Australian National University

COMP3610/6361
Principles of Programming Languages

Peter Höfner

Sep 27, 2023

Section 19

Concurrency

Concurrency and Distribution

so far we concentrated on semantics for sequential computation
but the world is not sequential. . .

- hardware is intrinsically parallel
- multi-processor architectures
- multi-threading (perhaps on a single processor)
- networked machines

Problems

aim: languages that can be used to model computations that execute in parallel and on distributed architectures

problems

- state-spaces explosion
  with n threads, each of which can be in 2 states, the system has 2^n states
- state-spaces become complex
- computation becomes nondeterministic
- competing for access to resources may deadlock or suffer starvation
- partial failure (of some processes, of some machines in a network, of some persistent storage devices)
- communication between different environments
- partial version change
- communication between administrative regions with partial trust (or, indeed, no trust)
- protection against malicious attack
- ...

Problems

this course can only scratch the surface

concurrency theory is a broad and active field for research

Process Calculi

- Observation (1970s): computers with shared-nothing architectures communicating by sending messages to each other would be important
  [Edsger W. Dijkstra, Tony Hoare, Robin Milner, and others]
- Hoare's Communicating Sequential Processes (CSP) is an early and highly-influential language that capture a message passing form of concurrency
- many languages have built on CSP including Milner's CCS and π-calculus, Petri nets, and others

IMP – Parallel Commands

we extend our while-language that is based on aexp, bexp and com

Syntax

$$\text{com} ::= \ldots \mid \text{com} \parallel \text{com}$$

Semantics

(par1) $$\frac{\langle c_0\ ,\ s\rangle \longrightarrow \langle c_0'\ ,\ s'\rangle}{\langle c_0 \parallel c_1\ ,\ s\rangle \longrightarrow \langle c_0' \parallel c_1\ ,\ s'\rangle}$$

(par2) $$\frac{\langle c_1\ ,\ s\rangle \longrightarrow \langle c_1'\ ,\ s'\rangle}{\langle c_0 \parallel c_1\ ,\ s\rangle \longrightarrow \langle c_0 \parallel c_1'\ ,\ s'\rangle}$$

IMP – Parallel Commands

Typing

(thread)

$$\frac{\Gamma \vdash c : \mathsf{unit}}{\Gamma \vdash c : \mathsf{proc}}$$

(par)

$$\frac{\Gamma \vdash c_0 : \mathsf{proc} \qquad \Gamma \vdash c_1 : \mathsf{proc}}{\Gamma \vdash c_0 \parallel c_1 : \mathsf{proc}}$$

Parallel Composition: Design Choices

- threads do not return a value
- threads do not have an identity
- termination of a thread cannot be observed within the language
- threads are not partitioned into ‘processes’ or machines
- threads cannot be killed externally

Asynchronous Execution

- semantics allow interleavings

$$\langle l := 1 \parallel l := 2\,,\ \{l \mapsto 0\}\rangle \longrightarrow \langle \textbf{skip} \parallel l := 2\,,\ \{l \mapsto 1\}\rangle \longrightarrow \langle \textbf{skip} \parallel \textbf{skip}\,,\ \{l \mapsto 2\}\rangle$$

$$\langle l := 1 \parallel l := 2\,,\ \{l \mapsto 0\}\rangle \longrightarrow \langle l := 1 \parallel \textbf{skip}\,,\ \{l \mapsto 2\}\rangle \longrightarrow \langle \textbf{skip} \parallel \textbf{skip}\,,\ \{l \mapsto 1\}\rangle$$

- assignments and dereferencing are *atomic*

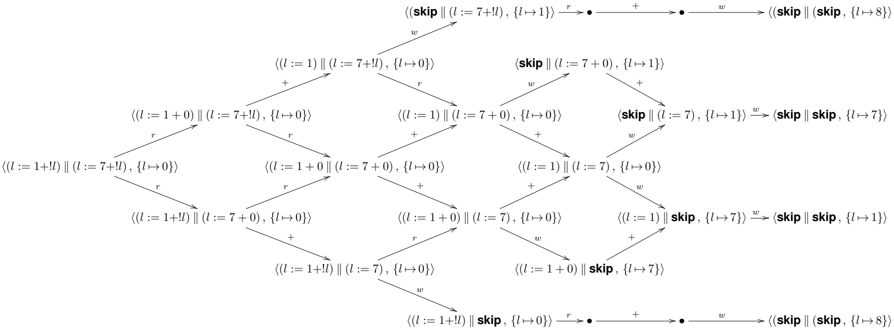
$$\langle l := N \parallel l := 2\,,\ \{l \mapsto 0\}\rangle \longrightarrow \langle \textbf{skip} \parallel l := 2\,,\ \{l \mapsto N\}\rangle \longrightarrow \langle \textbf{skip} \parallel \textbf{skip}\,,\ \{l \mapsto 2\}\rangle$$

$$\langle l := N \parallel l := 2\,,\ \{l \mapsto 0\}\rangle \longrightarrow \langle l := N \parallel \textbf{skip}\,,\ \{l \mapsto 2\}\rangle \longrightarrow \langle \textbf{skip} \parallel \textbf{skip}\,,\ \{l \mapsto N\}\rangle$$

for $N = 3498734590879238429384$.
(not something as the first word of one and the second word of the other)

Asynchronous Execution

- there interleaving in $(l := e) \parallel e'$



Morals

- combinatorial explosion
- drawing state-space diagrams only works for really tiny examples
- almost certainly the programmer does not want all those 3 outcomes to be possible
- complicated/impossible to analyse without formal methods

Parallel Commands – Nondeterminism

Semantics

(par1) $$\frac{\langle c_0\,,\,s\rangle \longrightarrow \langle c_0'\,,\,s'\rangle}{\langle c_0 \parallel c_1\,,\,s\rangle \longrightarrow \langle c_0' \parallel c_1\,,\,s'\rangle}$$

(par2) $$\frac{\langle c_1\,,\,s\rangle \longrightarrow \langle c_1'\,,\,s'\rangle}{\langle c_0 \parallel c_1\,,\,s\rangle \longrightarrow \langle c_0 \parallel c_1'\,,\,s'\rangle}$$

(+maybe rules for termination)

- study of nondeterminism
- $\parallel$ is not a partial function from state to state; big-step semantics needs adaptation
- can we achieve parallelism by nondeterministic interleaving
- communication via shared variable

Study of Parallelism (or Concurrency)

includes

Study of Nondeterminism

Dijkstra's Guarded Command Language (GCL)

- defined by Edsger Dijkstra for predicate transformer semantics
- combines programming concepts in a compact/abstract way
- simplicity allows correctness proofs
- closely related to Hoare logic

GCL – Syntax

- arithmetic expressions: aexp (as before)
- Boolean expressions: bexp (as before)
- Commands:

$$\begin{aligned} \mathsf{com} ::= {} & \textbf{skip} \mid \textbf{abort} \mid l := \mathsf{aexp} \mid \mathsf{com} \,;\, \mathsf{com} \mid \\ & \textbf{if}\ \mathsf{gc}\ \textbf{fi} \mid \textbf{do}\ \mathsf{gc}\ \textbf{od} \end{aligned}$$

- Guarded Commands:

$$\begin{aligned} \mathsf{gc} ::= {} & \mathsf{bexp} \to \mathsf{com} \mid \\ & \mathsf{gc} \mathbin{[\,]} \mathsf{gc} \end{aligned}$$

GCL – Semantics

- assume we have semantic rules for bexp and aexp (standard)
  we skip the deref-operator from now on
- assume a new configuration `fail`

Guarded Commands

(pos) $$\frac{\langle b\,,\ s\rangle \longrightarrow \langle \texttt{true}\,,\ s\rangle}{\langle b \rightarrow c\,,\ s\rangle \longrightarrow \langle c\,,\ s\rangle}$$

(neg) $$\frac{\langle b\,,\ s\rangle \longrightarrow \langle \texttt{false}\,,\ s\rangle}{\langle b \rightarrow c\,,\ s\rangle \longrightarrow \texttt{fail}}$$

(par1) $$\frac{\langle gc_0\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}{\langle gc_0 \ [\!]\ gc_1\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}$$

(par2) $$\frac{\langle gc_1\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}{\langle gc_0 \ [\!]\ gc_1\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}$$

(par3) $$\frac{\langle gc_0\,,\ s\rangle \longrightarrow \texttt{fail} \qquad \langle gc_1\,,\ s\rangle \longrightarrow \texttt{fail}}{\langle gc_0 \ [\!]\ gc_1\,,\ s\rangle \longrightarrow \texttt{fail}}$$

GCL – Semantics

Commands

- **skip** and sequencing ; as before (can drop determinacy)
- **abort** has no rules

(cond) $$\frac{\langle gc\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}{\langle \textbf{if}\ gc\ \textbf{fi}\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}$$

(loop1) $$\frac{\langle gc\,,\ s\rangle \longrightarrow \texttt{fail}}{\langle \textbf{do}\ gc\ \textbf{od}\,,\ s\rangle \longrightarrow \langle\!\langle s\rangle\!\rangle\ ^{\dagger}}$$

(loop2) $$\frac{\langle gc\,,\ s\rangle \longrightarrow \langle c\,,\ s'\rangle}{\langle \textbf{do}\ gc\ \textbf{od}\,,\ s\rangle \longrightarrow \langle c\,;\ \textbf{do}\ gc\ \textbf{od}\,,\ s'\rangle}$$

† new notation: behaves like **skip**

Processes

$$\textbf{do } b_1 \to c_1 \,[]\, \cdots \,[]\, b_n \to c_n \textbf{ od}$$

- form of (nondeterministically interleaved) parallel composition
- each c_i occurs atomically (uninterruptedly), provided b_i holds each time it starts

Some languages support/are based on GCL

- UNITY (Misra and Chandy)
- Hardware languages (Staunstrup)

GCL – Examples

- compute the maximum of x and y

if

$$x \geq y \rightarrow \max := x$$

$$[\!]$$

$$y \geq x \rightarrow \max := y$$

fi

- Euclid's algorithm

do

$$x > y \rightarrow x := x - y$$

$$[\!]$$

$$y > x \rightarrow y := y - x$$

od

GCL and Floyd-Hoare logic

guarded commands support a neat Hoare logic and decorated programs

Hoare triple for Euclid

$$\{x = m \wedge y = n \wedge m > 0 \wedge n > 0\}$$

Euclid

$$\{x = y = \mathrm{gcd}(m, n)\}$$

Proving Euclid's Algorithm Correct

- recall $\gcd(m,n)|m$, $\gcd(m,n)|n$ and

$$\ell|m,n \Rightarrow \ell|\gcd(m,n)$$

- invariant: $\gcd(m,n) = \gcd(x,y)$
- key properties:

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

Synchronised Communication

- communication by “handshake”
- possible exchange of value
  (localised to process-process (CSP) or to a channel (CCS))
- abstracts from the protocol underlying coordination
- invented by Hoare (CSP) and Milner (CCS)

Extending GCL

- 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 interactions between parallel processes is by sending / receiving values on channels
- communication is synchronised (no broadcast yet)
- allow send and receive in commands c and in guards g:

$$\textbf{do } y < 100 \wedge \alpha?x \quad \rightarrow \quad \alpha!(x \cdot x) \parallel y := y + 1 \textbf{ od}$$

Extending GCL – Semantics

transitions may carry labels when possibility of interaction

$$\frac{}{\langle \alpha?x\,,\,s\rangle \xrightarrow{\alpha?n} \langle\!\langle s + \{x \mapsto n\}\rangle\!\rangle} \qquad \frac{\langle a\,,\,s\rangle \longrightarrow \langle n\,,\,s\rangle}{\langle \alpha!a\,,\,s\rangle \xrightarrow{\alpha!n} \langle\!\langle s\rangle\!\rangle}$$

$$\frac{\langle c_0\,,\,s\rangle \xrightarrow{\lambda} \langle c_0'\,,\,s'\rangle}{\langle c_0 \parallel c_1\,,\,s\rangle \xrightarrow{\lambda} \langle c_0' \parallel c_1\,,\,s'\rangle}$$ (+ symmetric)

$$\frac{\langle c_0\,,\,s\rangle \xrightarrow{\alpha?n} \langle c_0'\,,\,s'\rangle \qquad \langle c_1\,,\,s\rangle \xrightarrow{\alpha!n} \langle c_1'\,,\,s\rangle}{\langle c_0 \parallel c_1\,,\,s\rangle \longrightarrow \langle c_0' \parallel c_1'\,,\,s'\rangle}$$ (+ symmetric)

$$\frac{\langle c\,,\,s\rangle \xrightarrow{\lambda} \langle c'\,,\,s'\rangle}{\langle c\backslash\alpha\,,\,s\rangle \xrightarrow{\lambda} \langle c'\backslash\alpha\,,\,s'\rangle}\ \lambda \notin \{\alpha?n, \alpha!n\}$$

λ may be the empty label

Examples

- forwarder:

$$\textbf{do } \alpha?x \rightarrow \beta!x \textbf{ od}$$

- buffer of capacity 2:

$$\begin{array}{l} (\quad \textbf{do } \alpha?x \rightarrow \beta!x \textbf{ od} \\ \;\; \| \textbf{ do } \beta?x \rightarrow \gamma!x \textbf{ od })\backslash\beta \end{array}$$

External vs Internal Choice

the following two processes are *not* equivalent w.r.t. deadlock capabilities

$$\textbf{if } (\texttt{true} \wedge \alpha?x \rightarrow c_0) \;[]\; (\texttt{true} \wedge \beta?x \rightarrow c_1) \textbf{ fi}$$

$$\textbf{if } (\texttt{true} \rightarrow \alpha?x \,;\, c_0) \;[]\; (\texttt{true} \rightarrow \beta?x \,;\, c_1) \textbf{ fi}$$