EENG 426/CPSC 459/ENAS 876
Silicon Compilation

CHP Examples

Computer Systems Lab
http://csl.yale.edu/~rajit

Fall 2018

Yale AVLSI

Copy and alternator

Copy:

$*[\ X?a;\ \ Y!a, Z!a\ ]$

Alternator:

$*[\ \ X?a; Y!a;\ \ X?a; Z!a\ ]$

Yale AVLSI

Controlled split and merge

Split:

$*[\ C?c; X?a;$
$[c \longrightarrow Y!a\ [\!]\ \neg c \longrightarrow Z!a\ ]$
$]$

Merge:

$*[\ C?c;\ \ [c \longrightarrow X?a\ [\!]\ \neg c \longrightarrow Y?a];$
$Z!a$
$]$

Yale AVLSI

Reactive process structure

$*[[G_1 \longrightarrow S_1$
$[\!]\ G_2 \longrightarrow S_2$
$[\!]\ \ldots$
$[\!]\ G_n \longrightarrow S_n$
$]]$

This process structure is used very often.

- Wait for some action to be enabled
- Execute that action
- Repeat

Yale AVLSI

Construction a lazy stack

The problem: construct a last-in first-out structure with capacity N.

Environment:

- insert: $push!x$
- remove: $pop?x$
- operations are mutually exclusive

Program is allowed to fail when attempting to insert into a full stack, or remove from an empty stack.

Sequential program

Using an array to store the values in the stack, we can solve the problem as follows:

$$
\begin{array}{l}
\quad n := 0; \\
*[[\overline{push} \wedge n < N \longrightarrow push?x[n]; n := n+1 \\
\quad [] \overline{pop} \wedge n > 0 \longrightarrow pop!x[n-1]; n := n-1 \\
\quad]]
\end{array}
$$

Invariant:
$x[0..n-1]$ are the elements stored in the stack.

$n > 0 \Rightarrow x[n-1]$ is the last element that was inserted.

Recursive construction

To add concurrency, we would like to construct the stack as the parallel composition of a number of processes.

We construct an N-place stack by assuming the existence of a $(N-1)$-place stack.

Stack element:

- *push*, *pop*: environment interface
- *put*, *get*: interface to the smaller stack

(This type of construction is used quite often.)

Base case

$N = 1$:

$$
\begin{array}{l}
*[[\overline{push} \longrightarrow push?x \\
\quad [] \overline{pop} \longrightarrow pop!x \\
\quad]]
\end{array}
$$

The "rest of the stack" has no storage.

Base case

Another possibility:

$$\begin{array}{l} \{stack\ \ is\ \ empty\} \\ *[\ \ \{stack\ \ is\ \ empty\} \\ \quad push?x; \\ \quad \{stack\ \ is\ \ full\} \\ \quad pop!x \\ ] \end{array}$$

What happens when the stack overflows? Underflows?

Stack element

1. Assume the stack element is empty:

$$\begin{array}{l} [\overline{push} \longrightarrow push?x\ \ \{full\} \\ []\overline{pop} \longrightarrow get?x; pop!x\ \ \{empty\} \\ ] \end{array}$$

2. Assume the stack element is full:

$$\begin{array}{l} [\overline{push} \longrightarrow put!x; push?x\ \ \{full\} \\ []\overline{pop} \longrightarrow pop!x\ \ \{empty\} \\ ] \end{array}$$

Stack element

$b \equiv$ the stack element is empty

$$\begin{array}{l} b\uparrow; \\ *[[b \wedge \overline{push} \longrightarrow push?x; b\downarrow \\ \ \ []b \wedge \overline{pop} \longrightarrow get?x; pop!x \\ \ \ []\neg b \wedge \overline{push} \longrightarrow put!x; push?x \\ \ \ []\neg b \wedge \overline{pop} \longrightarrow pop!x; b\uparrow \\ ]] \end{array}$$

Stack element

Another implementation that is equivalent:

$$\begin{array}{l} *[\{empty\} \\ \quad [\overline{push} \longrightarrow push?x \\ \quad []\overline{pop} \longrightarrow get?x \\ \quad]; \\ \quad \{full\} \\ \quad [\overline{push} \longrightarrow put!x \\ \quad []\overline{pop} \longrightarrow pop!x \\ \quad] \\ ] \end{array}$$