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
- $\rightsquigarrow$ 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
- $\rightsquigarrow$ 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 μ-calculus
  - CTL
  - model checking [Concurrency workbench]
- 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*:

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

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

$$\frac{\langle c_0, \sigma \rangle \to \langle c_0', \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \to \langle c_0'; c_1, \sigma' \rangle} \qquad \frac{\langle c_0, \sigma \rangle \to \sigma'}{\langle c_0; c_1, \sigma \rangle \to \langle c_1, \sigma' \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \text{true} \qquad \langle c, \sigma \rangle \to \langle c', \sigma' \rangle}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \langle c'; \texttt{while } b \texttt{ do } c, \sigma' \rangle}$$

$$\vdots$$

Parallel commands

Syntax extended with parallel composition:

$$c ::= \ldots \mid c_0 \parallel c_1$$

Rules:

$$\frac{\langle c_0, \sigma \rangle \rightarrow \langle c_0', \sigma' \rangle}{\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0' \parallel c_1, \sigma' \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \rightarrow \langle c_1', \sigma' \rangle}{\langle c_0 \parallel c_1, \sigma \rangle \rightarrow \langle c_0 \parallel c_1', \sigma' \rangle}$$

(+rules for termination of c_0, c_1)

Parallel commands

Syntax extended with parallel composition:

$$c ::= \dots \mid c_0 \parallel c_1$$

Rules:

$$\frac{\langle c_0, \sigma\rangle \to \langle c_0', \sigma'\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \to \langle c_0' \parallel c_1, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to \langle c_1', \sigma'\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \to \langle c_0 \parallel c_1', \sigma'\rangle}$$

(+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:

$$c ::= \texttt{skip} \mid \texttt{abort} \mid X := a \mid c_0; c_1 \mid \texttt{if } gc \texttt{ fi} \mid \texttt{do } gc \texttt{ od}$$

- Guarded commands:

$$\begin{array}{lcll} gc & ::= & b \to c & \text{guard} \\ & \mid & gc_0 \,[\!]\, gc_1 & \text{alternative} \end{array}$$

Operational semantics

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

$$\frac{\langle b, \sigma \rangle \rightarrow \mathit{true}}{\langle b \rightarrow c, \sigma \rangle \rightarrow \langle c, \sigma \rangle}$$

Operational semantics

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

$$\frac{\langle b, \sigma \rangle \to \mathit{true}}{\langle b \to c, \sigma \rangle \to \langle c, \sigma \rangle}$$

$$\frac{\langle gc_0, \sigma \rangle \to \langle c, \sigma' \rangle}{\langle gc_0 \; [] \; gc_1, \sigma \rangle \to \langle c, \sigma' \rangle} \qquad \frac{\langle gc_1, \sigma \rangle \to \langle c, \sigma' \rangle}{\langle gc_0 \; [] \; gc_1, \sigma \rangle \to \langle c, \sigma' \rangle}$$

introduces non-determinism

Operational semantics

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

$$\frac{\langle b, \sigma\rangle \to \mathit{true}}{\langle b \to c, \sigma\rangle \to \langle c, \sigma\rangle}$$

$$\frac{\langle gc_0, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle gc_0 \mathbin{[\!]} gc_1, \sigma\rangle \to \langle c, \sigma'\rangle} \qquad \frac{\langle gc_1, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle gc_0 \mathbin{[\!]} gc_1, \sigma\rangle \to \langle c, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \mathit{false}}{\langle b \to c, \sigma\rangle \to \mathrm{fail}}$$

fail is a new configuration

$$\frac{\langle gc_0, \sigma\rangle \to \mathrm{fail} \qquad \langle gc_1, \sigma\rangle \to \mathrm{fail}}{\langle gc_0 \mathbin{[\!]} gc_1, \sigma\rangle \to \mathrm{fail}}$$

Operational semantics

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

$$\frac{\langle b, \sigma\rangle \to \mathit{true}}{\langle b \to c, \sigma\rangle \to \langle c, \sigma\rangle}$$

$$\frac{\langle gc_0, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle gc_0 \,[\!]\, gc_1, \sigma\rangle \to \langle c, \sigma'\rangle} \qquad \frac{\langle gc_1, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle gc_0 \,[\!]\, gc_1, \sigma\rangle \to \langle c, \sigma'\rangle}$$

$$\frac{\langle b, \sigma\rangle \to \mathit{false}}{\langle b \to c, \sigma\rangle \to \mathrm{fail}}$$

$$\frac{\langle gc_0, \sigma\rangle \to \mathrm{fail} \qquad \langle gc_1, \sigma\rangle \to \mathrm{fail}}{\langle gc_0 \,[\!]\, gc_1, \sigma\rangle \to \mathrm{fail}}$$

- **Commands:**

$$\texttt{abort}\text{ has no rules}$$

- Conditional:

$$\frac{\langle gc, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle \texttt{if } gc \texttt{ fi}, \sigma\rangle \to \langle c, \sigma'\rangle}$$

  no rule in case $\langle gc, \sigma\rangle \to \text{fail}$; then conditional behaves like `abort`

- Loop:

$$\frac{\langle gc, \sigma\rangle \to \text{fail}}{\langle \texttt{do } gc \texttt{ od}, \sigma\rangle \to \sigma}$$

$$\frac{\langle gc, \sigma\rangle \to \langle c, \sigma'\rangle}{\langle \texttt{do } gc \texttt{ od}, \sigma\rangle \to \langle c; \texttt{do } gc \texttt{ od}, \sigma'\rangle}$$

  in case $\langle gc, \sigma\rangle \to \text{fail}$, the loop behaves like `skip`:

$$\langle \texttt{skip}, \sigma\rangle \to \sigma$$

The process

$$\texttt{do}\ b_1 \rightarrow c_1\ [\!]\ \ldots\ [\!]\ b_n \rightarrow c_n\ \texttt{od}$$

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

$$\boxed{b_1 \rightarrow c_1} \parallel \ldots \parallel \boxed{b_n \rightarrow c_n}$$

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

$\rightsquigarrow$

UNITY	(Misra and Chandy)
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:

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

$$
\begin{array}{l}
\texttt{if} \\
\quad X \geq Y \rightarrow MAX = X \\
\quad [\!] \\
\quad Y \geq X \rightarrow MAX = Y \\
\texttt{fi}
\end{array}
$$

- Euclid's algorithm:

$$
\begin{array}{l}
\texttt{do} \\
\quad X > Y \rightarrow X := X - Y \\
\quad [\!] \\
\quad Y > X \rightarrow Y := Y - X \\
\texttt{od}
\end{array}
$$

Have

$$
\{X = m \wedge Y = n \wedge m > 0 \wedge n > 0\}
$$
$$
\mathit{Euclid}
$$
$$
\{X = Y = gcd(m, n)\}
$$

. . . 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:

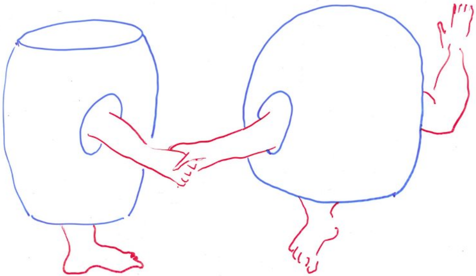
$$gcd(m, n) = gcd(X, Y)$$

On exiting loop, $X = Y$.

- Key properties:

$$\begin{aligned} gcd(m, n) &= gcd(m - n, n) && \text{if } m > n \\ gcd(m, n) &= gcd(m, n - m) && \text{if } n > m \\ gcd(m, m) &= m \end{aligned}$$

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 α
$\alpha?X$	receive value on channel α 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:

$$\texttt{do}\ \underbrace{Y < 100 \wedge \alpha?X}_{gc} \rightarrow \underbrace{\alpha!(X * X) \parallel Y := Y + 1}_{c}\ \texttt{od} \text{ is allowed}$$

- Language close to OCCAM and CSP

Extending GCL with synchronization

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

$$\frac{}{\langle \alpha?X, \sigma\rangle \xrightarrow{\alpha?n} \sigma[n/X]} \qquad \frac{\langle a, \sigma\rangle \to n}{\langle \alpha!a, \sigma\rangle \xrightarrow{\alpha!n} \sigma}$$

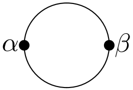
$$\frac{\langle c_0, \sigma\rangle \xrightarrow{\lambda} \langle c_0', \sigma'\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \xrightarrow{\lambda} \langle c_0' \parallel c_1, \sigma'\rangle} \quad (\lambda \text{ might be empty label}) + \text{symmetric}$$

$$\frac{\langle c_0, \sigma\rangle \xrightarrow{\alpha?n} \langle c_0', \sigma'\rangle \qquad \langle c_1, \sigma\rangle \xrightarrow{\alpha!n} \langle c_1', \sigma\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \to \langle c_0' \parallel c_1', \sigma'\rangle} + \text{symmetric}$$

$$\frac{\langle c, \sigma\rangle \xrightarrow{\lambda} \langle c', \sigma'\rangle}{\langle c \setminus \alpha, \sigma\rangle \xrightarrow{\lambda} \langle c' \setminus \alpha, \sigma'\rangle} \ \lambda \not\equiv \alpha?n \text{ or } \alpha!n$$

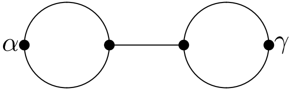
Examples

- forwarder:



$$\texttt{do}\ \alpha?X \rightarrow \beta!X\,\texttt{od}$$

- buffer capacity 2:

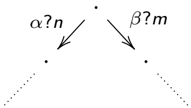


$$\begin{array}{ll} (& \texttt{do}\ \alpha?X \rightarrow \beta!X\ \texttt{od} \\ \| & \texttt{do}\ \beta?X \rightarrow \gamma!X\ \texttt{od}\ ) \setminus \beta \end{array}$$

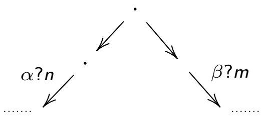
Branching: internal vs external choice

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

$$\texttt{if } (\mathit{true} \wedge \alpha?X \to c_0) \mathbin{[\!]} (\mathit{true} \wedge \beta?X \to c_1) \texttt{ fi}$$



$$\texttt{if } (\mathit{true} \to (\alpha?X; c_0)) \mathbin{[\!]} (\mathit{true} \to (\beta?X; c_1)) \texttt{ fi}$$



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