CSP Chapter 2: Concurrency

Andrzej Filinski
DIKU

Extreme Multiprogramming 4
November 29, 2013

Overview

- Today: deterministic concurrent composition.
  - Not an oxymoron.
  - Next time: *hidden* synchronizations $\Rightarrow$ nondeterminism.
- Main idea: proving a concurrent composition of processes *equivalent* to a sequential process (as defined last time)
  - Can't tell from the outside how a process is internally organized into subprocesses: compositionality again.
  - Also, justifies *parallel* implementations of nominally *sequential* specifications.
- Usual pattern:
  1. operators for constructing processes
  2. associated reasoning principles

  But will interleave more than last time.
- Main example: Dining Philosophers

Concurrent composition

- **Notation:** $P \parallel Q$. "P concurrently with Q". (Informally also: "P in parallel with Q".)
  - Note new color scheme for today: *left* vs. *right* processes.
- $\alpha(P \parallel Q) = (\alpha P) \cup (\alpha Q)$.
  - One of the few CSP combining forms that *doesn't* require equal alphabets.
- $P \parallel Q$ can engage in events that either
  - both P and Q are willing to engage in (at the same time), or
  - either P or Q is willing to engage in, and the other one is *manifestly unable to* (because of its alphabet).
- After every event, participating processes advance to their continuation states; non-participants are unchanged.

Examples of concurrent composition

- Consider definitions:

$$\begin{aligned} \alpha VMC &= \{coin, noise, coke, sprite\} \\ VMC &\triangleq coin \to noise \to (coke \to VMC \mid sprite \to VMC) \\ \alpha CUST &= \{coin, coke, sprite, drink\} \\ CUST &\triangleq coin \to coke \to drink \to CUST \end{aligned}$$

- Then we expect (informally):

$$\begin{aligned} VMC \parallel CUST &= coin \to noise \to coke \to drink \to (VMC \parallel CUST) \\ &= \mu X.coin \to noise \to coke \to drink \to X \end{aligned}$$

- Note: if we had $\alpha CUST = \{..., noise\}$, we would get

$$VMC \parallel CUST = coin \to STOP$$

because $CUST$ is now *able* but still *unwilling* to engage in $noise$ when VMC wants to.

Simple laws about concurrent composition

- *Commutativity:* $P \parallel Q = Q \parallel P$.
- *Associativity:* $(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$
- $\Rightarrow P_1 \parallel \cdots \parallel P_n = P_{\pi(1)} \parallel \cdots \parallel P_{\pi(n)}$ for any permutation π.
- *Neutral element:* $P \parallel RUN_{\alpha P} = P$, where
  $RUN_A : A \triangleq (v : A \rightarrow RUN_A)$
  - Actually $P \parallel RUN_A = P$ whenever $A \subseteq \alpha P$, including $A = \{\}$.
    - When RUN_A is manifestly *unable* to participate in some of P's events, it doesn't interfere with them.
  - What about when $A \nsubseteq \alpha P$?
- *Absorbing element:* $P \parallel STOP_{\alpha P} = STOP_{\alpha P}$
  - On the other hand: $P \parallel STOP_{\{\}} = P$.
- Cf. arithmetic: $x \cdot y = y \cdot x$, $x \cdot 1 = x$, $x \cdot 0 = 0$, ...

Reasoning about simple interaction

- **Beware:** for this slide *only*, we assume $\alpha P = \alpha Q = \alpha(P \parallel Q) = A$.
- Prefix agreement: $(x \to P) \parallel (x \to Q) = x \to (P \parallel Q)$
  - Both do x, then continue concurrently
- Disagreement: $(x \to P) \parallel (y \to Q) = STOP$, when $x \neq y$
  - Deadlock, even without explicit $STOP$
  - Typical pattern: $(x \to y \to P) \parallel (y \to x \to Q)$
- Compatible enumerated choices:
  $(x \to P \mid y \to Q \mid z \to R) \parallel (y \to S \mid z \to T \mid u \to U)$
  $= (y \to (Q \parallel S) \mid z \to (R \parallel T))$.
- General law for composing same-alphabet parametric choices (subsumes prefixing, enumerated choice):
  $(v : B \to P(v)) \parallel (w : C \to Q(w)) = (v : B \cap C \to (P(v) \parallel Q(v))$
  - Agreement: $B = C = \{x\}$; so $B \cap C = \{x\}$
  - Disagreement: $B = \{x\}$, $C = \{y\}$, $x \neq y$; so $B \cap C = \{\}$.

General concurrent composition

- In general: $\alpha(P \parallel Q) = (\alpha P) \cup (\alpha Q)$.
  Conventions on this slide:
  - $x \in \alpha P, x \notin \alpha Q$
  - $y \in \alpha Q, y \notin \alpha P$
  - $z \in \alpha P, z \in \alpha Q$.
- Agreement: $(z \to P) \parallel (z \to Q) = z \to (P \parallel Q)$.
  - Note: $(x \to P) \parallel (x \to Q)$ would be illegal because $x \notin \alpha Q$.
- Real disagreement: $R = (z_1 \to P) \parallel (z_2 \to Q)$, $z_1 \neq z_2$.
  - Both left and right process must participate in z_1 and z_2,
  - but they can't agree on which one,
  - so deadlock: $R = STOP$.
- Pseudo-disagreement: $R = (x \to P) \parallel (z \to Q)$
  - Only left process can participate in x,
  - while both must participate in z together,
  - so only x can happen first: $R = x \to (P \parallel (z \to Q))$

Example

- Definitions:

$$\begin{aligned} \alpha VM &= \{coin, coke, sprite\} \\ VM &\triangleq coin \rightarrow coke \rightarrow sprite \rightarrow VM \\ \alpha CUST &= \{think, coin, coke, sprite\} \\ CUST &\triangleq think \rightarrow coin \rightarrow sprite \rightarrow CUST \end{aligned}$$

- Then

$$\begin{aligned} VM \parallel CUST &\overset{def}{=} (coin \rightarrow coke \rightarrow sprite \rightarrow VM) \parallel \\ &\quad\; (think \rightarrow coin \rightarrow sprite \rightarrow CUST) \\ &\overset{PD}{=} think \rightarrow ((coin \rightarrow coke \rightarrow sprite \rightarrow VM) \parallel \\ &\qquad\qquad (coin \rightarrow sprite \rightarrow CUST)) \\ &\overset{A}{=} think \rightarrow coin \rightarrow ((coke \rightarrow sprite \rightarrow VM) \parallel \\ &\qquad\qquad\qquad (sprite \rightarrow CUST)) \\ &\overset{RD}{=} think \rightarrow coin \rightarrow STOP \end{aligned}$$

Concurrent composition (continued)

- Conventions (unchanged):
  - $x \in \alpha P, x \notin \alpha Q$; $y \in \alpha Q, y \notin \alpha P$; $z \in \alpha P, z \in \alpha Q$.
- Independent events: $R = (x \to P) \parallel (y \to Q)$
  - Only left process can participate in x,
  - only right process can participate in y,
  - so *either* event can happen first:

    $R = \big(x \to (P \parallel (y \to Q)) \mid y \to ((x \to P) \parallel Q)\big)$

    $\neq \big(x \to y \to (P \parallel Q)) \mid y \to x \to (P \parallel Q)\big)$ [in general].
- **Example:**

$$VM \triangleq coin \to coke \to noise \to STOP$$

$$CUST \triangleq think \to coin \to coke \to drink \to CUST$$

$$\begin{aligned} VM \parallel CUST = \\ think \to coin \to coke \to (&noise \to drink \to think \to STOP \\ &\mid drink \to (noise \to think \to STOP \\ &\qquad\qquad \mid think \to noise \to STOP)) \end{aligned}$$

General concurrent composition of choices

- **General law**:

$$\overbrace{(v : A \to P_1(v))}^{P} \parallel \overbrace{(v : B \to Q_1(v))}^{Q} = (v : C \to (P'(v) \parallel Q'(v)),$$

  where

  - $A \subseteq \alpha P$, $B \subseteq \alpha Q$.
  - $C = (A \cap B) \cup (A - \alpha Q) \cup (B - \alpha P)$
    - $A \cap B$: both P and Q willing
    - $A - \alpha Q$: P willing, Q unable
    - $B - \alpha P$: Q willing, P unable
  - $P'(x) = \begin{cases} P_1(x) & \text{if } x \in A \\ P & \text{if } x \notin A \end{cases}$
  - $Q'(x) = \begin{cases} Q_1(x) & \text{if } x \in B \\ Q & \text{if } x \notin B \end{cases}$

- Get all previous laws and examples as special cases.
  - Remember: with only guarded recursion, every process is equivalent to a (possibly empty) choice on its first step.

Example of general concurrent composition

- $\alpha V = \{coin, coke, sprite, noise\}$
  $V = (coin \rightarrow V1 \mid coke \rightarrow V2 \mid noise \rightarrow V3)$

  $\alpha C = \{coin, coke, sprite, think, drink\}$
  $C = (coin \rightarrow C1 \mid sprite \rightarrow C2 \mid think \rightarrow C3)$
- To compute $S = V \parallel C$, instantiate general law:

$$\begin{aligned} A &= \{coin, coke, noise\} \\ B &= \{coin, sprite, think\} \\ C &= (A \cap B) \cup (A - \alpha C) \cup (B - \alpha V) \\ &= \{coin\} \cup \{noise\} \cup \{think\} = \{coin, noise, think\} \end{aligned}$$

- Write out resulting enumerated choice:

  $\alpha S = \{coin, coke, sprite, noise, think, drink\}$
  $S = (coin \rightarrow (V1 \parallel C1) \mid noise \rightarrow (V3 \parallel C) \mid think \rightarrow (V \parallel C3))$
- Continue expanding $(V1 \parallel C1)$, $(V3 \parallel C)$, and $(V \parallel C3)$...

Parallel composition of recursive processes

Useful trick: peek inside next iteration of process

$$\begin{aligned} VM &\triangleq coin \to \overbrace{coke \to noise \to VM}^{VM'} \\ CUST &\triangleq coin \to \underbrace{coke \to drink \to CUST}_{CUST'} \end{aligned}$$

$$\begin{aligned} & VM \parallel CUST = \cdots \\ = \quad & coin \to coke \to (noise \to (\underline{VM} \parallel (drink \to CUST)) \\ & \qquad |drink \to ((noise \to VM) \parallel \underline{CUST})) \\ = \quad & coin \to coke \to (noise \to ((coin \to VM') \parallel (drink \to CUST)) \\ & \qquad |drink \to (\underline{(noise \to VM) \parallel (coin \to CUST')})) \\ = \quad & coin \to coke \to (noise \to drink \to ((\underline{coin \to VM'}) \parallel CUST) \\ & \qquad |drink \to noise \to (VM \parallel (\underline{coin \to CUST'}))) \\ = \quad & coin \to coke \to (noise \to drink \to (VM \parallel CUST) \\ & \qquad |drink \to noise \to (VM \parallel CUST)) \\ = \quad & \mu X.coin \to coke \to (noise \to drink \to X \mid drink \to noise \to X) \end{aligned}$$

POP QUIZ

- Definitions:

$$
\begin{aligned}
\alpha VMB &= \{coin, coke, sprite, noise\} \\
VMB &\triangleq coin \rightarrow (sprite \rightarrow VMB \mid coin \rightarrow coke \rightarrow STOP) \\
\alpha CUSTi &= \{coin, coke, sprite, think, drink\} \\
CUST1 &\triangleq (coin \rightarrow CUST1 \mid coke \rightarrow CUST1 \mid sprite \rightarrow CUST1) \\
CUST2 &\triangleq think \rightarrow (coin \rightarrow CUST2 \mid coke \rightarrow CUST2) \\
CUST3 &\triangleq (coin \rightarrow CUST3 \mid sprite \rightarrow drink \rightarrow CUST3)
\end{aligned}
$$

- $S1 = VMB \parallel CUST1 = ?$
- $S2 = VMB \parallel CUST2 = ?$
- $S3 = VMB \parallel CUST3 = ?$

Important reminder

- The framework so far is still *deterministic*, though processes may evolve in different ways.
- Consider:

$$\begin{aligned} VMC &\triangleq (coke \to STOP \mid noise \to VMC) \\ CCUST &\triangleq coke \to STOP \end{aligned}$$

  where $\alpha VMC = \{coke, noise\}$, $\alpha CCUST = \{coke\}$.
- All choices are subject to approval by environment: $CCUST$ does not unilaterally determine joint behavior.
- $VMC \parallel CCUST = \mu X.(coke \to STOP \mid noise \to X)$.
  Will never deadlock, as long as *its* environment only accepts *noise*s.
- Next time: nondeterminism and hiding: allow "spontaneous" internal communication between concurrent processes.
  - But all laws introduced so far will remain valid!

Traces

- **Recall:** a *trace* of a process P is a finite sequence s of events that P may engage in.
- $traces(P)$ is set of all traces of P.
  - If $t \in traces(P)$ and $s \leq t$ then $s \in traces(P)$
  - In particular, $\langle\rangle \in traces(P)$ for any P.
- Had laws:

$$\begin{aligned}
traces(v : B \rightarrow P(v)) &= \{\langle\rangle\} \cup \\
&\quad \{\langle x \rangle ^\frown s \mid x \in B \wedge s \in traces(P(x))\} \\
traces(\mu X : A.\, P(X)) &= traces(P(\mu X : A. P(X))) \\
&= \bigcup_{n \in \mathbb{N}} traces(P^n(STOP_A))
\end{aligned}$$

- Traces involved in defining...
  - satisfaction of specs: P **sat** $S(tr) \iff \forall s \in traces(P).\, S(s)$.
  - residual processes: $P \,/\, s$ for $s \in traces(P)$.

Traces of concurrent composition

- Recall informal semantics: for $P \parallel Q$ to participate in an event, must have:
  - Both P and Q willing (and hence able) to participate, *or*
  - P willing and Q unable, *or*
  - Q willing and P unable.

  Led directly to law for concurrent composition of two parametric choices.
- Equivalently (check yourselves: 9 combinations):
  - At least one of P or Q able, *and*
  - if P able then P willing, *and*
  - if Q able then Q willing
- Can be paraphrased as law about traces:

$$traces(P \parallel Q) = \{s \in (\alpha P \cup \alpha Q)^* \mid (s \upharpoonright \alpha P) \in traces(P) \wedge (s \upharpoonright \alpha Q) \in traces(Q)\}$$

Special cases

- $traces(P \parallel Q) = \{s \in (\alpha P \cup \alpha Q)^* \mid (s \restriction \alpha P) \in traces(P) \land (s \restriction \alpha Q) \in traces(Q)\}$
- If $\alpha P = \alpha Q = A$:

$$\begin{aligned} traces(P \parallel Q) &= \{s \in A^* \mid s \in traces(P) \land s \in traces(Q)\} \\ &= traces(P) \cap traces(Q) \end{aligned}$$

  I.e., processes synchronize on *every* event.
- If $\alpha P \cap \alpha Q = \{\}$:

$$traces(P \parallel Q) = \{s \mid \exists t \in traces(P), u \in traces(Q).\ s \ \textbf{interleaves}\ (t, u)\}$$

  I.e., processes synchronize on *no* event.

After

- **Recall:** $P \,/\, s$, where $s \in \mathit{traces}(P)$, is the residual process after P has engaged in s.
- Had: $(v : B \to P(v)) \,/\, (\langle x \rangle \hat{}\, s) = P(x) \,/\, s$.
  (Must necessarily have $x \in B$).
- What is $(P \parallel Q) \,/\, s$?
  - Each process P and Q has evolved according to the parts of s that affects it,
  - so $(P \parallel Q) \,/\, s = (P \,/\, (s \upharpoonright \alpha P)) \parallel (Q \,/\, (s \upharpoonright \alpha Q))$.
  - **Note:** alternative to eliminating (outermost) $\parallel$ first and then using laws about $/$ for choices.

Pictures

- Not to be confused with processes-as-automata pictures from last time.
- Each process is a box with lines labelled by events sticking out.
- Concurrent composition: put boxes next to another, connect lines with identical labels.
- **Note:** more than two processes can be connected to an event.
- **Note:** connections are undirected.
- **Note:** events cannot be hidden (yet).
- Will become much more useful later, when we look at communication networks with private, one-to-one channels.

Dining philosophers

- Classical concurrency problem. May have seen in OSM (or equivalent) already.
  - Round table with 5 seats, one fork between each pair of adjacent seats, spaghetti bowl in middle.
  - Each philosopher sits down (in his own fixed seat), picks up both forks (first left, then right), eats, puts down forks, leaves, repeats.
  - Evident possibility of deadlock when every philosopher has only picked up left fork.
    - *Not* solved by allowing philosophers to pick up forks in any order.
- Deadlock problem also relevant in CSP. Same solution.
- But correctness proof rather simpler than in other frameworks.

Original setup

- (Slightly different notation than in book)
- Philosophers modeled by processes $PHIL_0, \ldots, PHIL_4$; forks modeled by processes $FORK_0, \ldots, FORK_4$.
- $PHIL_i \triangleq i.sit \to i.get.i \to i.get.(i \oplus 1) \to i.eat \to i.drop.i \to i.drop.(i \oplus 1) \to i.leave \to PHIL_i$.
  - $n \oplus 1 = (n+1) \bmod 5$.
- $$FORK_j \triangleq (j.get.j \to j.drop.j \to FORK_j \mid (j \ominus 1).get.j \to (j \ominus 1).drop.j \to FORK_j) .$$
  - Fork can be picked up by philosopher on either side, but must be put down by the same one.
- $SYSTEM = PHIL_0 \parallel \cdots \parallel PHIL_4 \parallel FORK_0 \parallel \cdots \parallel FORK_4$.
- $SYSTEM \;/\; \langle 0.sit, 0.get.0, \ldots, 4.sit, 4.get.4 \rangle = STOP$.
  - (Each $PHIL_i$ only wants to do $i.get.(i \oplus 1)$; each $FORK_j$ only wants to do $j.drop.j$.)

Solution

- Introduce an additional *footman* process, to ensure that at most 4 philosophers are seated at any time.
- Event sets $S = \{i.sit \mid i \in 0..4\}$, $L = \{i.leave \mid i \in 0..4\}$.
- Define process $FOOT_n$ where $n \in 0..4$:

$$\alpha FOOT_n = S \cup L$$

$$\begin{aligned}
FOOT_0 &\triangleq (v : S \to FOOT_1) \\
FOOT_n &\triangleq (v : S \to FOOT_{n+1} \mid v : L \to FOOT_{n-1}) \quad (1 \leq n \leq 3) \\
FOOT_4 &\triangleq (v : L \to FOOT_3)
\end{aligned}$$

- $SAFESYS = SYSTEM \parallel FOOT_0$
  - **Note:** no modification to process definitions of philosophers or forks needed.

Liveness proof (1)

- **Def.** $\mathit{seated}(s) = \#(s \upharpoonright S) - \#(s \upharpoonright L)$
  - number of seated philosophers after trace s
- $\mathit{SAFESYS}$ **sat** $(0 \leq \mathit{seated}(tr) \leq 4)$.
  - Proof: Let $s \in \mathit{traces}(\mathit{SAFESYS})$. Then $s \upharpoonright (\alpha \mathit{FOOT}_0) = s \upharpoonright (S \cup L) \in \mathit{traces}(\mathit{FOOT}_0)$.
  - Can check that FOOT_0 **sat** $(0 \leq \mathit{seated}(tr) \leq 4)$: essentially like correctness of vending machine last time.
  - Completely independent of definitions of PHIL_i and FORK_j.
- In contrast, only have SYSTEM **sat** $(0 \leq \mathit{seated}(tr) \leq 5)$.

Liveness proof (2)

- Let $s \in traces(SAFESYS)$. To show: $SAFESYS \;/\; s \neq STOP$.
- Suppose $seated(s) \leq 3$. Then for at least one i, $PHIL_i \;/\; s = PHIL_i$; and $FOOT_0 \;/\; s \neq FOOT_4$, so event $i.sit$ can happen.
- Otherwise $seated(s) = 4$. Suppose some $PHIL_i$ has both his forks; then he is about to do $i.eat$ (no synchronization required), or $i.drop.i$ (which $FORK_i$ is also ready to participate in).
- Otherwise at most 4 forks are up, but no one has two. Suppose at most 3 forks are up. Then some $PHIL_i$ hasn't picked up any, and can do so.
- Otherwise, each philosopher has picked up his left fork, and so the one to left of the vacant seat can pick up his right fork.

The proof, abstractly

- Common way of ensuring absence of stuck states:
  - Restrict behavior of system, often *conservatively*. Here: adding footman process.
  - Identify *invariant* of restricted system. Here: at most 4 seated philosophers.
  - Prove *progress* property: a system satisfying the invariant can always take at least one more step.
  - Prove *preservation* property: every transition preserves the invariant.
  - Conclude that system will never get into a deadlocked (stuck) state.
- For those fresh (or not so fresh) from *Semantics and Types* course: sound familiar?

Event renaming

- In preparation for building larger process networks, want to build copy of process with different alphabet.
- Let $f : \alpha P \to A$ be an *injective* function $(x \neq y \Rightarrow f(x) \neq f(y))$.
- Define process $f(P)$, "P renamed by f".
  Process that engages in event $f(x)$ whenever P engages in x.
- $\alpha f(P) = f(\alpha P) = \{y \in A \mid \exists x \in \alpha P . f(x) = y\}$
- $traces(f(P)) = \{f^*(s) \mid s \in traces(P)\} = \{\langle f(x_1), \ldots, f(x_n)\rangle \mid \langle x_1, ..., x_n\rangle \in traces(P)\}$
- $f(v : B \to P(v)) = (w : f(B) \to f(P(f^{-1}(w))))$
  - That's why we need f to be invertible
- $f(P \parallel Q) = f(P) \parallel f(Q)$, ...

Example of renaming

- $VM \triangleq coin \rightarrow (coke \rightarrow VM \mid sprite \rightarrow VM)$
  $\alpha VM = \{coin, coke, sprite\}$
- Let $f(coin) = coin, f(coke) = pepsi, f(sprite) = sprite$
- $VM' \triangleq f(VM) = coin \rightarrow (pepsi \rightarrow VM' \mid sprite \rightarrow VM')$
  $\alpha VM' = \{coin, pepsi, sprite\}$

Process labeling

- Notation: $l : P = f_l(P)$, where $f_l(x) = l.x$ for all $x \in \alpha P$.
- Particularly useful for "server processes".
- Integer-state server:

$$INT_n \triangleq (get.n \to INT_n \mid set.0 \to INT_0 \mid set.1 \to INT_1 \mid \cdots)$$

- Process with multiple integer variables:
  $p : INT_0 \parallel q : INT_3 \parallel$
  $(\cdots p.set.4 \to (v : \{q.get.n \mid n \in \mathbf{Z}\} \to \ldots) \cdots)$
- Will prove quite convenient once we introduce channels and communication primitives.

Next time

- Read Chapter 3.
  - Feel free to only skim 3.7–3.8 (specifications, divergence).
  - Skip 3.9 (theory of nondeterministic processes) entirely.
- Homework problem for Chapters 1 & 2 will be out next week.
  - Nothing major, just a check that you are following the basics.
  - If you have spent the expected 5–10 hours of preparation per lecture so far, you should be fine.