Topics in Concurrency

Glynn Winskel

16 January 2020
Concurrency and distribution

- Computation has become increasingly distributed, concurrent and interactive
  - boundaries of computation becoming increasingly unclear,
  - behaviour of systems increasingly difficult to reproduce
-问题 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 unsettled . . .
Concurrency and distribution

- Computation has become 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 unsettled . . . However there are attempts:

  **topics in concurrency**
  - Theories of processes, logics & model checking, security
  - Unification through strategies in concurrent/distributed games (new)
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

- Security protocols
  - SPL (Security Protocol Language)
  - Petri net semantics
  - Proofs of secrecy and authentication

- Event structures

- Concurrent games - processes as strategies

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 $L_1$ from *Semantics of Programming Languages*:

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

- States $\sigma \in \Sigma$ are functions from locations to values
- Configurations: $\langle c, \sigma \rangle$ and $\sigma$
- Rules describe a single step of execution:

  \[
  \frac{\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c'_0; c_1, \sigma' \rangle}
  \]

  \[
  \frac{\langle c_0, \sigma \rangle \rightarrow \sigma'}{\langle c_0; c_1, \sigma \rangle \rightarrow \langle c_1, \sigma' \rangle}
  \]

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

  \[
  : \]

  \[
  : \]

  \[
  : \]
Parallel commands

Syntax extended with parallel composition:

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

Rules:

\[
\begin{align*}
\langle c_0, \sigma \rangle \rightarrow \langle c_0', \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0', \parallel c_1, \sigma' \rangle \\
\langle c_1, \sigma \rangle \rightarrow \langle c_1', \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0 \parallel c_1', \sigma' \rangle
\end{align*}
\]

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

Syntax extended with parallel composition:

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

Rules:

\[
\begin{align*}
\langle c_0, \sigma \rangle \rightarrow \langle c'_0, \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c'_0 \parallel c_1, \sigma' \rangle \\
\langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0 \parallel c'_1, \sigma' \rangle
\end{align*}
\]

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

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

\[
\langle b, \sigma \rangle \rightarrow true
\]

\[
\langle b \rightarrow c, \sigma \rangle \rightarrow \langle c, \sigma \rangle
\]
Assume given rules for evaluating Booleans and assignments.

**Guarded commands:**

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

Introduces non-determinism
Operational semantics

Assume given rules for evaluating Booleans and assignments.

**Guarded commands:**

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

fail is a new configuration
Operational semantics

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

\[
\begin{align*}
\langle b, \sigma \rangle &\rightarrow true \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow \langle c, \sigma \rangle \\
\langle gc_0, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle gc_0 [ gc_1, \sigma \rangle &\rightarrow \langle c, \sigma' \rangle \\
\langle b, \sigma \rangle &\rightarrow false \\
\langle b \rightarrow c, \sigma \rangle &\rightarrow fail \\
\langle gc_0, \sigma \rangle &\rightarrow fail \\
\langle gc_1, \sigma \rangle &\rightarrow fail \\
\langle gc_0 [ gc_1, \sigma \rangle &\rightarrow fail
\end{align*}
\]
- **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:**
  \[
  \langle gc, \sigma \rangle \rightarrow \text{fail}
  \]
  \[
  \langle \text{do gc od}, \sigma \rangle \rightarrow \sigma
  \]

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

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

  \[
  \langle \text{skip}, \sigma \rangle \rightarrow \sigma
  \]
The process

\[
\text{do } b_1 \rightarrow c_1 \boxplus \ldots \boxplus b_n \rightarrow c_n \text{ od}
\]

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

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

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

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

\[\sim \]

Hardware languages \quad \text{(Staunstrup)}
Examples

- Computing maximum:

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

- Euclid’s algorithm:

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

- Computing maximum:

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

- Euclid’s algorithm:

  \[
  \text{do} \begin{align*}
  X \swarrow Y & \rightarrow X := X - Y \\
  Y & \nearrow X \rightarrow Y := Y - X
  \end{align*}
  \text{od}
  \]

  Have

  \[
  \{X = m \land Y = n \land m > 0 \land n > 0\}
  \]

  \[
  \text{Euclid} \quad \{X = Y = \gcd(m, n)\}
  \]

  \[
  \ldots \text{guarded commands support a neat Hoare-style logic}
  \]
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 \( g \):

  \[
  \text{do } Y < 100 \land \alpha?X \rightarrow \alpha!(X \times X) \parallel Y := Y + 1 \text{ od is allowed}
  \]

- Language close to OCCAM and CSP
Extending GCL with synchronization

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

\[
\begin{align*}
\langle \alpha?X, \sigma \rangle & \xrightarrow{\alpha?n} \sigma[n/X] \\
\langle a, \sigma \rangle & \rightarrow n
\end{align*}
\]

\[
\lambda 
\]

\[
\begin{align*}
\langle c_0, \sigma \rangle & \xrightarrow{\lambda} \langle c'_0, \sigma' \rangle \\
\langle c_0 \parallel c_1, \sigma \rangle & \xrightarrow{\lambda} \langle c'_0 \parallel c_1, \sigma' \rangle
\end{align*}
\]

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

\[
\begin{align*}
\langle c_0, \sigma \rangle & \xrightarrow{\alpha?n} \langle c'_0, \sigma' \rangle \\
\langle c_1, \sigma \rangle & \xrightarrow{\alpha!n} \langle c'_1, \sigma \rangle
\end{align*}
\]

+ symmetric

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

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

\(\lambda \not\equiv \alpha?n \) or \(\alpha!n\)
Examples

- **forwarder:**

  ![Diagram of forwarder]

  \[
  \text{do } \alpha?X \rightarrow \beta!X \text{ od}
  \]

- **buffer capacity 2:**

  ![Diagram of buffer capacity 2]

  \[
  ( \text{do } \alpha?X \rightarrow \beta!X \text{ od} \\
  \parallel \text{do } \beta?X \rightarrow \gamma!X \text{ od} ) \setminus \beta
  \]
Branching: internal vs external choice

- Compare:

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

- \[
  \begin{array}{c}
  \alpha?n \\
  \alpha?n
  \end{array}
  \rightarrow
  \begin{array}{c}
  \cdot \\
  \cdot
  \end{array}
  \rightarrow
  \begin{array}{c}
  \beta?m \\
  \beta?m
  \end{array}
  \]

- \[
  \text{if } (true \rightarrow (\alpha?X; c_0)) \parallel (true \rightarrow (\beta?X; c_1)) \text{ fi}
  \]

- \[
  \begin{array}{c}
  \cdot \\
  \cdot
  \end{array}
  \rightarrow
  \begin{array}{c}
  \alpha?n \\
  \alpha?n
  \end{array}
  \rightarrow
  \begin{array}{c}
  \cdot \\
  \cdot
  \end{array}
  \rightarrow
  \begin{array}{c}
  \beta?m \\
  \beta?m
  \end{array}
  \]

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