APA: Syntax and Semantics

Draft, Version 0.6

Howard Barringer, Donal Fellows, Graham Gough, Alan Williams*
Department of Computer Science, University of Manchester, UK

email: {howard,fellowsd,graham,alanw}@cs.man.ac.uk

May 31, 2000

1 APA

1.1 APA Syntax

APA has the following syntax:

$$
\begin{array}{rcll}
Env & ::= & pdefn_1, \ldots, pdefn_k, proc & \\
PDefn & ::= & pnm(fargs) \triangleq P & \text{Process definition} \\
Proc & ::= & \textsf{skip} \mid \textsf{fail} & \text{Atomic processes} \\
 & \mid & \textsf{sync}\{ichans\}\{ochans\}\{P_1\} & \text{Synchronise} \\
 & \mid & guard \rightarrow P_1 & \text{Guard} \\
 & \mid & P_1 \ ; \ P_2 & \text{Sequence} \\
 & \mid & \mathbf{0} & \text{Zero} \\
 & \mid & P_1 \mid P_2 & \text{Parallel}
\end{array}
\quad
\begin{array}{lll}
\mid & P_1 \ + \ P_2 & \text{Choice} \\
\mid & P_1 \backslash \{ch\} & \text{Restrict} \\
\mid & P_1(ch_1 \mapsto ch_2 & \text{Rename} \\
\mid & (\nu\, x : S \ \bullet \ P_1) & \text{Variable binding} \\
\mid & pnm(args) & \text{Instance}
\end{array}
$$

)
with the following auxiliary constructs:

$$
\begin{array}{rcl}
Ichans & ::= & ichanbind_1, \ldots, ichanbind_k \\
Ochans & ::= & ochanbind_1, \ldots, ochanbind_k \\
IChanBind & ::= & chnm\mathbf{?}x \\
OChanBind & ::= & chnm\mathbf{!}expr \\
ChanBind & ::= & ichanbind | ochanbind \\
Args & ::= & \textsf{vals}\ expr_1, \ldots, expr_k \\
 & & \quad \textsf{inputs}\ chnm_1, \ldots, chnm_m \\
 & & \quad \textsf{outputs}\ chnm_1, \ldots, chnm_n
\end{array}
$$

where $guard \in BoolExpr$, $x \in VarName$, $chnm \in ChanName$, $pnm \in ProcName$ with $VarName, ChanName, ProcName$ being sets of identifiers (possibly ditinguished for implementation purposes). $Expr$ is the set of value expressions defined for Rainbow, which contains boolean expressions $BoolExpr$. The set of literals in $Expr$ is Val, with the set of values in a binding $S \subseteq Val$. Any of $ichans, ochans$ or the components of the actual arguments $args$ of a process instance can be empty.

The process declaration environment $env \in Env$ is simply a list of declarations, together with a the main process to be executed with respect to the declarations.

*The University of Manchester acknowledges support from the Engineering and Physical Sciences Research Council via research grants GR/K42073 and GR/M05744.

Note that no typing or other static semantic information is currently included in APA.

- There are strong similarities with EPA
- Restrict blocks any transition where *ch* appears in its label, returning chaos
- Variable binding introduces a new bound variable x which takes on some value from the set S in a process P_1. The scope of x is P_1
- **skip** and **fail** represent an 'empty' process and a failed process (there are no transitions from the latter, except the empty transition).

2 Transition Rules for APA

2.1 Syntax for APA Transition Rule Components

$$
\begin{array}{rcl}
Transition & ::= & P_1 \xrightarrow{action} P_2 \\
Action & ::= & \langle bool, bindset \rangle \mid \boxtimes \\
BindSet & ::= & chanbind_1, \ldots, chanbind_k
\end{array}
$$

As with EPA, transitions can be labelled with the 'zero' action $\boxtimes$, indicating an improper transition has been formed. These will not be considered as part of the proper transition set for a process.

$$\boxed{\text{Chaos}}\ \frac{}{P \xrightarrow{\boxtimes} P'} \qquad \boxed{\text{Skip}}\ \frac{}{\textbf{skip} \xrightarrow{\langle \mathsf{T}, \emptyset \rangle} \mathbf{0}}$$

$$\boxed{\text{Block}}\ \frac{P \xrightarrow{a\,/\,b} P'}{P \xrightarrow{\langle \mathsf{F}, [a]\,/\,[b], \rangle} P}\,(b \neq \emptyset) \qquad \boxed{\text{Seq-1}}\ \frac{P_1 \xrightarrow{\langle \mathsf{F}, a \rangle} P_1'}{P_1\ ;\ P_2 \xrightarrow{\langle \mathsf{F}, a \rangle} P_1'\ ;\ P_2} \qquad \boxed{\text{Seq-2}}\ \frac{P_1 \xrightarrow{\langle \mathsf{T}, a \rangle} P_1'}{P_1\ ;\ P_2 \xrightarrow{\langle \mathsf{F}, a, \rangle} P_2}$$

$$\boxed{\text{Par-1}}\ \frac{P_1 \xrightarrow{\langle \mathsf{F}, a \rangle} P_1'}{P_1 P \mid {}_2 \xrightarrow{\langle \mathsf{F}, a \rangle} P_1' P \mid {}_2} \qquad \boxed{\text{Par-2}}\ \frac{P_1 \xrightarrow{\langle \mathsf{T}, a \rangle} P_1'}{P_1 P \mid {}_2 \xrightarrow{\langle \mathsf{F}, a \rangle} P_2}$$

$$\boxed{\text{Par-3}}\ \frac{\begin{array}{ll}\text{(a)} & P_1 \xrightarrow{\langle \mathsf{F}, a \rangle} P_1' \\ \text{(b)} & P_2 \xrightarrow{\beta} P_2'\end{array}}{P_1 P \mid {}_2 \xrightarrow{\langle \mathsf{F}, a \rangle \star \beta} P_1' P \mid {}_2'} \qquad \boxed{\text{Par-4}}\ \frac{\begin{array}{ll}\text{(a)} & P_1 \xrightarrow{\langle \mathsf{T}, a \rangle} P_1' \\ \text{(b)} & P_2 \xrightarrow{\beta} P_2'\end{array}}{P_1 P \mid {}_2 \xrightarrow{\langle \mathsf{T}, a \rangle \star \beta} P_2'}$$

$$\boxed{\text{Sum}}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \boxed{\text{Bind}}\ \frac{\text{for some } u \in S \qquad P\,[\,u/x\,] \xrightarrow{\alpha} P'}{(\nu\, x : S \ \bullet\ P) \xrightarrow{\alpha} P'}$$

$$\boxed{\text{Guard}}\ \frac{b = \mathsf{true}, P \xrightarrow{\alpha} P'}{b \rightarrow P \xrightarrow{\alpha} P'}$$

$$\boxed{\text{Call}}\ \frac{pnm(\overline{x}) \triangleq P \quad P\,[\,\overline{v}/\overline{x}\,] \xrightarrow{\alpha} P'}{pnm(\overline{v}) \xrightarrow{\alpha} P'} \qquad \boxed{\text{Hide}}\ \frac{P \xrightarrow{\alpha} P'}{P \backslash \{ch\} \xrightarrow{hide(\alpha, ch)} P' \backslash \{ch\}}$$

$$\boxed{\text{Rename}}\ \frac{P(\overline{x}) \triangleq P \quad P\,[\,ch_2/ch_1\,] \xrightarrow{\alpha} P'}{P(ch_1 \mapsto ch_2) \xrightarrow{\alpha} P'}$$

$$\boxed{\textsf{Sync-1}}\ \frac{P \xrightarrow{\langle \mathsf{T}, c\rangle} P'}{\textbf{sync}\{a\}\{b\}\{P\} \xrightarrow{\langle \mathsf{T}, a\ /\ b\rangle \star \langle \mathsf{T}, c\rangle} \mathbf{0}}$$

$$\boxed{\textsf{Sync-2}}\ \frac{\text{for some } b' \subseteq b \quad b'' = b - b' \quad c \subseteq [b''] \quad P \xrightarrow{\langle \mathsf{F}, d\rangle} P'}{\textbf{sync}\{a\}\{b\}\{P\} \xrightarrow{\langle \mathsf{F}, [a]\ /\ b', c\rangle \star \langle \mathsf{F}, d\rangle} \textbf{sync}\{a\}\{b''\}\{P'\}}$$

$$\boxed{\textsf{Sync-3}}\ \frac{\text{for some } b' \subseteq b \quad b'' = b - b' \quad c \subseteq [b'']}{\textbf{sync}\{a\}\{b\}\{P\} \xrightarrow{\langle \mathsf{F}, ([a]\ /\ b', c)\rangle} \textbf{sync}\{a\}\{b''\}\{P\}}$$

- In Sync-2 b' may be empty; in Sync-3 b' must not be empty
- In Sync rules, some of the output actions may be blocked, indicated by actions $c \subseteq [b'']$ — note that these are retained in the target process within b''.
- Action labels: α, β are actions, $a, b, c, d...$ are input/output actions, e.g. with $a\ /\ b = \alpha$. Each input/output action is a map (from channels to data expressions), with input/output channels distinguished with '?' and '!'. Action product is indicated by $\alpha \star \beta$. Blocked (AKA postponed) actions are denoted by $[a]$ or $[\alpha]$. Note that although we now use '?' and '!', and there is strictly no need for separation of actions into input and output actions, it is convenient to keep the '$a\ /\ b$' format since it is utilised in the transition rules.
- Block (AKA Postpone): only transitions with non-empty output actions can be blocked, i.e. $b \neq \emptyset$
- Hide: the function $hide(\alpha, ch)$ returns α if ch is not in its domaon, otherwise is returns chaos.
- Skip, fail: Skip indicates that the first process in a sequence has completed. Fail results from a false boolean guard. There are no rules specifically for fail.
- Sync: b' is a subset of the possible actions still to do in b, and b'' is then those that are leftover for future transitions, i.e. $b' \subseteq b$ and $b'' = b - b'$.

2.2 Action Product

The action product is indicated by $\alpha \star \beta$. In the following definition, the 'destructive' cases resulting in the zero action are given first. The 'useful' case, yielding a non-zero action, is given last:

$$\_\star\_ \ :\ Action \times Action \to Action$$

$$
\begin{array}{l}
\boxtimes \star \langle bool_2, bindset_2\rangle = \boxtimes \\
\langle bool_2, bindset_2\rangle \star \boxtimes = \boxtimes \\
\langle bool_1, bindset_1\rangle \star \langle bool_2, bindset_2\rangle = \langle bool_1 \wedge bool_2, bindset_3\rangle \\
\quad \textsf{where} \\
\qquad \begin{array}{rcl}
bindset_3 & = & \boxtimes \text{ if } bind_1 \in bindset_1 \wedge bind_2 \in bindset_2 \\
& & \qquad \wedge\ \mathsf{chanof}(bind_1) = \mathsf{chanof}(bind_2) \\
& & (\text{for any } val_1, val_2) \\
& = & \boxtimes \text{ if } bind_1 \in bindset_1 \wedge bind_2 \in bindset_2 \\
& & \qquad \wedge\ \mathsf{chanof}(bind_1) = \overline{\mathsf{chanof}(bind_2)} \\
& & \qquad \wedge\ \mathsf{valof}(bind_1) \neq \mathsf{valof}(bind_2) \\
& & (\text{for any } val_1, val_2,\ \text{where } val_1 \neq val_2 \\
& = & bindset_1 \bigcup bindset_2\ - \{bind_1, bind_2 | bind_1 = \overline{bind_2}\} \\
& & \text{otherwise}
\end{array} \\
chanof(chnm\mathbf{?}expr) = chnm\mathbf{?} \\
chanof(chnm\mathbf{!}expr) = chnm\mathbf{!} \\
valof(chnm\mathbf{?}expr) = expr \\
valof(chnm\mathbf{!}expr) = expr
\end{array}
$$

This says

1. $bindset_1, bindset_2$ cannot both contain channels with the same name and direction; if they do, return the 'empty' action
2. $bindset_1$ cannot contain a channel whose complement is in $bindset_2$ if the values bound to the channels are different; again if this occurs, return the 'empty' action
3. Otherwise the union of the channel bindings is formed, with complementing channel binding pairs removed.

Examples

1. $\langle \mathsf{F}, a?1, b!2, c?3 \rangle \star \langle \mathsf{T}, a!1, b?2, d!4 \rangle = \langle \mathsf{F}, c?3, d!4 \rangle$
2. $\langle \mathsf{T}, a?1, b!2 \rangle \star \langle \mathsf{T}, a?1, c!3 \rangle = \boxtimes$
   (bindsets both contain an input channel a)
3. $\langle \mathsf{F}, a?1, b!2 \rangle \star \langle \mathsf{T}, a!2, c?3 \rangle = \boxtimes$
   (bindsets contain complementary channels but values bound are different)

Blocked (AKA Postponed) Actions When a channel binding in an action is blocked, or a binding set is blocked, then the channels involved are simply so that they are distinct from those in *ChanName*. Channel matching during action product then proceeds in exactly the same way, using the new names.

In a channel binding set, then some channels may be blocked, while others can proceed.

Channel Complementation $Ctrl?42$ and $Ctrl!42$ are a matching pair of channel bindings for a channel named $Ctrl$. Channel complementation simply returns to co-channel name:

$$\overline{chnm?expr} = chnm!expr$$
$$\overline{chnm!expr} = chnm?expr$$
$$\overline{chnm?} = chnm!$$
$$\overline{chnm!} = chnm?$$

Free Variables Variables are bound via the binding construct $(\nu\, x : S \;\bullet\; P)$, so that the variable x is bound in process P. Otherwise variables are free.

Clearly this has effect during variable substitution. $P[\,expr/x\,]$ will replace all free occurences of x in P with the expression *expr*, but not altering the bound occurences.

Channel Hiding Channel hiding applied to an action requires that the action does *not* contain the channel specified, *except* if it is a blocked output channel, allowing a process to offer a (blocked) communication on a channel, even if this is not consumed because the corresponding input is not accepting:

$$\begin{array}{lcll}
hide : Action \times ChanNames \rightarrow Action & & & \\
\quad hide(\langle bool, bindset \rangle, chnm) & = & (bindset - [chnm!expr_1]) & \text{if } chnm?expr_2, \\
 & & & \quad [chnm?expr_2], \\
 & & & \quad chnm!expr_3 \notin bindset \\
 & = & \boxtimes & \text{otherwise}
\end{array}$$

A blocked output channel is also removed from the action label so that it remains within the scope of the hidden channel definition (otherwise it could match with a blocked input channel from a separate process outside the hiding)

2.3 APA Definitions for Rainbow Constructs

The semantics of Rainbow essentially constructs netlists of calls to APA process definitions, with usualllly one call for each atomic component in Rainbow. These basic APA process definitions are included in the environment within which the process term representing the Rainbow design is evaluated.

Because Yellow is similar to APA then only the following separate process definition for a loop construct is beneficial. If each Yellow $\mathsf{guardedbody}_i$ in a $\mathsf{guardedchoice}$ is represented in APA by guard expressions b_i and guarded bodies P_i, for $i = 1, \ldots, k$, then the following loop process can be defined:

$$
\begin{aligned}
& Loop(b_1, \ldots, b_k, P_1, \ldots, P_k) \triangleq \\
& \quad \sum_{i=1}^{k} not(b_1), \ldots, not(b_{i-1}), b_i \to P_i \; ; \; Loop(b_1, \ldots, b_k, P_1, \ldots, P_k) \\
& \quad + \; not(b_1), \ldots, not(b_k) \to \textsf{skip}
\end{aligned}
$$

All other Yellow constructs can be defined directly via APA. Note that a static check is necessary to avoid the situation where all guarded bodies for which there are true guards reduce to skips, which would lead to the problem of a recursive definition with only 'tau' actions guarding each recursive call.

The following process definitions are required in Green:

$$\textsf{Sink}(ty \; ; \; \texttt{i1} \; / \;) \triangleq (\nu \, x : ty \, \bullet \, \textsf{sync}\{i1?x\}\{\}\{\}) \; ; \; \textsf{Sink}(ty \; ; \; \texttt{i1} \; / \;)$$

$$\textsf{Const}(val, ty \; ; \; / \; \texttt{o1}) \triangleq \textsf{sync}\{\}\{o1!val\}\{\} \; ; \; \textsf{Const}(val, ty \; ; \; / \; \texttt{o1})$$

$$\textsf{Chan}(ty \; ; \; ichnm \; / \; \texttt{o1}) \triangleq (\nu \, x : ty \, \bullet \, \textsf{sync}\{ichnm1?x\}\{o1!x\}\{\}) \; ; \; \textsf{Chan}(ty \; ; \; ichnm \; / \; \texttt{o1})$$

$$\textsf{Chan}(ty \; ; \; \texttt{i1} \; / \; ochnm) \triangleq (\nu \, x : ty \, \bullet \, \textsf{sync}\{i1?x\}\{ochnm!x\}\{\}) \; ; \; \textsf{Chan}(ty \; ; \; \texttt{i1} \; / \; ochnm)$$

$$
\begin{aligned}
& \textsf{Buf}(val, ty \; ; \; \texttt{i1} \; / \; \texttt{i2}) \triangleq \\
& \quad \textsf{sync}\{\}\{o1!val\}\{\textsf{skip}\} \; ; \; \textsf{EBuf}(ty \; ; \; \texttt{i1} \; / \; \texttt{i2})
\end{aligned}
$$

$$\textsf{EBuf}(ty \; ; \; \texttt{i1} \; / \; \texttt{i2}) \triangleq (\nu \, x : ty \, \bullet \, \textsf{sync}\{i1?x\}\{\}\{\textsf{skip}\} \; ; \; \textsf{sync}\{\}\{o1!x\}\{\textsf{skip}\}) \; ; \; \textsf{EBuf}(ty \; ; \; \texttt{i1} \; / \; \texttt{i2})$$

$$
\begin{aligned}
& \textsf{Split}_k(lbl_1, \ldots, lbl_k, ity, cty \; ; \; \texttt{i1} \; / \; o1, \ldots, ok, c) \triangleq \\
& \quad \sum_{j=1}^{k} (\nu \, x : ity \, \bullet \, \textsf{sync}\{i1?x, c?lbl_j\}\{o_j!x\}\{skip\}) \; ; \; \textsf{Split}_k(lbl_1, \ldots, lbl_k, ity, cty \; ; \; \texttt{i1} \; / \; o1, \ldots, ok, c)
\end{aligned}
$$

$$
\begin{aligned}
& \textsf{Merge}_k(lbl_1, \ldots, lbl_k, ity, cty \; ; \; i1, \ldots, ik, c \; / \; \texttt{o1}) \triangleq \\
& \quad \sum_{j=1}^{k} (\nu \, x : ity \, \bullet \, \textsf{sync}\{i_j?x, c?lbl_j\}\{o1!x\}\{skip\}) \; ; \; \textsf{Merge}_k(lbl_1, \ldots, lbl_k, ity, cty \; ; \; i1, \ldots, ik, c \; / \; \texttt{o1})
\end{aligned}
$$

$$
\begin{aligned}
& \textsf{Arbitrate}(ty \; ; \; i1, i2 \; / \; \texttt{o1}) \triangleq \\
& \quad (\nu \, x : ty \, \bullet \, \textsf{sync}\{i1?x\}\{\}\{\textsf{skip} \; ; \; \textsf{sync}\{\}\{o!x\}\{\}\}) \; ; \; \textsf{Arbitrate}(ty \; ; \; i1, i2 \; / \; \texttt{o1}) \\
& \qquad + \; (\nu \, x : ty \, \bullet \, \textsf{sync}\{i2?x\}\{\}\{\textsf{skip} \; ; \; \textsf{sync}\{\}\{o!x\}\{\}\}) \; ; \; \textsf{Arbitrate}(ty \; ; \; i1, i2 \; / \; \texttt{o1})
\end{aligned}
$$

$$
\begin{aligned}
& \textsf{Tuple}_k(ty_1, \ldots, ty_k \; ; \; \texttt{i1}, \ldots, \texttt{ik} \; / \; \texttt{o1}) \triangleq \\
& \quad (\nu \, x_1 : ty_1, \ldots, x_k : ty_k \, \bullet \, \textsf{sync}\{i1?x_1, \ldots, ik?x_k\}\{o!\langle x_1, \ldots, x_k \rangle\}\{\}) \; ; \; \textsf{Tuple}_k(ty_1, \ldots, ty_k \; ; \; \texttt{i1}, \ldots, \texttt{ik} \; / \; \texttt{o1})
\end{aligned}
$$

$$
\begin{aligned}
& \textsf{UnTuple}_k(\langle ty, \ldots, ty_k \rangle \; ; \; \texttt{i1} \; / \; \texttt{o1}, \ldots, \texttt{ok}) \triangleq \\
& \quad (\nu \, x_1 : ty_1, \ldots, x_k : ty_k \, \bullet \, \textsf{sync}\{i1?\langle x_1, \ldots, x_k \rangle\}\{o1!x_1, \ldots, ok!x_k\}\{\}) \; ; \; \textsf{UnTuple}_k(\langle ty, \ldots, ty_k \rangle \; ; \; \texttt{i1} \; / \; \texttt{o1}, \ldots, \texttt{ok})
\end{aligned}
$$

$$
\begin{aligned}
& \textsf{Synchronise}_k(ty_1, \ldots, ty_k \; ; \; \texttt{i1}, \ldots, \texttt{ik} \; / \; \texttt{o1}, \ldots, \texttt{ok}) \triangleq \\
& \quad (\nu \, x_1 : ty_1, \ldots, x_k : ty_k \, \bullet \, \textsf{sync}\{i1?x_1, \ldots, ik?x_k\}\{o1!x_1, \ldots, ok!x_k\}\{\}) \; ; \; \textsf{Synchronise}_k(ty_1, \ldots, ty_k \; ; \; \texttt{i1}, \ldots, \texttt{ik} \; / \; \texttt{o1}, \ldots, \texttt{ok}
\end{aligned}
$$

The runtime evaluation environment generated during compilation of Rainbow stores bindings for register and value variables from Rainbow. At the pure APA level, registers and values are modelled by processes. In order to simplify problems with separate or simultaneous read/writes from registers, *all* read/writes are simultaneous.

It is also necessary to terminate the register when the Yellow block in which it declares terminates; this is achieved via a a channel t in the following, where synchronisations on t are added after the APA representing the block (see end of this section):

$$\textsf{\textbf{Reg}}(ty\ ;\ w,t\ /\ r) \triangleq (\nu\, x : ty\ \bullet\ \textsf{\textbf{sync}}\{w?x\}\{\}\{\textsf{\textbf{skip}}\}\ ;\ \textsf{\textbf{Reg}}(x,ty\ ;\ w,t\ /\ r))$$
$$+\ \textsf{\textbf{sync}}\{t?()\}\{\}\{\textsf{\textbf{skip}}\}$$

$$\textsf{\textbf{Reg}}(x,ty\ ;\ w,t\ /\ r) \triangleq (\nu\, y : ty\ \bullet\ \textsf{\textbf{sync}}\{w?y\}\{rx\}\{\textsf{\textbf{skip}}\}\ ;\ \textsf{\textbf{Reg}}(y,ty\ ;\ w,t\ /\ r))$$
$$+\ \textsf{\textbf{sync}}\{t?()\}\{\}\{\textsf{\textbf{skip}}\}$$

Assume an expression contains free variables $x1,\ldots,xk$ that are values to be read from registers $r1,\ldots,rk$:

$$F(expr\ ;\ read_{r1},\ldots,read_{rk}\ /\ o,write_{r1},\ldots,write_{rk}) \triangleq$$
$$(\nu\, x1,\ldots,xk : S\ \bullet\ \textsf{\textbf{sync}}\{read_{r1}?x1,\ldots,read_{rk}?xk\}\{o!expr,write_{r1}!x1,\ldots,write_{rk}!xk\}\{skip\}\ ;\ F(expr\ ;\ read_{r1},\ldots,read_{rk}$$

For an assignment of an expression value to register r:

```
r := expr
```

then the old register value is discarded into a sink and the return value is assigned via renaming:

$$(F(expr\ ;\ read_{r1},\ldots,read_{rk}\ /\ o1,write_{r1},\ldots,write_{rk})(o1 \mapsto write_r write_r \mapsto throw) \mid \textsf{\textbf{Sink}}(throw\ ;\ \texttt{i1}\ /\ ))\backslash\{throw\}$$

If *expr* does not read register *r* then renaming of channel $write_r$ will have no effect.
When a register *r* is declared within a Yellow block:

```
{ reg r:T ; B }
```

then a register process is placed in parallel with the process P_B representing the Yellow body `B`; read/write channels are then hidden:

$$(\textsf{\textbf{Reg}}(T\ ;\ write_r\ /\ read_r) \mid (P\ ;\ \textsf{\textbf{sync}}\{\}\{t!()\}\{\textsf{\textbf{skip}}\})\backslash\{read_r,write_r\}$$

2.4 Derived Rules for Yellow

$$\boxed{\textsf{Assign}}\ \frac{\text{M-}[\![e]\!](\sigma) = v}{\langle r\ :=\ e,\sigma\rangle \xrightarrow{\langle \mathsf{T},\emptyset\rangle} \langle \mathbf{0},\sigma \dagger \{r \mapsto v\}\rangle}$$

$$\boxed{\textsf{Loop-1}}\ \frac{b = T \quad b \to P \xrightarrow{\alpha} P'}{\textsf{\textbf{loop}}\ b \longrightarrow P\textsf{\textbf{end\_loop}} \xrightarrow{\alpha} P\ ;\ \textsf{\textbf{loop}}\ b \longrightarrow P\textsf{\textbf{end\_loop}}} \qquad \boxed{\textsf{Loop-2}}\ \frac{b = F}{\textsf{\textbf{loop}}\ b \longrightarrow P\textsf{\textbf{end\_loop}} \xrightarrow{\langle \mathsf{T},\emptyset\rangle} \mathbf{0}}$$

- same-register assign or assign/read combinations in different parallel processes are not permitted
- , ; — translate directly into APA
- **accept** $\{ins\}$ **do** *P* **end** $\triangleq$ $\textsf{\textbf{sync}}\{ins\}\{\}\{P\}$
- **call** $\{outs\}$ $\triangleq$ $\textsf{\textbf{sync}}\{\}\{outs\}\{\textsf{\textbf{skip}}\}$ (possibly even with the '**skip**' omitted)

2.5 Derived Rules for Green

$$\boxed{\textsf{Pipe}}\ \dfrac{\begin{array}{ll}\text{(a)} & P \xrightarrow{a \,/\, b} P' \\ \text{(b)} & Q \xrightarrow{b \,/\, c} Q'\end{array}}{P \ >> \ Q \xrightarrow{a \,/\, c} P' \ >> \ Q'}$$

$$\boxed{\textsf{Buf-1}}\ \dfrac{}{\mathcal{B}(x\,;\,\texttt{i1}\,/\,\texttt{i2}) \xrightarrow{o!x} \mathcal{B}(-\,;\,\texttt{i1}\,/\,\texttt{i2})} \qquad \boxed{\textsf{Buf-2}}\ \dfrac{}{\mathcal{B}(-\,;\,\texttt{i1}\,/\,\texttt{i2}) \xrightarrow{i?x} \mathcal{B}(x\,;\,\texttt{i1}\,/\,\texttt{i2})}$$

$$\boxed{\textsf{Split-1}}\ \dfrac{}{\textbf{Split} \xrightarrow{i?x,c?true \,/\, o_1!x} \textbf{Split}} \qquad \boxed{\textsf{Split-2}}\ \dfrac{}{\textbf{Split} \xrightarrow{i?x,c?false \,/\, o_2!x} \textbf{Split}}$$

$$\boxed{\textsf{Merge-1}}\ \dfrac{}{\textbf{Merge} \xrightarrow{i_1?x,c?true \,/\, o!x} \textbf{Merge}} \qquad \boxed{\textsf{Merge-2}}\ \dfrac{}{\textbf{Merge} \xrightarrow{i_2?x,c?false \,/\, o!x} \textbf{Merge}}$$

$$\boxed{\textsf{Dup-1}}\ \dfrac{}{\textsf{Dup}_{<>} \xrightarrow{i?x \,/\, o_1!x,o_2!x} \textsf{Dup}_{<>}} \qquad \boxed{\textsf{Dup-2}}\ \dfrac{}{\textsf{Dup}_{<>} \xrightarrow{[i?x] \,/\, o_1!x} \textsf{Dup}_{<o_1!x,[o_2!x]>}}$$

$$\boxed{\textsf{Dup-3}}\ \dfrac{}{\textsf{Dup}_{<o_1!x,[o_2!x]>} \xrightarrow{i?x \,/\, o_2!x} \textsf{Dup}_{<>}}$$

$$\boxed{\textsf{Arb-1}}\ \dfrac{}{\oplus_{\langle\rangle} \xrightarrow{[i_1?x]} \oplus_{\langle i_1?x\rangle}} \qquad \boxed{\textsf{Arb-2}}\ \dfrac{}{\oplus_{\langle i_1?x\rangle} \xrightarrow{i_1?x \,/\, o!x} \oplus_{\langle\rangle}}$$