begin{array}{c}
  b_1 \rightarrow c_1 \mid \ldots \mid b_n \rightarrow c_n
\end{array}
\]

in which each \(c_i\) occurs atomically (i.e. uninterruptedly) provided \(b_i\) holds each time it starts.

\[\Rightarrow\text{UNITY (Misra and Chandy)}\]

\[\Rightarrow\text{Hardware languages (Staunstrup)}\]
Examples

- Computing maximum:

  ```
  if
  X ≥ Y → MAX = X
  Y ≥ X → MAX = Y
  fi
  ```

- Euclid’s algorithm:

  ```
  do
  X > Y → X := X − Y
  Y > X → Y := Y − X
  od
  ```
Examples

- Computing maximum:

\[
\begin{align*}
\text{if} & \quad X \geq Y \rightarrow \text{MAX} = X \\
\| & \\
Y \geq X \rightarrow \text{MAX} = Y \\
\text{fi}
\end{align*}
\]

- Euclid’s algorithm:

\[
\begin{align*}
\text{do} & \quad X > Y \rightarrow X := X - Y \\
\| & \\
Y > X \rightarrow Y := Y - X \\
\text{od}
\end{align*}
\]

Have

\[
\begin{align*}
\{X = m \land Y = n \land m > 0 \land n > 0\} & \quad \text{Euclid} \\
\{X = Y = \gcd(m, n)\} & \quad \ldots \text{guarded commands support a neat Hoare-style logic}
\end{align*}
\]
Recalling:

\[ \gcd(m, n) \mid m, n \]

and

\[ \ell \mid m, n \implies \ell \mid \gcd(m, n) \]

Invariant:

\[ \gcd(m, n) = \gcd(X, Y) \]

On exiting loop, \( X = Y \).

Key properties:

\[
\begin{align*}
\gcd(m, n) &= \gcd(m - n, n) & \text{if } m > n \\
\gcd(m, n) &= \gcd(m, n - m) & \text{if } n > m \\
\gcd(m, m) &= m
\end{align*}
\]
Synchronized communication (Hoare, Milner)

Communication by “handshake”,
with possible exchange of value,
localised to process-process (CSP)
or to a channel (CCS, OCCAM)

[Abstracts away from the protocol underlying coordination/“handshake”
in the implementation]
Extending GCL with synchronization

- Allow processes to send and receive values on channels
  \( \alpha!a \) evaluate expression \( a \) and send value on channel \( \alpha \)
  \( \alpha?X \) receive value on channel \( \alpha \) and store it in \( X \)
- All interaction between parallel processes is by sending / receiving values on channels
- Communication is synchronized and only one process listening on the channel may receive the message
- Allow send and receive in commands \( c \) and in guards \( gc \): 
  \[
  \text{do } Y < 100 \land \alpha?X \rightarrow \alpha!(X \times X) \parallel Y := Y + 1 \text{ od is allowed}
  \]
- Language close to OCCAM and CSP
Extending GCL with synchronization

Transitions may now carry labels when possibility of interaction with another process.

\[
\langle \alpha?X, \sigma \rangle \xrightarrow{\alpha?n} \sigma[n/X]
\]

\[
\langle a, \sigma \rangle \rightarrow n
\]

\[
\langle \alpha!a, \sigma \rangle \xrightarrow{\alpha!n} \sigma
\]

\[
\langle c_0, \sigma \rangle \xrightarrow{\lambda} \langle c'_0, \sigma' \rangle
\]

\[
\langle c_0 \parallel c_1, \sigma \rangle \xrightarrow{\lambda} \langle c'_0 \parallel c_1, \sigma' \rangle
\]

(\(\lambda\) might be empty label) + symmetric

\[
\langle c_0, \sigma \rangle \xrightarrow{\alpha?n} \langle c'_0, \sigma' \rangle
\]

\[
\langle c_1, \sigma \rangle \xrightarrow{\alpha!n} \langle c'_1, \sigma \rangle
\]

\[
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c'_0 \parallel c'_1, \sigma' \rangle
\]

+symmetric

\[
\langle c, \sigma \rangle \xrightarrow{\lambda} \langle c', \sigma' \rangle
\]

\[
\langle c \ \alpha, \sigma \rangle \xrightarrow{\lambda} \langle c' \ \alpha, \sigma' \rangle
\]

\(\lambda \neq \alpha?n\) or \(\alpha!n\)
Examples

- forwarder:
  \[
  \begin{array}{c}
  \alpha \\
  \beta \\
  \end{array}
  \]
  \[
  \text{do } \alpha?X \rightarrow \beta!X \text{ od}
  \]

- buffer capacity 2:
  \[
  \begin{array}{c}
  \alpha \\
  \beta \\
  \gamma \\
  \end{array}
  \]
  \[
  ( \text{do } \alpha?X \rightarrow \beta!X \text{ od} \parallel \text{do } \beta?X \rightarrow \gamma!X \text{ od} ) \setminus \beta
  \]
Branching: internal vs external choice

- Extend the language, allowing Booleans to be attached to input/output actions
- Compare:

  \[
  \text{if } (true \land \alpha?X \rightarrow c_0) \equiv (true \land \beta?X \rightarrow c_1) \text{ fi}
  \]

- Not equivalent processes w.r.t. their deadlock capabilities.