Lecture 6

To run CWB:

From command line using

```
cwb
```

or from emacs (see `man cwb` for installation instructions).

Examples of CWB specifications:

```
/local/share/cwb/examples/ccs
```


CWB Syntax

Identifiers - `(A-Z)(A-Z, a-z, 0-9, ?, !, _, ', `, -, #)*`
Actions - `['](a-z)(A-Z, a-z, 0-9, ?, !, _, ', `, -, #)*`

- Action τ is represented as `tau`
- Inverse actions like $\bar{a}$ are represented as `'a`
- Constant 0 is represented as `0`
- Constant `@` represents agent $\perp$ (divergence)

The rest is identical to CCS. Operations include action prefixing, summation, parallel composition, restriction, relabeling.

Example Session

```
eddie% cwb
Edinburgh Concurrency Workbench, version 7.0,
Fri Oct  6 11:36:58 BST 1995
Command: agent Cell = a.'b.Cell;
Command: agent C0 = Cell[c/b];
Command: agent C1 = Cell[c/a,d/b];
Command: agent C2 = Cell[d/a];
Command: agent Buff3 = (C0 | C1 | C2)\{c,d};
Command: agent Spec = a.Spec';
Command: agent Spec' = 'b.Spec + a.Spec'';
Command: agent Spec'' = 'b.Spec' + a.'b.Spec'';
Command: save "spec1";
Command: eq (Buff3, Spec);
true
Command: quit
eddie%
```


Environments

- CWB has several separate environments.
- All bindings are dynamic:

```
agent Cell = a.'b.Cell;
agent Cell' = a.Cell;
agent Cell = c.'b.Cell;
```

- Environments are: agents, action sets, and propositions.
- Identifiers do not clash between environments.

```
set Cell = {c, d};
agent Buff3 = (C0 | C1 | C2)\Cell;
print;
** Agents **
...
agent Cell = a.'b.Cell
** Action Sets **
set Cell = {c, d}
```

Another Example Session

```
eddie% cwb
Edinburgh Concurrency Workbench, version 7.0,
Fri Oct  6 11:36:58 BST 1995
Command: input "junk";
Command: sort Buff3;
{a,'b}
Command: size Buff3;
Buff3 has 12 states.
Command: min (Buff3Min, Buff3);
Resetting tables...
Buff3Min has 4 states.
Command: vs (3, Buff3Min);
=== a a a ===>
=== a a 'b ===>
=== a 'b a ===>
Command: random (16, Buff3Min);
a,a,a,'b,a,'b,a,'b,'b,'b,a,a,a,'b,'b,'b
```


Formatting μ-calculus formulae. Modal Operators

If P is a proposition, $a_1, ...a_n$ are actions, and L is a set identifier, then the following are propositions:

- $[a_1, ...a_n]$P and [L]P - strong necessity

Agent A satisfies $[K]P$ if every K-derivative of A satisfies P; that is, there is an $a \in K$ such that $A \xrightarrow{a} A'$ and A' satisfies P.

- $[[a_1, ...a_n]]$P and [[L]]P - weak necessity

Agent A satisfies $[K]P$ if every K-observation derivative of A satisfies P; that is, there is an $a \in K$ such that $A \stackrel{a}{\Rightarrow} A'$ and A' satisfies P.

- $< a_1, ...a_n >$P and <L>P - strong necessity

Agent A satisfies $[K]P$ if it has a K-derivative of A satisfies P; that is, there is an $a \in K$ such that $A \xrightarrow{a} A'$ and A' satisfies P.

- $<< a_1, ...a_n >>$P and <<L>>P - weak necessity

Formatting μ-calculus formulae (Cont'd)

- `-` indicates any transition (e.g. `[-]`). In μ-calculus, use [.].
- For strong modalities, the action sets must not include the empty action `eps`. For weak modalities, they must not include the unobservable action `tau`.

- *Propositional Connectives*: if P and Q are propositions, then so are `T`(true), `F`(false), `P` (negation), `P&Q` (conjunction), `P|Q` (disjunction) and `P=>Q` (implication).

- *Fixed Point Operators*: greatest fixpoint $\nu X.P$ is `max(X.P)`; least fixpoint $\mu X.P$ is `min(X.P)`.

To check a property, use command

```
checkprop (A, P);
```


Macros for Conversion between μ-calculus and CTL

`/local/share/cwb/examples/ccs/tl.macro`:

```
*  "Bx" is the "Box" ("always") operator. AG.
prop Bx(P) =  max(Z.P & [-]Z);
*  "Poss" is the "possibility" operator. EF.
prop Poss(P) = min(Z.P | <->Z);
* "Ev" is the "eventuality" operator.  AF and
* future states exist.
prop Ev(P) = min(Z.P | ([-]Z & <->T));
prop StrongUntil(P,Q) =
         min(Z.Q | (P & [-]Z & <->T));
prop WeakUntil(P,Q) = max(Z. Q | (P & [-]Z));
```

Some Useful CWB Commands

- `help`, `quit`
- `agent`, `set`, `relabel`, `prop`, `print`, `clear`
- `input "file"`, `output "file"` - send CWB output to a file rather than terminal, `save "file"`
- `sim` - simulate behavior of an agent using interactive simulation (see manual)
- `checkprop`
- `transitions` - list single-step transitions of an agent, `min`, `init` - observable actions that agent can perform immediately, `vs` - visible sequences of length `n`, `random`, `sort`, `size`, `states` - list the state-space of finite-state agent, `deadlocks` - find deadlocks and list traces leading to them
- `eq` - two agents are observationally equivalent, `pre` two agents are related by the weak divergence (bisimulation) preorder.

For More Information...

- See `man` pages for CWB
- See CWB Manual (Version 7)
- See examples in `/local/share/cwb/examples/ccs`
- See R. Milner, **Communication and Concurrency**, Prentice Hall International, 1989.
- See D. Kozen, *"Results on the Propositional μ-Calculus"*, Theoretical Computer Science 27, pp. 333-354, 1983.

What Other (Untimed) Process Algebras Are Out There?

- CCS (Calculus of Communicating Systems)
- Milner
- CSP (Communicating Sequential Processes)
- C.A.R. Hoare, **Communicating Sequential Processes**, Prentice Hall, 1985.
- ASP (Algebra of Communicating Processes)
- J. Begstra and J. Klop. *"Algebra of Communicating Processes with Abstraction"*. Journal of Theoretical Computer Science, 37:77-121, 1985.
- SCCS (Synchronous CCS) - used in CWB. Reference?

What Other (Timed) Process Algebras Are Out There?

- CSR (Communicating Shared Resources) - R. Gerber, Ph.D. Thesis, University of Pennsylvania, 1991.
- ACSR (Asynchronous CSR) - P. Bremond-Gregoire, J.Y. Choi and I. Lee, *"The Soundness and Completeness of ACSR"*, Technical Report MS-CIS-93-59, Univ. of Pennsylvania, June 1993.
- Timed CSP - G. Reed and A. Roscoe. *"Metric Spaces as Models for Real-Time Concurrency"*, in **Proceedings of Mathematical Foundations of Computer Science, LNCS**, volume 298, Springer-Verlag, 1987.
- and many others.