Language Specification for CHP and Friends

1 CHP

CHP is the highest-level language. Seen here is CHP with boolean support only.

1.1 Grammar

$$
\begin{aligned}
\ell &::= \top \mid \bot \\
\mathrm{b} &::= a \mid \ell \mid \overline{A?} \mid \overline{A!} \mid \widehat{A} \mid \mathrm{b}_1 \wedge \mathrm{b}_2 \mid \mathrm{b}_1 \vee \mathrm{b}_2 \mid \neg \mathrm{b} \mid (\mathrm{b}) \\
\mathrm{P} &::= a := \mathrm{b} \mid S \mid C \mid P_1; P_2 \mid P_1 || P_2 \mid \texttt{skip} \mid N \mid (P) \\
\mathrm{S} &::= [\mathrm{b}_1 \to P_1 \,[]\, \ldots \,[]\, \mathrm{b}_n \to P_n] \mid [\mathrm{b}_1 \to P_1 | \ldots | \mathrm{b}_n \to P_n] \\
&\quad \mid *[\mathrm{b}_1 \to P_1 \,[]\, \ldots \,[]\, \mathrm{b}_n \to P_n] \mid *[\mathrm{b}_1 \to P_1 | \ldots | \mathrm{b}_n \to P_n] \\
\mathrm{C} &::= A!\mathrm{b} \mid A?a \mid C_1 \bullet C_2 \mid \underline{C} \\
\mathrm{N} &::= \texttt{new}\ a\ P \mid \texttt{NEW}\ A\ P \mid \underline{N}
\end{aligned}
$$

Above, we are using $a...z$ to range over variables, and $A...Z$ to range over channels.

Additionally, there is some well-accepted syntactic sugar:

$L?$ or $R!$ are dataless channels. They are equivalent to receiving into a new variable, and sending an arbitrary value, respectively.

$*[S]$ is syntactic sugar for $*[\top \to S]$, which is used for infinite loops.

[b] is used to mean [b $\to$ `skip`], which means that we wait for a boolean expression to be true before continuing.

In addition, nothing that is underlined is valid in the definition of a program; these are all intermediate expressions that can only arise during program execution.

1.2 Environment Variables and Notation

Here we use ξ as a metasyntactic variable for the elements of σ.

$$
\begin{aligned}
\text{Set Definitions} &::= \mathbb{B} \mid \mathbb{V} \mid \mathbb{C} \mid \Sigma \mid \Omega \\
\text{Environment Variables} &::= \sigma, \varsigma, \varsigma_r, \varsigma_w, \rho, \rho_r, \rho_w, \omega \\
\text{Environment Variable Ops} &::= \oslash_\xi, \odot, \oplus, \oplus_a, \ominus, \triangleleft, \diamond, \triangleright, +, -
\end{aligned}
$$

The set definitions are:

1. $\mathbb{B} = \{\top; \bot\}$ is the set of booleans.
2. $\mathbb{V} = \{a; \ldots; z\}$ is the set of variables.
3. $\mathbb{C} = \{A; \ldots; Z\}$ is the set of channels.
4. $\Sigma = \{\sigma\}$ is the set of all environments.
5. $\Omega = \{\omega\}$ is the set of all stacks.

The environment variable definitions are:

1. $\sigma = \{\varsigma, \varsigma_r, \varsigma_w, \rho, \rho_r, \rho_w, \omega\}$ is simply the symbol that represents the entire environment. We will use e.g. $\sigma[\varsigma]$ to refer to the ς contained inside σ.
2. $\varsigma \subseteq \mathbb{V} \times \mathbb{B}$ is the map from in-scope variables to their boolean values.

3. $\varsigma_r \subseteq \mathbb{V}$ is the set of variables that the program currently has read access to.
4. $\varsigma_w \subseteq \mathbb{V}$ is the set of variables that the program currently has write access to.
5. $\rho \subseteq \mathbb{C}$ is the set of in-scope channels.
6. $\rho_r \subseteq \mathbb{C}$ is the multiset of channels currently being read from (is a set if channels are end-to-end).
7. $\rho_w \subseteq \mathbb{C} \times \mathbb{B}$ is the multimap from channels currently being written to to the booleans being sent on them (is a map if channels are end-to-end).
8. $\omega \in (\Sigma + (\Omega \times \Omega))^*$ is the stack of environments/double stacks, used for handling scoping.

The environment variable operation definitions are:

1. $\sigma \oslash_\xi \xi'$ is defined to be σ with ξ replaced with ξ' (e.g. $\sigma \oslash_\varsigma \varnothing = \{\varnothing, \sigma[\varsigma_r], \sigma[\varsigma_w], \sigma[\rho_r], \sigma[\rho_w]\}$).
2. $\sigma \odot a$ is defined to be $\sigma \oslash_\varsigma (\{(a', \ell) \mid (a', \ell) \in \sigma[\varsigma] \wedge a' \neq a\}) \oslash_{\varsigma_r} (\sigma[\varsigma_r] \cup \{a\}) \oslash_{\varsigma_w} (\sigma[\varsigma_w] \cup \{a\})$. In other words, this is σ with a added to both ς_r and ς_w (assuming it wasn't already there) and any binding for a removed from ς.
3. $\sigma \odot A$ is defined to be $\sigma \oslash_\rho (\sigma[\rho] \cup \{A\}) \oslash_{\rho_r} (\{A' \mid A' \in \sigma[\rho_r] \wedge A' \neq A\}) \oslash_{\rho_w} (\{(A', \ell) \mid (A', \ell) \in \sigma[\rho_w] \wedge A' \neq A\})$. In other words, this is σ with A added to ρ (assuming it wasn't already there) and any sends and receives currently in progress for A removed from ρ_w and ρ_r, respectively.
4. $\varsigma \oplus \varsigma'$ is defined to be $\{(a, \ell) \mid (a, \ell) \in \varsigma'\} \cup \{(a, \ell) \mid (a, \ell) \in \varsigma \wedge a \notin \mathrm{dom}(\varsigma')\}$. Intuitively, this represents merging the changes from ς' back into ς. Note that this operation is NOT commutative!
5. $\varsigma \oplus_a \varsigma'$ is defined to be $\{(a', \ell) \mid (a', \ell) \in \varsigma' \wedge a' \neq a\} \cup \{(a', \ell) \mid (a', \ell) \in \varsigma \wedge (a' = a \vee a' \notin \mathrm{dom}(\varsigma'))\}$. Intuitively, this represents merging the changes from ς' back into ς, but excluding any updates to a. Note that this operation is NOT commutative!
6. $\rho_r \ominus_A \rho'_r$ is defined to be $\{A' \mid A' \in \rho_r \wedge A' \neq A\} \cup \{A \mid A \in \rho_r\}$. In other words, this is ρ_r with all instances of A removed and replaced with all instances of A in ρ'_r.
7. $\rho_w \ominus_A \rho'_w$ is defined to be $\{(A', \ell) \mid (A', \ell) \in \rho_w \wedge A' \neq A\} \cup \{(A, \ell) \mid (A, \ell) \in \rho_w\}$. In other words, this is ρ_w with all pairs containing A removed and replaced with all pairs containing A in ρ'_w.
8. $\omega \triangleleft \sigma$ is ω with σ pushed onto it.
9. $\omega \triangleleft (\omega', \omega'')$ is ω with (ω', ω'') pushed onto it.
10. $\omega \diamond \sigma$ is ω with its top element replaced with σ (this operation gets 'stuck' if ω happens to be empty).
11. $\omega \diamond (\omega', \omega'')$ is ω with its top element replaced with (ω', ω'') (this operation gets 'stuck' if ω happens to be empty).
12. $\omega \triangleright$ is ω with its top element popped off (this operation gets 'stuck' if ω happens to be empty).
13. $\omega \triangleright \sigma$ assigns σ to be the top element of ω (this operation gets 'stuck' if ω happens to be empty).
14. $\omega \triangleright (\omega', \omega'')$ assigns (ω', ω'') to be the top element of ω (this operation gets 'stuck' if ω happens to be empty).
15. $+, -$ represent adding and removing an element from a multiset or multimap, respectively.

It is an invariant that $\varsigma_w \subseteq \varsigma_r \subseteq \mathrm{dom}(\varsigma)$; likewise, it is an invariant that $\rho_r \cup \mathrm{dom}(\rho_w) \subseteq \rho$.
σ is initialized to be $\{\varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \langle\rangle\}$.

1.3 Small-Step Semantics

1.3.1 Boolean Rules

$$\frac{}{\sigma \models \ell : \ell} \qquad \frac{\sigma \models \mathrm{b} : \top}{\sigma \models \neg \mathrm{b} : \bot} \qquad \frac{\sigma \models \mathrm{b} : \bot}{\sigma \models \neg \mathrm{b} : \top} \qquad \frac{\sigma \models \mathrm{b} : \top \quad \sigma \models \mathrm{b}' : \top}{\sigma \models \mathrm{b} \wedge \mathrm{b}' : \top} \qquad \frac{\sigma \models \mathrm{b} : \bot}{\sigma \models \mathrm{b} \wedge \mathrm{b}' : \bot} \qquad \frac{\sigma \models \mathrm{b}' : \bot}{\sigma \models \mathrm{b} \wedge \mathrm{b}' : \bot}$$

$$\frac{\sigma \models \mathrm{b} : \bot \quad \sigma \models \mathrm{b}' : \bot}{\sigma \models \mathrm{b} \vee \mathrm{b}' : \bot} \qquad \frac{\sigma \models \mathrm{b} : \top}{\sigma \models \mathrm{b} \vee \mathrm{b}' : \top} \qquad \frac{\sigma \models \mathrm{b}' : \top}{\sigma \models \mathrm{b} \vee \mathrm{b}' : \top} \qquad \frac{a \in \sigma[\varsigma_r] \quad (a, \ell) \in \sigma}{\sigma \models a : \ell}$$

$$\frac{A \in \sigma[\rho_r]}{\sigma \models \overline{A?} : \top} \qquad \frac{A \notin \sigma[\rho_r]}{\sigma \models \overline{A} : \bot} \qquad \frac{(A, \ell) \in \sigma[\rho_w]}{\sigma \models \overline{A!} : \top} \qquad \frac{(A, \ell) \notin \sigma[\rho_w]}{\sigma \models \overline{A!} : \bot} \qquad \frac{(A, \ell) \in \sigma[\rho_w]}{\sigma \models \widehat{A} : \ell}$$

1.3.2 LTS Rules

Here we use δ as a metasyntactic variable for the labels on transitions.

$$\frac{}{\langle \texttt{skip}; P, \sigma\rangle \xrightarrow{\tau} \langle P, \sigma\rangle} \qquad \frac{a \in \sigma[\varsigma_w] \quad \sigma \models \mathrm{b} : \ell}{\langle a := \mathrm{b}, \sigma\rangle \xrightarrow{\tau} \langle \texttt{skip}, \sigma \oslash_{\varsigma} (\sigma[\varsigma] \oplus \{(a, \ell)\})\rangle}$$

$$\frac{\langle P, \sigma\rangle \xrightarrow{\delta} \langle P', \sigma'\rangle}{\langle P; Q, \sigma\rangle \xrightarrow{\delta} \langle P'; Q, \sigma'\rangle} \qquad \frac{\langle P, \sigma \oslash_{\varsigma_w} \varnothing\rangle \xrightarrow{\delta} \langle P', \sigma'\rangle}{\langle P || Q, \sigma\rangle \xrightarrow{\delta} \langle P' || Q, \sigma\rangle} \qquad \frac{\langle Q, \sigma \oslash_{\varsigma_w} \varnothing\rangle \xrightarrow{\delta} \langle Q', \sigma'\rangle}{\langle P || Q, \sigma\rangle \xrightarrow{\delta} \langle P || Q', \sigma\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i \quad \sigma \models \neg \mathrm{b}_1 \wedge \ \ldots \ \wedge \ \neg \mathrm{b}_{i-1} \ \wedge \neg \mathrm{b}_{i+1} \wedge \ \ldots \ \wedge \ \neg \mathrm{b}_n}{\langle [\mathrm{b}_1 \to P_1 \ [\!] \ \ldots \ [\!] \ \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle P_i, \sigma\rangle} \qquad \frac{\sigma \models \mathrm{b}_i}{\langle [\mathrm{b}_1 \to P_1 | \ \ldots \ | \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle P_i, \sigma\rangle}$$

$$\frac{\sigma \models \neg b_1 \wedge \ldots \wedge \neg b_n}{\langle *[\mathrm{b}_1 \to P_1 \ [\!] \ \ldots \ [\!] \ \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle \texttt{skip}, \sigma\rangle} \qquad \frac{\sigma \models \neg b_1 \wedge \ldots \wedge \neg b_n}{\langle *[\mathrm{b}_1 \to P_1 | \ \ldots \ | \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle \texttt{skip}, \sigma\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i \quad \sigma \models \neg \mathrm{b}_1 \wedge \ \ldots \ \wedge \ \neg \mathrm{b}_{i-1} \wedge \neg \mathrm{b}_{i+1} \wedge \ \ldots \ \wedge \ \neg \mathrm{b}_n}{\langle *[\mathrm{b}_1 \to P_1 \ [\!] \ \ldots \ [\!] \ \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle P_i; \ *[\mathrm{b}_1 \to P_1 \ [\!] \ \ldots \ [\!] \ \mathrm{b}_n \to P_n], \sigma\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i}{\langle *[\mathrm{b}_1 \to P_1 | \ \ldots \ | \mathrm{b}_n \to P_n], \sigma\rangle \xrightarrow{\tau} \langle P_i; \ *[\mathrm{b}_1 \to P_1 | \ \ldots \ | \mathrm{b}_n \to P_n], \sigma\rangle}$$

$$\frac{}{\langle \texttt{new}\ a\ P, \sigma\rangle \xrightarrow{\tau} \langle \underline{\texttt{new}\ a\ P}, \sigma \oslash_{\omega} (\sigma[\omega] \triangleleft \sigma \odot a)\rangle} \qquad \frac{}{\langle \underline{\texttt{new}\ a\ \texttt{skip}}, \sigma\rangle \xrightarrow{\tau} \langle \texttt{skip}, \sigma \oslash_{\omega} (\sigma[\omega] \triangleright)\rangle}$$

$$\frac{\sigma[\omega] \triangleright \sigma' \quad \langle P, \sigma'\rangle \xrightarrow{\delta} \langle P', \sigma''\rangle}{\langle \underline{\texttt{new}\ a\ P}, \sigma\rangle \xrightarrow{\delta} \langle \underline{\texttt{new}\ a\ P'}, \sigma'' \oslash_{\varsigma} (\sigma[\varsigma] \oplus_a \sigma''[\varsigma]) \oslash_{\varsigma_r} \sigma[\varsigma_r] \oslash_{\varsigma_w} \sigma[\varsigma_w] \oslash_{\omega} (\sigma[\omega] \diamond \sigma'')\rangle}$$

$$\frac{}{\langle \texttt{NEW}\ A\ P, \sigma\rangle \xrightarrow{\tau} \langle \underline{\texttt{NEW}\ A\ P}, \sigma \oslash_{\omega} (\sigma[\omega] \triangleleft \sigma \odot A)\rangle} \qquad \frac{}{\langle \underline{\texttt{NEW}\ A\ \texttt{skip}}, \sigma\rangle \xrightarrow{\tau} \langle \texttt{skip}, \sigma \oslash_{\omega} (\sigma[\omega] \triangleright)\rangle}$$

$$\frac{\sigma[\omega] \triangleright \sigma' \quad \langle P, \sigma'\rangle \xrightarrow{\delta} \langle P', \sigma''\rangle \quad A?a : \ell \notin \delta \quad A!\ell \notin \delta}{\langle \underline{\texttt{NEW}\ A\ P}, \sigma\rangle \xrightarrow{\delta} \langle \underline{\texttt{NEW}\ A\ P'}, \sigma'' \oslash_{\rho} \sigma[\rho] \oslash_{\rho_r} (\sigma''[\rho_r] \ominus_A \sigma[\rho_r]) \oslash_{\rho_w} (\sigma''[\rho_w] \ominus_A \sigma[\rho_w]) \oslash_{\omega} (\sigma[\omega] \diamond \sigma'')\rangle}$$

$$\frac{A \in \sigma[\rho] \quad a \in \sigma[\varsigma_w]}{\langle A?a, \sigma\rangle \xrightarrow{\tau} \langle \underline{A?a}, \sigma \oslash_{\rho_r} (\sigma[\rho_r] + A)\rangle} \qquad \frac{A \in \sigma[\rho] \quad \sigma \models \mathrm{b} : \ell}{\langle A!\mathrm{b}, \sigma\rangle \xrightarrow{\tau} \langle \underline{A!\ell}, \sigma \oslash_{\rho_w} (\sigma[\rho_w] + (A, \ell))\rangle}$$

$$\frac{}{\langle \underline{A?a}, \sigma\rangle \xrightarrow{\{A?a:\ell\}} \langle \texttt{skip}, \sigma \oslash_{\varsigma} (\sigma[\varsigma] \oplus \{(a, \ell)\}) \oslash_{\rho_r} (\sigma[\rho_r] - A)\rangle}$$

$$\frac{}{\langle \underline{A!\ell}, \sigma\rangle \xrightarrow{\{A!\ell\}} \langle \texttt{skip}, \sigma \oslash_{\rho_w} (\sigma[\rho_w] - (A, \ell))\rangle}$$

$$\frac{A \in \sigma[\rho] \quad a \in \sigma[\varsigma_w] \quad \langle C, \sigma\rangle \xrightarrow{\tau} \langle \underline{C}, \sigma'\rangle}{\langle A?a \bullet C, \sigma\rangle \xrightarrow{\tau} \langle \underline{A?a \bullet C}, \sigma' \oslash_{\rho_r} (\sigma'[\rho_r] + A)\rangle} \qquad \frac{A \in \sigma[\rho] \quad \sigma \models \mathrm{b} : \ell \quad \langle C, \sigma\rangle \xrightarrow{\tau} \langle \underline{C}, \sigma'\rangle}{\langle A!\mathrm{b} \bullet C, \sigma\rangle \xrightarrow{\tau} \langle \underline{A!\ell \bullet C}, \sigma' \oslash_{\rho_w} (\sigma'[\rho_w] + (A, \ell))\rangle}$$

$$\frac{\langle \underline{C}, \sigma\rangle \xrightarrow{\delta} \langle \texttt{skip}, \sigma'\rangle}{\langle \underline{A?a \bullet C}, \sigma\rangle \xrightarrow{\delta+\{A?a:\ell\}} \langle \texttt{skip}, \sigma' \oslash_{\varsigma} (\sigma'[\varsigma] \oplus \{(a,\ell)\}) \oslash_{\rho_r} (\sigma'[\rho_r] - A)\rangle}$$

$$\frac{\langle \underline{C}, \sigma\rangle \xrightarrow{\delta} \langle \texttt{skip}, \sigma'\rangle}{\langle \underline{A!\ell \bullet C}, \sigma\rangle \xrightarrow{\delta+\{A!\ell\}} \langle \texttt{skip}, \sigma' \oslash_{\rho_w} (\sigma'[\rho_w] - (A,\ell))\rangle}$$

$$\frac{\langle P, \sigma\rangle \xrightarrow{\{A!\ell\}} \langle P', \sigma'\rangle \quad \langle Q, \sigma'\rangle \xrightarrow{\{A?a:\ell\}} \langle Q', \sigma''\rangle}{\langle P||Q, \sigma\rangle \xrightarrow{\tau} \langle P'||Q', \sigma''\rangle} \qquad \frac{\langle P, \sigma\rangle \xrightarrow{\{A?a:\ell\}} \langle P', \sigma'\rangle \quad \langle Q, \sigma'\rangle \xrightarrow{\{A!\ell\}} \langle Q', \sigma''\rangle}{\langle P||Q, \sigma\rangle \xrightarrow{\tau} \langle P'||Q', \sigma''\rangle}$$

$$\frac{\begin{array}{c}\langle P, \sigma\rangle \xrightarrow{\delta} \langle P', \sigma'\rangle \quad \langle P'', \sigma'\rangle \xrightarrow{\delta'} \langle P''', \sigma''\rangle \quad \langle P||P'', \sigma\rangle \xrightarrow{\tau} \langle P'||P''', \sigma''\rangle \\ \langle Q, \sigma'''\rangle \xrightarrow{\delta+A?a:\ell} \langle Q', \sigma''''\rangle \quad \langle Q'', \sigma''''\rangle \xrightarrow{\delta'+A!\ell} \langle Q''', \sigma'''''\rangle\end{array}}{\langle Q||Q'', \sigma'''\rangle \xrightarrow{\tau} \langle Q'||Q''', \sigma'''''\rangle}$$

$$\frac{\begin{array}{c}\langle P, \sigma\rangle \xrightarrow{\delta} \langle P', \sigma'\rangle \quad \langle P'', \sigma'\rangle \xrightarrow{\delta'} \langle P''', \sigma''\rangle \quad \langle P||P'', \sigma\rangle \xrightarrow{\tau} \langle P'||P''', \sigma''\rangle \\ \langle Q, \sigma'''\rangle \xrightarrow{\delta+A!\ell} \langle Q', \sigma''''\rangle \quad \langle Q'', \sigma''''\rangle \xrightarrow{\delta'+A?a:\ell} \langle Q''', \sigma'''''\rangle\end{array}}{\langle Q||Q'', \sigma'''\rangle \xrightarrow{\tau} \langle Q'||Q''', \sigma'''''\rangle}$$

The labels need to be multisets, unless channels are end-to-end, in which case they just need to be sets.

2 HSE

One level lower is the handshaking expansions, which remove operations on channels, and operations on non-booleans.

2.1 Grammar

$$\begin{aligned}
&\ell ::= \top \mid \bot \\
&\mathrm{b} ::= a \mid \ell \mid \mathrm{b}_1 \wedge \mathrm{b}_2 \mid \mathrm{b}_1 \vee \mathrm{b}_2 \mid \neg \mathrm{b} \mid (\mathrm{b}) \\
&\mathrm{P} ::= a\uparrow \mid a\downarrow \mid S \mid *S \mid P_1; P_2 \mid P_1||P_2 \mid \texttt{skip} \mid (P) \\
&\mathrm{S} ::= [b_1 \to P_1 \,[]\, \ldots \,[]\, b_n \to P_n] \mid [b_1 \to P_1 | \ldots | b_n \to P_n]
\end{aligned}$$

Above, we are using $a...z$ to range over variables, and $A...Z$ to range over channels.
Additionally, there is some well-accepted syntactic sugar:
$*[S]$ is syntactic sugar for $*[\top \to S]$, which is used for infinite loops.
[b] is used to mean [b $\to$ skip], which means that we wait for a boolean expression to be true before continuing.

2.2 Small-Step Semantics

$$\frac{}{\langle \texttt{skip}; P, \sigma\rangle \to \langle P, \sigma\rangle}$$

$$\frac{}{\langle a\uparrow, \sigma\rangle \to \langle \texttt{skip}, \sigma[\top/a]\rangle} \quad \frac{}{\langle a\downarrow, \sigma\rangle \to \langle \texttt{skip}, \sigma[\bot/a]\rangle}$$

$$\frac{\langle P_1, \sigma\rangle \to \langle P_1', \sigma'\rangle}{\langle P_1; P_2, \sigma\rangle \to \langle P_1'; P_2, \sigma'\rangle} \quad \frac{\langle P_1, \sigma\rangle \to \langle P_1', \sigma'\rangle}{\langle P_1 || P_2, \sigma\rangle \to \langle P_1' || P_2, \sigma'\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i \quad \sigma \models \neg \mathrm{b}_1 \wedge \ ... \ \wedge \neg \mathrm{b}_{i-1} \wedge \neg \mathrm{b}_{i+1} \wedge \ ... \ \wedge \ \neg \mathrm{b}_n}{\langle [\mathrm{b}_1 \to P_1 \ [] \ ... \ [] \ \mathrm{b}_n \to P_n], \sigma\rangle \to \langle P_i, \sigma\rangle} \quad \frac{\sigma \models \mathrm{b}_i}{\langle [\mathrm{b}_1 \to P_1 | \ ... \ | \mathrm{b}_n \to P_n], \sigma\rangle \to \langle P_i, \sigma\rangle}$$

$$\frac{\sigma \models \neg b_1 \wedge ... \wedge \neg b_n}{\langle *[\mathrm{b}_1 \to P_1 \ [] \ ... \ [] \ \mathrm{b}_n \to P_n], \sigma\rangle \to \langle \texttt{skip}, \sigma\rangle} \quad \frac{\sigma \models \neg b_1 \wedge ... \wedge \neg b_n}{\langle *[\mathrm{b}_1 \to P_1 | \ ... \ | \mathrm{b}_n \to P_n], \sigma\rangle \to \langle \texttt{skip}, \sigma\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i \quad \sigma \models \neg \mathrm{b}_1 \wedge \ ... \ \wedge \neg \mathrm{b}_{i-1} \wedge \neg \mathrm{b}_{i+1} \wedge \ ... \ \wedge \ \neg \mathrm{b}_n}{\langle *[\mathrm{b}_1 \to P_1 \ [] \ ... \ [] \ \mathrm{b}_n \to P_n], \sigma\rangle \to \langle P_i; \ *[\mathrm{b}_1 \to P_1 \ [] \ ... \ [] \ \mathrm{b}_n \to P_n], \sigma\rangle}$$

$$\frac{\sigma \models \mathrm{b}_i}{\langle *[\mathrm{b}_1 \to P_1 | \ ... \ | \mathrm{b}_n \to P_n], \sigma\rangle \to \langle P_i; \ *[\mathrm{b}_1 \to P_1 | \ ... \ | \mathrm{b}_n \to P_n], \sigma\rangle}$$

3 PRS

The production rule set is the lowest non-circuit level. It represents the logical operations that happen in circuits.

3.1 Grammar

The production rules are represented as a set, Ξ, consisting of boolean guards, $G_{x\uparrow}$ and $G_{x\downarrow}$ for each variable x in the program. These rules take the form:

$$\begin{aligned} \mathrm{b} &::= a \mid l \mid \mathrm{b}_1 \wedge \mathrm{b}_2 \mid \mathrm{b}_1 \vee \mathrm{b}_2 \mid \neg \mathrm{b} \mid (\mathrm{b}) \\ G_{x\uparrow} &::= \mathrm{b} \mapsto x\uparrow \\ G_{x\downarrow} &::= \mathrm{b} \mapsto x\downarrow \end{aligned}$$

N.B. The choice for Ξ is arbitrary and not based on any previous literature. If there is actually a symbol that's used for the PR Set somewhere else, we should use that.

3.2 Semantics

$$\frac{G_{x\uparrow} \in \Xi \quad \sigma \models G_{x\uparrow}}{\langle \Xi, \sigma\rangle \to \langle \Xi, \sigma[\top / x]\rangle} \quad \frac{G_{x\downarrow} \in \Xi \quad \sigma \models G_{x\downarrow}}{\langle \Xi, \sigma\rangle \to \langle \Xi, \sigma[\bot / x]\rangle}$$

3.3 Properties

A few additional properties must be obeyed. First, the rule of non-interference, which states that it can never be the case that both the pull-up ($G_{x\uparrow}$) and pull-down $G_{x\downarrow}$ are active at the same time. This needs to be true for implementability – both guards being true corresponds to a transistor short-circuit.

Additionally, there is the rule of stability, which states that $G_{x\uparrow}$ can only change from true to false in states where x is true, and $G_{x\downarrow}$ can only change from false to true in states where x is false.

4 Naïve Synthesis

The synthesis process from CHP→HSE→PRS is an interesting problem. A naïve solution to it is presented below.

The five steps that we will follow in synthesis are:

1. CHP Generation
2. Handshaking Expansion
3. Ambiguous States Identification
4. State Variable Insertion
5. Production Rule Synthesis

4.1 CHP Generation

For this example, we are going to be using a simple one-place dataless buffer:

$$*[L?; R!]$$

This reads in a dataless value from L, and sends one out on R. It is not particularly useful, but it will serve for this example. A more complicated CHP program might be broken-up at this stage into several parallel programs, using the techniques of projection or process decomposition.

4.2 Handshaking Expansion

We will be using a four-phase handshake on both of the channels. Using this, the above expands to

$$*[l_o\uparrow;\ [l_i];\ l_o\downarrow;\ [\,l_i];\ [r_i];\ r_o\uparrow;\ [\,r_i];\ r_o\downarrow\,]$$

4.3 Ambiguous State Identification

To identify ambiguous states, we will first annotate the handshaking expansion with the states that it can be in at any point. Each point in the program will be annotated with a vector representing (l_o, l_i, r_o, r_i) We will be using 'X' to represent the times where we don't know if a value is true or false.

$$*[^{(000X)}\ l_o\uparrow^{(1X0X)};\ [l_i]^{(110X)};\ l_o\downarrow^{(010X)};\ [\,l_i]^{(000X)};\ [r_i]^{(0001)};\ r_o\uparrow^{(001X)};\ [\,r_i]^{(0010)};\ r_o\downarrow^{(000X)}\ ]$$

Each set of ambiguous states has been colored above. The red and green sets do not matter, as they are immediately before and after a wait, where we expect to see an ambiguity due to the X. However, the blue states are interesting since they're distributed throughout the program, and we would like different things to happen at each of them.

4.4 State Variable Insertion

To eliminate ambiguous states in the above handshaking expansion, we can insert a state variable, s. To demonstrate the removal of ambiguity, the vector will be updated, as (l_o, l_i, r_o, r_i, s).

$$*[^{(000X0)}\ l_o\uparrow^{(1X0X0)};\ [l_i]^{(110X0)};\ s\uparrow^{(110X1)};\ l_o\downarrow^{(010X1)};\ [\,l_i]^{(000X1)};$$
$$[r_i]^{(00011)};\ r_o\uparrow^{(001X1)};\ [\,r_i]^{(00101)};\ s\downarrow^{(00100)};\ r_o\downarrow^{(000X0)}\ ]$$

There are still ambiguous states after we have added in the state variables, however, all of them are either essentially the same state (such as in the case of the blue ambiguity) or do not matter (as in the case of the red and green ambiguity).

4.5 Production Rule Synthesis

There are two systematic methods for production rule generation, the first is to guard each and every action with the entirety of the (non-ambiguous) state at the moment it would like to fire, and then reducing the number of guards, and the second is to build up the guards by successively adding to them. Below we present the first method:

1. Initial Set

$$\neg l_o \wedge \neg l_i \wedge \neg r_o \wedge \neg s \mapsto l_o \uparrow \qquad \neg l_o \wedge \neg l_i \wedge \neg r_o \wedge r_i \wedge s \mapsto r_o \uparrow$$
$$l_o \wedge l_i \wedge \neg r_o \wedge s \mapsto l_o \downarrow \qquad \neg l_o \wedge \neg l_i \wedge r_o \wedge \neg r_i \wedge s \mapsto r_o \downarrow$$

$$l_o \wedge l_i \wedge \neg r_o \wedge \neg s \mapsto s \uparrow$$
$$\neg l_o \wedge \neg l_i \wedge r_o \wedge \neg r_i \wedge s \mapsto s \downarrow$$

2. First Reduction

   Every rule in the first set was "self-invalidating", meaning that the output of the rule appeared as part of the guard. As this is not possible to implement, this is removed as the first transformation. Doing this changes the states the rules can fire in subtlety – each of them now can also fire in the states where their result has occurred. This is OK.

$$\neg l_i \wedge \neg r_o \wedge \neg s \mapsto l_o \uparrow \qquad \neg l_o \wedge \neg l_i \wedge r_i \wedge s \mapsto r_o \uparrow$$
$$l_i \wedge \neg r_o \wedge s \mapsto l_o \downarrow \qquad \neg l_o \wedge \neg l_i \wedge \neg r_i \wedge s \mapsto r_o \downarrow$$

$$l_o \wedge l_i \wedge \neg r_o \mapsto s \uparrow$$
$$\neg l_o \wedge \neg l_i \wedge r_o \wedge \not{r_i} \mapsto s \downarrow$$

3. A Few More

   After a few more reductions, we get the set below. None of these rules can fire except in the states where they should, or in the states where they already hold.

$$\neg r_o \wedge \neg s \mapsto l_o \uparrow \qquad \neg l_i \wedge r_i \wedge s \mapsto r_o \uparrow$$
$$l_i \wedge \wedge s \mapsto l_o \downarrow \qquad \neg l_o \wedge s \mapsto r_o \downarrow$$

$$l_i \mapsto s \uparrow$$
$$\neg r_i \mapsto s \downarrow$$

4. Further Transformations

   As this production rule set exists, it is not CMOS-implementable. In order to be CMOS-implementable, each production rule of the form $G_{x\uparrow}$ must only use the inverted sense of variables in its guards, and each of the form $G_{x\downarrow}$ must use the non-inverted sense. Making this transformation may involve the insertion of additional variables, and is known as bubble reshuffling.

4.6 Discussion

The method above works for simple processes, however, it expands quickly when applied to large processes. It does not address the technique of handshake reshuffling, where the waits in the expanded handshakes are re-ordered to remove ambiguous states. This process is one where great care needs to be taken, as it can eliminate possible actions, and constrains other processes to needing compatible reshuffling.

5 Handshaking Reshuffling

The naïve synthesis method presented above will create handshaking expansions without ambiguous states, however, in many cases it will result in very inefficient implementations with a very large number of state variables.

To take the single-place buffer from above, we originally had:

$$*[l_o\uparrow;\ [l_i];\ l_o\downarrow;\ [\,l_i];\ [r_i];\ r_o\uparrow;\ [\,r_i];\ r_o\downarrow\,]$$

Which has ambiguous states in it. However, it is possible to reorder the sequence of events to not have any significant ambiguities while at the same time still preserving many of the possible sets of behaviors.

One such reshuffling is known as the "Weak Charge Half Buffered" or WCHB reshuffling, and it looks like:

$$*[l_o\uparrow;\ [r_i \wedge l_i]\,; r_o\uparrow;\ l_o\downarrow;\ [\neg r_i \wedge \neg l_i];\ r_o\downarrow\,]$$

6 CHP to CHP Transformations

To reduce the implementation complexity, and to increase the amount of parallelism and pipelining that is allowed, it is often advantageous to take a single CHP program and break it down into several smaller processes.

6.1 Process Decomposition

Process decomposition is a method for taking arbitrary sections of code and moving them into a separate process. It does this by replacing the code to remove with a single dataless channel action, and then using that channel action to synchronize the behavior.

For example, if the original CHP is $(....; S; ...)$, the decomposed CHP might be:

$$\begin{array}{l}\texttt{NEW}\ C;\\ ...; C!; ...\\ ||*[\bar{C} \rightarrow S; C?]\end{array}$$

This can be applied to more complicated processes, such as: (note that the following is from Prof. Manohar's class notes, and is using CHP with the extension onto numbers)

$$\begin{array}{l}*[x := 0; i := 0;\\ *[i < 10\ \rightarrow\ I?t; x := x + t; i := i + 1];\\ O!x\\ ]\end{array}$$

This is a process that reads in 10 items from channel I, and then sends out the summation of those on channel O. It might be helpful to move the summation on x out into a different channel.

$$\begin{array}{l}*[x := 0; i := 0;\\ \quad *[i < 10\ \rightarrow\ I?t; (A!||B!t)i := i + 1];\\ \quad\ O!x\\ ]\\ ||*[\bar{A}\ \rightarrow\ B?v; x := x + v; A?]\end{array}$$

A natural extension to this is to move all of the actions on x into the parallel process, as sharing variables becomes a headache once we get into implementation.

$$
\begin{array}{l}
*[C!;\ i := 0; \\
\quad *[i < 10\ \rightarrow\ I?t; (A!||B!t) i := i + 1]; \\
\quad D! \\
] \\
||*[\bar{A}\ \rightarrow\ B?v; x := x + v; A? \\
\quad []\bar{C}\ \rightarrow\ x := 0; B? \\
\quad []\bar{D}\ \rightarrow\ O!x; D? \\
]
\end{array}
$$

As the value from channel I is basically passing through B in all situations, we can replace it. This is only possible because the dataless channel A is ensuring that the value only gets read at the correct time.

$$
\begin{array}{l}
*[C!;\ i := 0; \\
\quad *[i < 10\ \rightarrow\ A!; i := i + 1]; \\
\quad D! \\
] \\
||*[\bar{A}\ \rightarrow\ I?v; x := x + v; A? \\
\quad []\bar{C}\ \rightarrow\ x := 0; B? \\
\quad []\bar{D}\ \rightarrow\ O!x; D? \\
]
\end{array}
$$

And lastly, as the selection statement will ensure the exclusivity of execution of all of its branches, and in this system, x is not shared, we can do some reordering of the actions in the second process, to allow the first process to proceed more rapidly.

$$
\begin{array}{l}
*[C!;\ i := 0; \\
\quad *[i < 10\ \rightarrow\ A!; i := i + 1]; \\
\quad D! \\
] \\
||*[\bar{A}\ \rightarrow\ A?; I?v; x := x + v \\
\quad []\bar{C}\ \rightarrow\ B?; x := 0 \\
\quad []\bar{D}\ \rightarrow\ O!x; D? \\
]
\end{array}
$$

And now, because of this decomposition, the additions on x and i can happen in parallel, the processes will be simpler to implement in hardware, and will perform better.

7 Project Goals

1. Formally define the semantics for CHP, HSE, and PRS
2. Discuss Handshaking Expansions in a good way

3. Discuss Synthesis from HSE to PRS in a better way
4. Build a system to prove CHP $\rightarrow$ CHP transformations
5. Build a system to prove HSE $\rightarrow$ HSE transformations
6. Build a system to prove equivalences for CHP $\rightarrow$ HSE transforms post-reshufflings