Overview of Lecture 5

- Concurrency Workbench overview
- CW Language - CCS
- Example specifications
- μ-calculus
- Various approaches to verification
- A "larger" example - a simple protocol

Lecture 6 - using CW - syntax, semantics and some verification algorithms.

Concurrency Workbench

- Specify: a set of (communicating) concurrent processes (using CCS or SCCS)
- Use various verification methods to check that the processes meet their specification.
- The system is designed to be easy to extend (so CW in Manual $\neq$ CW in Paper)

Concurrency Workbench - Design

Design of CW - 3 layers
- First layer manages interaction with the user and contains the basic definition of process semantics in terms of *labeled transition graphs*
- Second layer provides transformations that may be applied to transition graphs (so we can change the semantic model of processes under consideration)
- Third layer includes basic algorithms to establish whether the process meets its specification. Depending on the verification method used, a specification may either be another process (describing the desired behavior) or a formula in a modal logic expressing a relevant property.

Overview of CCS

Processes are called *agents*, built from a set of *actions*. Actions can be:
- Observable (or *communication*), marked by letters a, b, etc., and
- Unobservable (or silent), marked by τ.

Observable actions:
- a, b, ... - input actions
- $\bar{a}$, $\bar{b}$, ... - output actions

Input action a and output action $\bar{a}$ are *complimentary*, reflecting input and output on the "port" a (used to represent synchronization).

Some standard operators

Nil - Terminated process

$\perp$ - Undefined process. Its behavior is unknown ("don't care").

$a.P$ - Process performs action a and then acts exactly like P.

More on that:

$\stackrel{a}{\rightarrow}$ - transition relation.

$p \stackrel{a}{\rightarrow} p\prime$ holds when p can evolve into $p\prime$ by performing action a. $p\prime$ is called an *a-derivative* of p.

$a.p \stackrel{a}{\rightarrow} p$ holds for any p.

Standard Operators (Cont'd)

$+$ - Choice. $p+q$ - either p or q will get performed. $p+q \stackrel{a}{\rightarrow} p\prime$ if either $p \stackrel{a}{\rightarrow} p\prime$ or $q \stackrel{a}{\rightarrow} p\prime$.

$|$ - Parallel composition. The agent $p \mid q$ behaves like the "interleaving" of p and q with the possibility of complementary actions synchronizing to produce a τ action.

Example:

$\backslash L$ - Restriction on (finite) set L of actions. $p\backslash L$ behaves like p with the exception that no actions in L are allowed.

$[f]$ - Relabeling of f, which maps actions to actions. $p[f]$ behaves like p with actions renamed by function f.

CCS Operators - Formal Semantics

$$
\begin{array}{rcl}
a.p \stackrel{a}{\rightarrow} p & & \\
p \stackrel{a}{\rightarrow} p\prime & \Rightarrow & p+q \stackrel{a}{\rightarrow} p\prime \\
q \stackrel{a}{\rightarrow} q\prime & \Rightarrow & p+q \stackrel{a}{\rightarrow} q\prime \\
p \stackrel{a}{\rightarrow} p\prime & \Rightarrow & p|q \stackrel{a}{\rightarrow} p\prime|q \\
q \stackrel{a}{\rightarrow} q\prime & \Rightarrow & p|q \stackrel{a}{\rightarrow} p|q\prime \\
p \stackrel{a}{\rightarrow} p\prime, q \stackrel{\bar{a}}{\rightarrow} q\prime & \Rightarrow & p|q \stackrel{\tau}{\rightarrow} p\prime|q\prime \\
p \stackrel{a}{\rightarrow} p\prime,\ a, \bar{a} \notin L & \Rightarrow & p\backslash L \stackrel{a}{\rightarrow} p\prime\backslash L \\
p \stackrel{a}{\rightarrow} p\prime & \Rightarrow & p[f] \stackrel{f(a)}{\rightarrow} p\prime[f] \\
p_A \stackrel{a}{\rightarrow} p\prime & \Rightarrow & A \stackrel{a}{\rightarrow} p\prime
\end{array}
$$

Here p_A is the agent expression bound to identifier A.

Specification examples - Buffers

BUF_n - a buffer of capacity n.

$\mathsf{BUF}_n = \mathsf{BUF}_n^0$
$\mathsf{BUF}_n^0 = \mathit{in}.\mathsf{BUF}_n^1$
$\mathsf{BUF}_n^i = \mathit{in}.\mathsf{BUF}_n^{i+1} + \overline{\mathit{out}}.\mathsf{BUF}_n^{i-1}$
for $i = 1, \ldots, n-1$
$\mathsf{BUF}_n^n = \overline{\mathit{out}}.\mathsf{BUF}_n^{n-1}$

Transition graph for BUF_n.

Specification examples - Buffers

CBUFn - a compositional buffer of capacity n.

$$\mathrm{CBUF}_n = (\mathrm{BUF}_1[x_1/\mathit{out}] \mid \underbrace{... \mid \mathrm{BUF}_1[x_i/\mathit{in}, x_{i+1}/\mathit{out}] \mid ...}_{i=1,...,n-2} \mathrm{BUF}_1[x_{n-1}/\mathit{in}]) \backslash \{x_1, ..., x_{n-1}\}$$

Transition graph for CBUF_2.

Notion of Observation

Transition graphs make a transition on every time "tick" (even if it is just τ. If timing is removed, we might be interested in just *observable* transitions.

Definition:

- $p \stackrel{\epsilon}{\Rightarrow} p\prime$ iff $p \stackrel{\tau *}{\rightarrow} p\prime$

(transitive and reflexive closure of $\stackrel{\tau}{\rightarrow}$)

- $p \stackrel{a}{\Rightarrow} p\prime$ iff $p \stackrel{\epsilon}{\Rightarrow}\stackrel{a}{\Rightarrow}\stackrel{\epsilon}{\Rightarrow}$

(relational products of $\stackrel{\epsilon}{\Rightarrow}$ and $\stackrel{a}{\rightarrow}$)

Can compute observation graphs, which takes $O(n_3)$, where n - number of nodes in the graph.

Observation Graph for CBUF_2

For clarity, ϵ-loops - one self-looping edge from each node - are omitted from the following graph.

Figure 5, page 8

μ-calculus

Specifications can be written in a modal logic based on the *propositional* μ-calculus.

Syntax of formulas:

$$\begin{array}{lll} \Phi & ::= & tt \mid ff \mid X \\ & \mid & \neg\Phi \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \Phi \Rightarrow \Phi \\ & \mid & < a > \Phi \mid [a]\Phi \mid < . > \Phi \mid [.]\Phi \\ & \mid & B \textit{ arg-list} \\ & \mid & \nu X.\Phi \mid \mu X.\Phi \end{array}$$

X ranges over variables,
a - over actions,
B - over user-defined macro identifiers,
arg-list - over lists of actions and/or formulas that B requires in order to produce a proposition,
tt and ff hold on every node and no node, respectively.

Semantics of μ-calculus formulas

Constructors $< a >$, $[a]$, $< . >$ and $[.]$ - to reason about edges leaving a node.

A node n satisfies:
- $< a > \Phi$ if it has an a-derivative satisfying Φ
- $[a]\Phi$ if *all* of its a-derivatives satisfy Φ

In the case that n has no a-derivatives, n trivially satisfies $[a]\Phi$

. acts like a "wild-card" action in $[.]$, $< . >$. n satisfies:
- $< . > \Phi$ if it satisfies $< a > \Phi$ for some a
- $[.]\Phi$ if it satisfies $[a]\Phi$ for all a

Semantics of μ-calculus formulas (cont'd)

Formulas of type $\nu X.\Phi$ and $\mu X.\Phi$ are recursive formulas, representing the greatest- and least-fixpoints, respectively.
- $\nu X.\Phi = \bigwedge_{i=0}^{\infty} \Phi_i$, where
  Φ_0 is tt and
  $\Phi_{i+1} = \Phi[\Phi_i/X]$ (substitute Φ_i for all free occurrences of X in Φ)
- $\mu X.\Phi = \bigvee_{i=0}^{\infty} \hat{\Phi}$, where
  Φ_0 is ff and
  $\Phi_{\hat{i+1}} = \Phi[\hat{\Phi_i}/X]$

Restriction: Φ should be such that any free occurrences of X appear positively.

μ-calculus and CTL

Formulas in general are unintuitive and difficult to understand. But using macros facility, they can be "coded up" into better-understood operators like CTL (its logic is a subset of μ-calculus). For example,

$$\begin{aligned}
AG\Phi &= \nu X.(\Phi \wedge [.]X) \\
AF\Phi &= \mu X.(\Phi \vee (< . > tt \wedge [.]X)) \\
AU1\Phi\Psi &= \nu X.(\Phi \vee (\Psi \wedge [.]X)) \\
AU2\Phi\Psi &= \mu X.(\Phi \vee (\Psi \wedge < . > tt \wedge [.]X))
\end{aligned}$$

You can do similar encoding for Assignment 2.

Verification example

Want to prove that CBUF$_n$, for a particular n, is deadlock free.
- Define a macro *Deadlock* $= \neg . < . > tt$ (true in states that cannot perform any actions)
- Using model-checker, check $AG\neg$*Deadlock*

Want to prove liveness property - buffer will eventually get engaged in an *in* or an $\overline{out}$
- Check $(AG((AF < in > tt) \vee (AF < \overline{out} > tt))$

Equivalence Checking

Idea - node matching. Two transition graphs are equivalent if their nodes can be matched such that
1. two matched nodes have compatible information fields
2. if two nodes are matched and one has an a-derivative, then the other must have a matching a-derivative
3. the root nodes of two transition graphs are matched.

Equivalence Checking - formal definition

Let G_1 and G_2 be transition graphs with node sets N_1 and N_2, respectively. Let $N = N_1 \cup N_2$, and let $\mathcal{C} \subseteq N \times N$ be an equivalence relation reflecting a notion of "compatibility" between information fields. A *$\mathcal{C}$-bisimulation* on G_1 and G_2 is a relation $\mathcal{R} \subseteq N \times N$ such that $< m, n > \in \mathcal{R}$ implies that:
1. if $m \xrightarrow{a} m\prime$ then $\exists n\prime : n \xrightarrow{a} n\prime$ and $< m\prime, n\prime > \in \mathcal{R}$, and
2. if $n \xrightarrow{a} n\prime$ then $\exists m\prime : m \xrightarrow{a} m\prime$ and $< m\prime, n\prime > \in \mathcal{R}$, and
3. $< m, n > \in \mathcal{C}$

If root nodes can be related by $\mathcal{C}$-bisimulation, then two transition graphs are $\mathcal{C}$-equivalent.

Equivalence Checking - Cont'd

Many equivalences are instances of $\mathcal{C}$-equivalence combined with graph transformations. For example, *observation equivalence* corresponds to equivalence on observation graphs where $\mathcal{C}$ is replaced by $U = N \times N$.

Similarly, can define *testing equivalence* for acceptance graphs (see paper).

BUF$_n$ and CBUF$_n$ are observationally equivalent for each n. Notation:

$$\text{BUF}_n \approx \text{CBUF}_n, \forall n$$

Preorder Checking

A process A is "more defined than" a process B if A has the same behavior as B except for the holes in B. The preorder algorithm determines if a process is more defined than its specification. One transition graph is less than another if the states of first can be matched to the second such that:
1. the information field of the "lesser" node must be "less" than the "greater".
2. if the "greater" node has "valid" a-transitions, then each a-transitions of the "lesser" must be matched by some a-transitions of the "greater".
3. if the "lesser" node has "viable" a-transitions, then each transition of the "greater" must be matched by some a-transition of the "lesser".
4. start state of the "lesser" and the "greater" must be matched.

Preorder Checking

Weak divergence preorder, $\underset{\approx}{\sqsubset}$, is obtained from the observation graph where
- "viable" holds for all nodes
- "valid" stands for $\{n \mid n \Downarrow a\}$. $n \Downarrow a$ holds in n is not globally divergent and cannot be triggered by means of an a-action to reach a globally divergent state.

This interpretation is based upon regarding divergent states as being *underspecified*. So, $\perp$ allows any process as a correct implementation. Preorder $\underset{\approx}{\sqsubset}$ coincides with $\approx$ for complete specifications.
Actually, when left-hand side process is completely specified, then so is the right-hand side process.

Other preorders can be defined similarly (see paper).

A Simple Protocol

Service specification of the protocol requires that any message sent must be received before a second message may be sent:

$$SRV = s.\overline{r}.\mathrm{SRV}$$

Graph:

Protocol specification - two processes, a sender and a receiver, and a medium connecting them.

Protocol Design

Create the following (logical) design:

Define:

$$\begin{array}{rcl} SND & = & s.\overline{from}.ack_{to}.SND \\ MDM & = & from.\overline{to}.MDM + ack_{from}.\overline{ack_{to}}.MDM \\ RCV & = & to.\overline{r}.\overline{ack_{from}}.RCV \\ PROT & = & SND \mid MDM \mid RCV \\ & & \backslash\{from, to, ack_{from}, ack_{to}\} \end{array}$$

Restriction operator means that these actions are internal.

- Can show $PROT \approx SRV$.

Another Protocol Design

Produce a partial definition, reflecting the fact that there may be different implementations for the medium still leading to a correct overall implementation of the service specification.

$$\begin{array}{rcl} PM & = & from.(\overline{to}.PM + ack_{from}.\perp)+ \\ & & ack_{from}.(\overline{ack_{to}}.PM + from.\perp) \\ P\_PROT & = & SND \mid PM \mid RCV \\ & & \backslash\{from, to, ack_{from}, ack_{to}\} \end{array}$$

Now define an implementation, consisting of two one-piece buffers, running in parallel: one for messages, one for acknowledgments.

$$\begin{array}{rcl} NM & = & MB \mid AB \\ MB & = & from.\overline{to}.MB \\ AB & = & ack_{from}.\overline{ack_{to}}.AB \\ N\_PROT & = & SND \mid NM \mid RCV \\ & & \backslash\{from, to, ack_{from}, ack_{to}\} \end{array}$$

Verification for this Example

- Can show that $N\_PROT \approx SRV$:
  - $PP \approx SRV$
  - $PM \stackrel{\sqsubset}{\approx} NM$
  - PP never reaches an underspecified state

(via model-checking)

Therefore, $N\_PROT \approx PP$ and hence $N\_PROT \approx SRV$

Model-checking

Define the following macros:

$$
\begin{array}{rcl}
AG\Phi & = & \nu X.(\Phi \wedge [.]X) \\
\text{Can } \Phi & = & \mu X.(X. < \Phi > tt \mid < \tau > X) \\
\text{Can't } \Phi & = & \neg \text{Can}
\end{array}
$$

Now, check:

- $S_1 = AG((\text{Can } s) \mid (\text{Can} \bar{r}))$

either a s or a $\bar{r}$ can always happen

- $S_2 = AG([s](\text{Can } \bar{r}) \& [\bar{r}](\text{Can } s))$

after a s, a process can $\bar{r}$ and vice versa

- $S_3 = AG([s](\text{Can't } s) \& [\bar{r}](\text{Can't } \bar{r}))$

two consecutive s's or $\bar{r}$'s cannot happen

- $S_4 = \text{Can } s$

s must eventually be possible

food for slide eater

food for slide eater