CSP Chapter 1: Processes

Andrzej Filinski
DIKU

Extreme Multiprogramming 3
November 26, 2013

Overview

- Today: single, deterministic processes and events.
  - concurrency, nondeterminism, and channel-based communication next times
- Will follow book fairly closely, but with a few refinements. Compared to book, view CSP as
  - less of an abstract mathematical theory of processes
  - more of a concrete programming notation
- Two parts:
  1. Constructing processes
  2. Reasoning about processes

Processes

- A CSP *process* is an independent unit of behavior.
- Somewhat like a *function* in ML or Haskell, but executes concurrently with other processes in system.
  - Exchange of information intermixed with computation, instead of only at start and end (call/return).
- May be named or anonymous (expressed directly as a combination of simpler processes).
- The other processes in the system are called the *environment* of the process.
- Running examples of processes: vending machines, customers.
- Abstract naming: P, Q, R, ...

Events

- Events are the basic CSP construct for *synchronization* and *information exchange*.
- Considered *indivisible* and *instantaneous*, at the relevant level of abstraction.
- May require active participation (or at least acceptance) from multiple processes to happen.
- **Examples:** depositing coins, dispensing products, ...
- Event names may be atomic (e.g., "*coke*") or structured ("*coin*.10").
- Notation: events x, y, z, ...; event sets: A, B, C, ...

Alphabets

- The set of events in which a process may *in principle* participate.
- **Notation:** $\alpha P = A$, "Process P's alphabet is A".
- May be finite or infinite
- Acts as the *type* of a process.
  - static semantics: disallow combinations of incompatible processes
  - dynamic semantics: determines which processes must participate for an event to happen
- **Examples:**
  - vending machine: $\{coin, coke, sprite, noise\}$
  - customer: $\{coin, coke, sprite, drink, talk, \ldots\}$

The finished process

- **Notation**: $P = STOP_A$. (Alphabet A may be omitted when clear from context)
- The completely inactive process, refuses to participate in any events.
- Often undesirable: models *deadlock*.
- Crucial that alphabet is known.
  - Determines which events the process is *expected* to participate in (and may hence block), once we consider concurrent composition.

Prefixing

- **Notation**: $P = x \to Q$, pronounced "x, then Q"
  - $\alpha P = \alpha Q = A$
  - Requires $x \in A$
- The process that first participates in event x, and then behaves as Q.
- **Note:** can only prefix by an event, not by a general process; "$P \to Q$" is a *syntax error*.
  - Later, will define general sequencing "$P; Q$" in terms of parallel composition (conceptually!)
- **Example:** $VM = coin \to coke \to STOP_{\{coin, coke, sprite\}}$

Enumerated choice

- **Notation:** $P = (x_1 \to P_1 \mid \cdots \mid x_n \to P_n)$, $n \geq 2$
  - requires $\alpha P_1 = \cdots = \alpha P_n = A \; (= \alpha P)$
  - requires $x_1, \ldots, x_n \in A$
  - requires $x_1, \ldots, x_n$ pairwise distinct
- Process that can participate in any of events $x_1 \ldots x_n$, and then continues with chosen P_i
- Process *offers* choices ("menu"), environment *selects* one.
- **Example:** $VMC = coin \to (coke \to STOP \mid sprite \to STOP)$
- **Note:** can only choose between events, not general processes; "$((x_1 \to P_1) \mid (x_2 \to P_2))$" is a *syntax error*!
  - Later will introduce *general choice* between full processes, e.g., "$(x_1 \to P_1) \,\Box\, (x_2 \to P_2)$".

On syntax errors

- Seriously, **DON'T EVER WRITE**:

$$\text{“}((x_1 \to P_1) \mid (x_2 \to P_2))\text{”} \text{ or } \text{“}(P \mid Q)\text{”}$$

- Since there are no laws for such processes, you'll have to invent your own to make further progress.
  - And such "laws" are almost always incorrect.
  - E.g., "$(P \mid Q) \parallel R = (P \parallel R) \mid (Q \parallel R)$".
    - Doesn't hold even with $\square$ in place of $\mid$
  - "When you find yourself in a hole, quit digging!"
- By handing in "proofs" with syntactically ill-formed choices, you are signalling:
  1. That you didn't read the book (p. 9 bottom).
  2. That you skipped (or paid no attention to) this lecture.
  3. That you didn't even look at the slides.

Parametric choice

- **Notation:** $P = (v : B \to Q(v))$
  - v is a *variable* ranging over events
  - $Q(v)$ indicates that Q may depend on v.
    $Q(x)$ is the result of replacing v in Q with x.
  - $\alpha P = \alpha Q = A$, requires $B \subseteq A$.
- Process that offers a choice between all of the events in B.
- **Examples:**

$$
\begin{aligned}
VMC &= coin \to (v : \{coke, sprite\} \to STOP) \\
VMD &= (v : \{coin.10, coin.20\} \to \\
&\qquad \textbf{if } v = coin.10 \textbf{ then } coke \to STOP \\
&\qquad\qquad \textbf{else } sprite \to STOP) \\
INC &= (v : \{in.n \mid n \in \mathbb{N}\} \to out.(msg(v) + 1) \to STOP) \\
&\qquad \text{where } msg(in.n) = n
\end{aligned}
$$

- Note: simple computations and tests on variables are allowed.

Everything is a parametric choice!

By suitably choosing B and $Q(v)$, we can see deadlock, prefixing, and enumerated choice as special cases of parametric choice:

$$
\begin{aligned}
STOP_A &= (v : \{\} \to Q) \qquad (Q \text{ arbitrary}) \\
x \to P &= (v : \{x\} \to P) \\
(x_1 \to P_1 \mid \cdots \mid x_n \to P_n) &= (v : \{x_1, \ldots, x_n\} \to Q(v)), \text{where} \\
& \qquad Q(x_1) = P_1 \\
& \qquad \quad \vdots \\
& \qquad Q(x_n) = P_n
\end{aligned}
$$

POP QUIZ

What do these machines do (if anything)?

- $VMX = coin \rightarrow (coin \rightarrow coke \rightarrow STOP \mid sprite \rightarrow STOP)$
- $VMY = coin \rightarrow (v : \{coin, sprite\} \rightarrow coke \rightarrow STOP)$
- $VMZ = coin \rightarrow (v : \{coin, sprite\} \rightarrow$
  $\textbf{if } v = coin \textbf{ then } coke \rightarrow STOP \textbf{ else } STOP)$

Recursion

- Have already used *abbreviations* $X = P$, where X is a process name.
- Process definitions may also be recursive: P may refer to X.
- Can even have mutual recursion. In general, we work with a whole *process system*:

$$X_1 \triangleq P_1, \ldots, X_n \triangleq P_n$$

- **Examples:**

$$VM \triangleq coin \rightarrow coke \rightarrow VM$$

$$\begin{aligned} VMA &\triangleq coin \rightarrow VMB \\ VMB &\triangleq (coke \rightarrow VMA \mid sprite \rightarrow VMB) \end{aligned}$$

- **Note:** recursion must be *guarded*: can only recurse after engaging in at least one event. This is illegal (for now):

$$MYSTERY \triangleq MYSTERY$$

Alphabets of recursive processes

- With recursion, cannot tell alphabet just from definition;

$$VM \triangleq coin \rightarrow coke \rightarrow VM$$

might be a (partially empty) machine that also sells *sprite*s.

- Must either declare alphabet in definition:

$$VM : \{coin, coke, sprite\} \triangleq coin \rightarrow coke \rightarrow VM$$

or separately (cf. Haskell type annotations):

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

- **Remember:** knowing the declared alphabet of a process will be crucial once we get to parallel composition.

Parameterized recursive processes

- Convenient to also allow *families* of process definitions:

$$X_{v_1,\ldots,v_n} \triangleq P(v_1, \ldots, v_n)$$

- **Example:** a coke costs 3 coins:

$$\begin{aligned} VM &\triangleq VM_0 \\ VM_n &\triangleq \textbf{if } n = 3 \textbf{ then } coke \to VM_0 \textbf{ else } coin \to VM_{n+1} \end{aligned}$$

  Corresponds to 4 mutually recursive processes:

$$\begin{aligned} VM &\triangleq coin \to VM' \\ VM' &\triangleq coin \to VM'' \\ VM'' &\triangleq coin \to VM''' \\ VM''' &\triangleq coke \to VM \end{aligned}$$

- $v_1 \ldots v_n$ keep track of the *local state* of a process.

Anonymous recursive processes

- Sometimes convenient to introduce *local recursion*.
- **Notation:** $P = \mu X : A.\, Q(X)$ [or $\mu X : A \bullet Q(X)$]
  - Alphabet A may be omitted when clear from context.
- **Example:**

  $$VM \;=\; coin \to \mu X : \{coin, coke\}.\, coin \to coke \to X$$

  equivalent to global definitions:

  $$\begin{aligned} VM &\triangleq coin \to VM' \\ VM' &\triangleq coin \to coke \to VM' \,, \end{aligned}$$

  but VM' cannot be used by other processes.
- **Hint:** Avoid μ-notation, unless specifically asked for.
  - Doesn't scale well to recursive processes with parameters, mutual recursion.

Pictures (transition diagrams)

- (Not to be confused with *composition diagrams* later)
- Can draw processes like automata. Directed graph:
  - Nodes: process states
  - *Labeled* arcs: events
  - *Unlabeled* arcs: expansion of recursive definitions
- Determinism requirement: every node has either:
  1. zero or more outgoing arcs labelled with *distinct* events, or
  2. exactly one unlabelled outgoing arc.
- Guardedness requirement:
  - no all-unlabelled cycles.

Reasoning

Will work with two kinds of reasoning:

1. Equational: processes are equivalent, $P = P'$.
   - Requires $\alpha P = \alpha P'$.
   - Reasoning is independent of what P and P' are meant to do.
   - Compositionality principle: replacing equals by equals inside process expressions.
2. Relational/logical: process satisfies specification, P **sat** S.
   - Used for expressing more complicated properties than behaving exactly the same as another process.
   - Builds on notion of *traces*.
   - Compositionality principle: showing that system satisfies top-level specification because components satisfy theirs.

Laws for parametric choice

- Recall that deadlock, prefixing, and enumerated choice are all special cases of $(v : B \to P(v))$.
- **Fundamental law**:

$$(v : B \to P(v)) = (w : B' \to P'(w)) \;\Leftrightarrow\; B = B' \wedge \forall x \in B.\, P(x) = P'(x).$$

  Note: book uses $\equiv$ for $\Leftrightarrow$.
- Some immediate consequences:
  - $STOP \neq (x \to P)$ [because $B = \{\}, B' = \{x\}$]
  - $(x_1 \to P_1 \mid x_2 \to P_2) = (x_2 \to P_2 \mid x_1 \to P_1)$ [because $B = \{x_1, x_2\}$, $B' = \{x_2, x_1\} = B$, and $P(x) = P'(x)$]
  - $(x \to P) = (x \to Q) \;\Leftrightarrow\; P = Q$. ["$\Rightarrow$" direction particularly useful for showing *non-equality*]

Laws for recursion

When recursion is guarded, need just two reasoning principles:

- *Validity* of recursive definitions:
  - **Law:** If declarations include $X \triangleq P$ then $X = P$ (unfolding)
    - (completely implicit in book's notation)
  - Note: can also be used to replace P by X (folding)
- *Uniqueness* of recursive definitions:
  - **Law:** If declarations include $X \triangleq P(X)$ and $Y \triangleq Q(Y)$, and if $P(Z) = Q(Z)$ for all Z, then $X = Y$.
- Easy example: free coke

$$VM1 \triangleq coke \to VM1, \qquad VM2 \triangleq coke \to coke \to VM2$$

$$\begin{aligned} VM1 &= coke \to VM1 \quad \text{[by unfolding } VM1\text{]} \\ &= coke \to coke \to VM1 \quad \text{[by unfolding } VM1 \text{ again]} \end{aligned}$$

so $P(Z) = Q(Z) = coke \to coke \to Z$

Larger example

$$
\begin{aligned}
VM1 &\triangleq coin \to (coin \to coke \to VM1 \mid sprite \to VM1)\\[1ex]
VM2 &\triangleq coin \to VM2'\\
VM2' &\triangleq (sprite \to coin \to VM2' \mid coin \to coke \to VM2)
\end{aligned}
$$

Does $VM1 = VM2$? Let us rewrite $VM2$ using the laws:

$$
\begin{aligned}
\underline{VM2} &= coin \to \underline{VM2'}\\
&= coin \to \underline{(sprite \to coin \to VM2' \mid coin \to coke \to VM2)}\\
&= coin \to (coin \to coke \to VM2 \mid sprite \to \underline{coin \to VM2'})\\
&= coin \to (coin \to coke \to VM2 \mid sprite \to VM2)
\end{aligned}
$$

Again LHS and RHS are now identical up to the choice of process name.

POP QUIZ: Equivalences

- Consider definitions:

$$\begin{aligned} \alpha VM1 = \alpha VM2 &= \{coin, coke, sprite\} \\ VM1 &\triangleq coke \to coke \to VM1 \\ VM2 &\triangleq coke \to coke \to coke \to VM2 \end{aligned}$$

Is $VM1 = VM2$? Why or why not? How can we (dis)prove it?

- Consider these:

$$\begin{aligned} \alpha VM3 = \alpha VM4 &= \{coin, coke, sprite\} \\ VM3 &\triangleq coke \to sprite \to VM3 \\ VM4 &\triangleq sprite \to coke \to VM4 \end{aligned}$$

Is $VM3 = VM4$? Why or why not? How can we (dis)prove it?

Traces

- A *trace* of a process is any sequence of events that the process *could* be observed to engage in, when placed in a suitable environment.
- Variables s, t, u range over traces. $s = \langle x_0, ..., x_{n-1} \rangle$, $n \geq 0$
- **Note:** a single trace is always finite, but in general, a process may have arbitrarily long traces.
- Set of all possible traces of a process P is written $traces(P)$.
- **Examples:**
  - $traces(coin \to coke \to STOP) = \{\langle\rangle, \langle coin \rangle, \langle coin, coke \rangle\}$
  - $traces(\mu X.(sprite \to X \mid coke \to STOP)) =$
    $\{\langle\rangle, \langle sprite \rangle, \langle coke \rangle, \langle sprite, sprite \rangle, \langle sprite, coke \rangle,$
    $\langle sprite, sprite, sprite \rangle, \langle sprite, sprite, coke \rangle, \ldots\}$

(will see formal definition later)

Trace operations: concatenation

- **Notation:** $s\,\hat{}\,t$.
- **Def:** $\langle x_0, \ldots, x_{n-1}\rangle\,\hat{}\,\langle y_0, \ldots, y_{m-1}\rangle = \langle x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}\rangle$
- **Laws** (follow directly from definition):

$$\begin{aligned} \langle\rangle\,\hat{}\,s &= s \\ s\,\hat{}\,\langle\rangle &= s \\ (s\,\hat{}\,t)\,\hat{}\,u &= s\,\hat{}\,(t\,\hat{}\,u) \end{aligned}$$

Trace operations: length

- **Notation:** $\#s$.
- **Def:** $\#\langle x_0, \ldots, x_{n-1} \rangle = n$
- **Laws:**

$$
\begin{array}{rcl}
\#\langle\rangle & = & 0 \\
\#\langle x \rangle & = & 1 \\
\#(s \,\hat{}\, t) & = & \#s + \#t
\end{array}
$$

Trace operations: restriction

- **Notation:** $s \upharpoonright B$
- **Def. (informal):** the subsequence of events from s that belong to B.
- **Laws:**

$$
\begin{aligned}
\langle\rangle \upharpoonright B &= \langle\rangle \\
\langle x\rangle \upharpoonright B &= \langle x\rangle, \quad \text{if } x \in B \\
\langle x\rangle \upharpoonright B &= \langle\rangle, \quad \text{if } x \notin B \\
(s \frown t) \upharpoonright B &= (s \upharpoonright B) \frown (t \upharpoonright B)
\end{aligned}
$$

- Event counting: $s \downarrow x = \#(s \upharpoonright \{x\})$
- **Example:** $\langle coin, coke, coin, coke, coke\rangle \downarrow coke = 3$

Prefix ordering on traces

- **Notation:** $s \leq t$
- **Def:** $s \leq t \iff \exists u.\, s \hat{}\, u = t$
- **Laws** (*partial order* laws):
  - $s \leq s$ (reflexivity)
  - If $s \leq t$ and $t \leq u$ then $s \leq u$ (transitivity)
  - If $s \leq t$ and $t \leq s$ then $s = t$ (antisymmetry)
- **Note:** not a *total* order: can have $s \not\leq t$ and $t \not\leq s$.

Traces of processes

- Traces of various process forms:

$$
\begin{array}{rcl}
traces(STOP_A) & = & \{\langle\rangle\} \\
traces(x \to P) & = & \{\langle\rangle\} \cup \{\langle x\rangle ^\frown s \mid s \in traces(P)\} \\
traces((x \to P \mid y \to Q)) & = & \{\langle\rangle\} \cup \{\langle x\rangle ^\frown s \mid s \in traces(P)\} \\
 & & \cup \{\langle y\rangle ^\frown s \mid s \in traces(Q)\} \\
traces(v : B \to P(v)) & = & \{\langle\rangle\} \cup \{\langle x\rangle ^\frown s \mid x \in B, s \in traces(P(x))\} \\
(\, traces(X) & = & traces(P) \qquad \text{when } X \triangleq P\,)
\end{array}
$$

- **Note:** traces are prefix-closed: if $t \in traces(P)$ and $s \leq t$, then $s \in traces(P)$.
- **Hint:** To show $s \in traces(P)$, do *not* try to write out all of $traces(P)$ first.
  - Rather, let argument be guided by actual structure of s.
  - **Ex:** $\langle y, z\rangle \in traces((x \to P \mid y \to Q))$ if $\langle z\rangle \in traces(Q)$.

Trace refinement

- Will later introduce a notion of *trace refinement*:

$$P \sqsubseteq_{\mathrm{T}} Q \iff \mathit{traces}(Q) \subseteq \mathit{traces}(P)$$

- There are tools (e.g., FDR) that can quickly verify that $P \sqsubseteq_{\mathrm{T}} Q$, or find a specific counterexample.
- P and Q are called *trace-equivalent*, $P =_{\mathrm{T}} Q$, if $\mathit{traces}(P) = \mathit{traces}(Q)$.
  - I.e., if both $P \sqsubseteq_{\mathrm{T}} Q$ and $Q \sqsubseteq_{\mathrm{T}} P$.
- For *deterministic* processes only: If $P =_{\mathrm{T}} Q$, then $P = Q$.
  - With nondeterminism, this will no longer hold; will also need to look at *refusals*.
  - Even for deterministic processes, this is almost never the easiest way to establish equality of processes.

Residuation

- If $s \in traces(P)$, then $P \,/\, s$ ("P after s") is the *residual process* after having executed s.
- **Ex:** $VM \triangleq coin \to (coin \to coke \to VM \mid sprite \to VM)$

$$
\begin{aligned}
VM \,/\, \langle\rangle &= VM \\
VM \,/\, \langle coin\rangle &= (sprite \to VM \mid coin \to coke \to VM) \\
VM \,/\, \langle coin, sprite\rangle &= VM \\
VM \,/\, \langle coin, coin\rangle &= coke \to VM \\
VM \,/\, \langle coin, coin, coke\rangle &= VM
\end{aligned}
$$

- **Laws:**

$$
\begin{aligned}
P \,/\, \langle\rangle &= P \\
P \,/\, (s \,\hat{}\, t) &= (P \,/\, s) \,/\, t \\
(v : B \to P(v)) \,/\, \langle x\rangle &= P(x) \quad (\text{must have } x \in B) \\
(X \,/\, s &= P \,/\, s \quad \text{when } X \triangleq P\,)
\end{aligned}
$$

More operations on traces (see book)

- head, s_0
- tail, s'
- map, $f^*(s)$
- catenation of subsequences, $\hat{}/s$
- reversal: $\overline{s}$
- sequencing, $s;t$
- interleaving: $s \ \mathit{interleaves}\ (t, u)$

Specifications

- A *specification* of a process is an assertion about all its traces.
- **Notation:** P **sat** S, where S is a logical formula involving the special variable tr.
- **Def.** P **sat** $S(tr) \equiv \forall s \in traces(P).S(s)$
- Immediate consequences of definition:
  - **Law:** if $P = Q$, and Q **sat** S, then P **sat** S. [Compositionality]
  - **Law:** If P **sat** S, and P **sat** T, then P **sat** $S \wedge T$. [Conjunction]
  - **Law:** If P **sat** S, and $S \Rightarrow T$, then P **sat** T. [Implication]

Proof rules, cf. def of $traces(P)$

- **Deadlock**: $STOP$ **sat** $(tr = \langle\rangle)$.
- **Prefix**:
  If P **sat** $S(tr)$,
  then $(x \to P)$ **sat** $(tr = \langle\rangle \vee \exists s. tr = \langle x\rangle \hat{\ } s \wedge S(s))$.
- **Enumerated choice** (here for $n = 2$):
  If P **sat** $S(tr)$ and Q **sat** $T(tr)$,
  then $(x \to P \mid y \to Q)$ **sat**
  $$(tr = \langle\rangle \vee \exists s. (tr = \langle x\rangle \hat{\ } s \wedge S(s)) \vee (tr = \langle y\rangle \hat{\ } s \wedge T(s)))$$
- **Parametric choice**:
  If for all $x \in B$, $P(x)$ **sat** $S(tr, x)$,
  then $(v : B \to P(v))$ **sat**
  $$(tr = \langle\rangle \vee \exists x, s.\ tr = \langle x\rangle \hat{\ } s \wedge x \in B \wedge S(s, x)).$$
- **Recursion**:
  If $STOP_A$ **sat** S and for all Z, $(Z$ **sat** $S) \Rightarrow (P(Z)$ **sat** $S)$,
  then X **sat** S, where $X : A \triangleq P(X)$.

A satisfaction proof (1)

- Let $VM : \{coin, coke\} \triangleq coin \to coke \to VM$
  - I.e., $VM \triangleq P(VM)$, where $P(Z) = coin \to coke \to Z$.
- Want to show: in any trace of VM, the number of *coke* events is no greater than the number of *coin* events
  - Need not be equal: last *coke* may not have occurred yet
- Take $S(tr) \equiv (tr \downarrow coke \leq tr \downarrow coin)$
- To show: VM **sat** $S(tr)$, i.e., VM **sat** $(tr \downarrow coke \leq tr \downarrow coin)$.
- Recursion law base: Show $STOP$ **sat** $(tr \downarrow coke \leq tr \downarrow coin)$.
  - $STOP$ **sat** $tr = \langle\rangle$, by Deadlock law.
  - Check: $tr = \langle\rangle \Rightarrow tr \downarrow coke \leq tr \downarrow coin$.
    - OK, since $\langle\rangle \downarrow coke = \langle\rangle \downarrow coin = 0$.
  - So $STOP$ **sat** $(tr \downarrow coke \leq tr \downarrow coin)$ by Implication.

A satisfaction proof (2)

- Inductive step: assume Z **sat** $tr \downarrow coke \leq tr \downarrow coin$; show $(coin \rightarrow coke \rightarrow Z)$ **sat** $tr \downarrow coke \leq tr \downarrow coin$.
  - By assumption on Z and Prefix, we have: $(coke \rightarrow Z)$ **sat** $(tr = \langle\rangle \vee \exists s. tr = \langle coke \rangle ^\frown s \wedge s \downarrow coke \leq s \downarrow coin)$.
  - Let $S'(tr) \equiv (tr \downarrow coke \leq 1 + tr \downarrow coin)$
  - Check: $(tr = \langle\rangle \vee \exists s. tr = \langle coke \rangle ^\frown s \wedge s \downarrow coke \leq s \downarrow coin) \Rightarrow S'(tr)$
    - If $tr = \langle\rangle$, then: $\langle\rangle \downarrow coke = 0 \leq 1 = 1 + \langle\rangle \downarrow coin$.
    - If $tr = \langle coke \rangle ^\frown s$ and $s \downarrow coke \leq s \downarrow coin$, then: $tr \downarrow coke = (\langle coke \rangle ^\frown s) \downarrow coke = 1 + s \downarrow coke \leq 1 + s \downarrow coin = 1 + (\langle coke \rangle ^\frown s) \downarrow coin = 1 + tr \downarrow coin$.
  - Hence $(coke \rightarrow Z)$ **sat** $S'(tr)$, by Implication.
  - Analogously, by Prefix again: $coin \rightarrow (coke \rightarrow Z)$ **sat** $(tr = \langle\rangle \vee \exists s. tr = \langle coin \rangle ^\frown s \wedge S'(tr))$.
  - Check: $(tr = \langle\rangle \vee \exists s. tr = \langle coin \rangle ^\frown s \wedge S'(tr)) \Rightarrow S(tr)$.
  - Hence $(coin \rightarrow coke \rightarrow Z)$ **sat** $S(tr)$, qed.

Assessment of satisfaction

- P **sat** S is a *safety* property: no matter what environment does, P will never engage in sequence of events that would violate S.
- Like *partial correctness* in Hoare logic: the program will not return incorrect results (but may not return at all).
  - $STOP$ **sat** S for any specification S that is realizable at all. If P **sat** S for some P, then also $STOP$ **sat** S.
- Also need *liveness* property: process will never deadlock, i.e., $P \,/\, s \neq STOP$ for all $s \in traces(P)$.
- For now, trivial: if no occurrence of $STOP$, and all recursion guarded, any process can always engage in at least one event.
  - Once we add concurrency, will need to think harder.

For next time

- Read Chapter 1 again, if necessary.
- Read Chapter 2. May again skip Implementation sections and Section 2.8 on theory. Be sure to work through the examples.