

ABCD cheatsheet

```
$ abcd [option]... spec.abcd
```

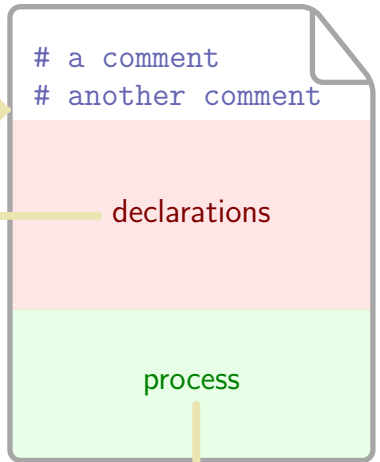
`--pnml=FILE` save net as PNML (SNAKES' variant)

`--dot=FILE` `--neato=FILE` `--towpi=FILE`

`--circo=FILE` `--fdp=FILE` draw net using a GraphViz engine

`--load=PLUGIN` load a plugin before to build net (may be repeated)

`--simul` start interactive simulator



buffer declarations:

`buffer name: type = ()`

▷ empty buffer

`buffer name: type = val, ...`

▷ buffer with initial content

type expressions:

any Python type or class

▷ eg, `int`, `str`, `object`, ...,
or user-defined classes

`enum(val, val, ...)`

▷ enumerated type

`type * type`

▷ cross-product of types

`type | type`

▷ union of types

`type & type`

▷ intersection of types

types definition:*

`typedef name: type`

constants:*

`const name = expr`

symbols:*

`symbol name, ...`

▷ define fresh unique values

Python imports:*

▷ just use regular Python imports

sub-processes

control flow:

`process ; process`

▷ sequential composition

`process + process`

▷ non-deterministic choice
(use opposite guards to
make it deterministic)

`process * process`

▷ iterate left-hand-side and
exit with right-hand-side

`process | process`

▷ parallel composition

(no priorities ⇒ use parentheses)

sub-process instances:

`name(args)`

▷ substitute `args` in net

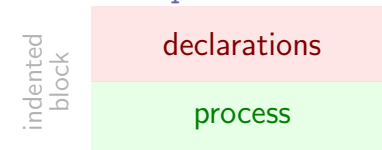
`name`

scope its local buffers

insert the net

sub-process declarations:

`net name (params):`



sub-process parameters:

a comma-separated list of:

`name`

▷ a value is expected

`name: buffer`

▷ a buffer name is expected

atomic actions:

`[True]`

▷ no-op non-blocking action

`[False]`

▷ always-blocking action

`[accesses]`

▷ unguarded action

`[accesses if expr]`

▷ guarded action

accesses:

`buff-(val)`

▷ consume `val` from `buff`

`buff-(var)`

▷ consume a value from `buff`
and binds it to `var`

`buff+(expr)`

▷ produce a value into `buff`

`buff?(val)`

▷ test for `val` in `buff`

`buff?(var)`

▷ test for a value in `buff`
and binds it to `var`

`buff>>(var)`

▷ flush `buff` into `var`

`buff<<(var)`

▷ add the values contained in
`var` to `buff`

`buff<>(val=expr)`

▷ replace `val` in `buff`
with `expr`

`buff<>(var=expr)`

▷ replace `var` in `buff`
with `expr`

(a comma-separated list of accesses
is performed atomically)

* global declarations only



© 2018 Franck Pommereau

franck.pommereau@univ-evry.fr

snakes.ibisc.univ-evry.fr

CC BY-SA