CS 536 Class 27
Tue 5/4
Await/Synchronization

We've seen how to avoid problems of interference.
Avoid bad things.

For threads to cooperate, they need to synchronize occasionally.
Thread 1 waits until thread 2 sets x to 7.

await 13 then send
   if no awaits, loops, or
   S is executed atomically in atomic regions
   when B becomes true
conditional atomic region
If > 1 thread is waiting for same B, only one is woken, chosen non-deterministically.

No thread runs between noticing B is true and starting S

If \( \sigma(B) = T \) \(<\text{await} B \text{ then } S \text{ end}, 0>\)
and \(<S,0> \rightarrow <E,t>\)
\(\begin{array}{l}
(b \text{ is true}) \times b \text{ false}\n\end{array}\)

Nothing happens to the wait while B is false

Thread is blocked if at \text{await} B \text{ then } S \text{ end}
but B is false.

\text{compare} \quad \text{await} B \text{ then } S \text{ end} \quad B \text{ false}?

if B then S fi \quad B \text{ true}?

B false? Jump to end of if

B true interleaving allowed before starting S

\(<\text{if } B \text{ then } S \text{ fi, } 0>\) \(\sigma(B) = T\)
\(<S,0>\)
Abbreviations

atomic $\langle S \rangle \overset{\top}{=} \text{wait true then Send}$

wait $B = \text{await } B \text{ then skip end}$

wait $B; S = \langle \text{wait } B; S; \sigma \rangle$

$\langle \text{await } B \text{ then Send}, \sigma \rangle \overset{\top}{=} \langle S, \sigma \rangle$

$\langle \text{if } B \text{ true}, \sigma \rangle \overset{\top}{=} \langle x, \sigma \rangle$

$\langle \text{else } \tau, \sigma \rangle \overset{\top}{=} \langle \epsilon, \tau \rangle$

Outlines

$\{p \exists \} \text{ await } B \text{ then}$

$\{ p \times B^3 \}$

$\langle S^* \rangle$ and $\langle q^3 \rangle$

$\text{wp}(\text{await } B \text{ then Send}, q)$

$= B \rightarrow \text{wp}(S, q)$

Never use $\{ p \exists \} \text{ await } B \text{ then}$

requires $\{ p \exists B^3 \text{ await } B \text{ then } \{ p \exists B^3 \text{ wait } B \text{ end } \}$

requires $\{ p \exists B^3 \text{ await } B \text{ then } \{ p \exists B^3 \text{ wait } B \text{ end } \}$

$\langle B^3 \rangle$
Producer/Consumer problem

1 thread produces items, puts them into finite buffer

1 thread removes items from buffer, uses it

If buffer is empty, consumer must wait.
If buffer is full, producer must wait.

Can view problem symmetrically if roles
"producer" and "consumer" change

"Consumer" produces items, etc.
Init empty buffer ()

while ~done do
    thing = Create ()
    await NotFull Buffer ()
    Buffer Add (thing)
end

while ~done do
    await NotEmpty Buffer ()
    thing = Buffer Remove ()
    consume (thing)
end

Buffer code
Init Empty Buffer ()

<Buffer = empty; Not Full Buffer = T;
Not Empty Buffer = F>

Buffer Add (t)

<Buffer = Buffer + t;
if Buffer full Not Full Buffer = F, fi
Not Empty Buffer = T>

Buffer Remove ()

<x = Buffer minus entry;
Buffer Not Full = T
if Buffer empty then Buffer Not Empty = F>
Note: buffer implementation uses synch. atomic regions too.

Synchronization add + remove avoid critical sections

Mutual exclusion using await

await critsectok then $; critsectok =

atomic c.s. code

T end

nonatomic c.s.

await critsectok $; critsectok =

then c.s. ok = F

end

Deadlock: multiple threads

$1$ waiting (blocked)

other threads complete

Waiting code never unblocks everyone else is blocked or complete

$\{y\}$

I await y $=$ 0 then $x$ $=$ 1 end;

I await $x$ $=$ 0 then $y$ $=$ 1 end;

$\Rightarrow x = 0 \land y = 0$ causes deadlock

$\Rightarrow x \neq 0 \lor y \neq 0$ doesn't
D. Lock - A program might deadlock

along only 1 exec. path w/ only 1 state

or > 1 path or > 1 state

or every path or every state

In these situations, D.L. might not occur

---

We can specify a set of D.L. conditions (sufficient to guarantee no D.L.)

so that if all D.L. conditions are \( \iff \) P

then deadlock will never occur

Each D.L. condition specifies one way in which D.L. might conceivably occur

But D.L. condition \( \iff \) P doesn't guarantee D.L.

The conditions aren't necessary always
Parallel program $[S_1 || S_2 || \ldots || S_n]$

A potential D.L. condition is $r_1 \land r_2 \land \ldots \land r_n$ (one for each thread)

$r_i$ can be postcondition of $S_i$

$r_i$ can be $p \Rightarrow B$ where $\{p\}$ await $B$ is in $S_i$

At least one $r_i$ has to be a blockage (i.e. all threads are complete)

To prove D.L. freedom of parallel program

Show that every D.L condition $p$ for $p_{\text{for}}$

it's sufficient to show $\neg p \Rightarrow T$

Use $p$ as precondition

D.L. condition $1^* (p \land y = 0) \land q_2$

D.L. $\land 2^* (q_1 \land (p \land x = 0))$ Both threads

D.L. $\land 3^* (p \land y = 0) \land (q \land x = 0)$ blocked

The complete
1: \( y = o \times T \) if we ignore \( p \) and \( q_1 = q_2 = T \)
2: \( T \times y = 0 \)
3: \( y = x \times 0 \)

1: \( p \times y = 0 \) if we put back \( p \)
2: \( p \times x = 0 \)
3: \( p \times x = o \times y = 0 \)

To avoid D.L. we need \( p \times (y = 0 \lor x = 0) \rightarrow F \Rightarrow (a) \)

By definition \( a \rightarrow b \Leftrightarrow a \lor b \)

(a) \( \Rightarrow (p \times (x = o \lor y = o)) \lor F \Rightarrow (b) \) using de Morgan

(b) \( \Leftrightarrow \neg p \lor (x \neq o \land y \neq o) \)

Need, \( \neg p \Leftrightarrow \neg (x \neq o \land y \neq o) \):

\[ p \Rightarrow x \neq o \lor y \neq o \] avoids D.L.

\[ \begin{align*}
\text{await } y & \neq 0 \\
\text{await } x & \neq 0
\end{align*} \]