Time Dependence
Time Dependence

\[ \text{deadline} := t + 5 \]
Time Dependence

\[ \text{deadline} := t + 5 \]  
no problem
Time Dependence

deadline := t + 5

if $t < \text{deadline}$ then ... else ... fi

no problem
Time Dependence

\[ \text{deadline} := t + 5 \]

\[ \text{if } t < \text{deadline} \text{ then } \ldots \text{ else } \ldots \text{ fi} \]

no problem
Time Dependence

\[ \text{deadline} := t + 5 \]

\[ \text{if } t < \text{deadline then } \ldots \text{ else } \ldots \text{ fi} \]

\[ t := 5 \]

no problem

no problem

problem: unimplementable
Time Dependence

\[ \text{deadline} := t + 5 \]

\[ \text{if } t < \text{deadline} \text{ then } \ldots \text{ else } \ldots \text{ fi} \]

\[ t := 5 \]

\[ \text{wait until } w \]

no problem

no problem

problem: unimplementable
Time Dependence

\[
\text{deadline} := t + 5 \\
\text{if } t < \text{deadline} \text{ then } ... \text{ else } ... \text{ fi} \\
t := 5 \\
\text{wait until } w = t := t↑w
\]

no problem

no problem

problem: unimplementable
Time Dependence

\[ \text{deadline} := t + 5 \]

if \( t < \text{deadline} \) then ... else ... fi

\[ t := 5 \]

wait until \( w = t \uparrow w \)

no problem

no problem

problem: unimplementable

busy-wait loop
Time Dependence

\[ \text{deadline} := t + 5 \quad \text{no problem} \]
\[ \text{if } t < \text{deadline} \text{ then } ... \text{ else } ... \text{ fi} \quad \text{no problem} \]
\[ t := 5 \quad \text{problem: unimplementable} \]
\[ \text{wait until } w = t \uparrow w \quad \text{busy-wait loop} \]
\[ \text{wait until } w \Leftarrow \text{ if } t \geq w \text{ then ok else } t := t + 1. \text{ wait until } w \text{ fi} \]
Time Dependence

\[ \text{deadline} := t + 5 \]

\[ \text{if } t < \text{deadline} \quad \text{then} \quad \ldots \quad \text{else} \quad \ldots \quad \text{fi} \]

no problem

\[ t := 5 \]

problem: unimplementable

\[ \text{wait until } w = t := t \uparrow w \]

busy-wait loop

\[ \text{wait until } w \iff \text{if } t \geq w \quad \text{then ok} \quad \text{else} \quad t := t + 1. \quad \text{wait until } w \quad \text{fi} \]

proof
Time Dependence

\[
\text{deadline} := t + 5 \\
\text{if } t < \text{deadline} \text{ then } ... \text{ else } ... \text{ fi} \\
t := 5 \\
\text{wait until } w \implies t := t \uparrow w \\
\text{wait until } w \iff \text{if } t \geq w \text{ then ok else } t := t + 1. \text{ wait until } w \text{ fi}
\]

\textbf{proof}

\[t \geq w \land \text{ok}\]
Time Dependence

deadline:= t + 5

if t < deadline then ... else ... fi

no problem

t:= 5

problem: unimplementable

wait until w = t↑w

busy-wait loop

wait until w ⇐ if t≥w then ok else t:= t+1. wait until w fi

proof

\( t ≥ w \land ok \)

= \( t ≥ w \land (t := t) \)
Time Dependence

declaration: $t = t + 5$

no problem

if $t < \text{deadline}$ then ... else ... fi

no problem

$t := 5$

problem: unimplementable

wait until $w = t \uparrow w$

busy-wait loop

wait until $w \Leftarrow$ if $t \geq w$ then ok else $t := t + 1$. wait until $w$ fi

proof

$t \geq w \land ok$

$= t \geq w \land (t := t)$

$= t \geq w \land (t := t \uparrow w)$
Time Dependence

deadline := t + 5

if $t < \text{deadline}$ then ... else ... fi

no problem

t := 5

problem: unimplementable

wait until $w = t \uparrow w$

busy-wait loop

wait until $w \iff \text{if } t \geq w \text{ then ok else } t := t + 1 \text{. } \text{wait until } w \text{ fi}$

proof

$t \geq w \land ok$

$= t \geq w \land (t := t)$

$= t \geq w \land (t := t \uparrow w)$

$\Rightarrow \text{wait until } w$
Time Dependence

deadline := t + 5

if $t < \text{deadline}$ then ... else ... fi

$t := 5$

no problem

problem: unimplementable

wait until $w = t \uparrow w$

busy-wait loop

wait until $w \iff \text{if } t \geq w \text{ then ok else } t := t + 1. \text{ wait until } w \text{ fi}$

proof
Time Dependence

deadline := t + 5

if $t < \text{deadline}$ then ... else ... fi

$t := 5$

wait until $w = t \uparrow w$

wait until $w \Leftarrow$ if $t \geq w$ then ok else $t := t+1$. wait until $w$ fi

proof

$t < w \land (t := t+1. \text{ wait until } w)$
Time Dependence

\[ \text{deadline} := t + 5 \]

\[ \text{if } t < \text{deadline} \text{ then } \ldots \text{ else } \ldots \text{ fi} \]

\[ t := 5 \]

no problem

problem: unimplementable

\[ \text{wait until } w \quad = \quad t := t \uparrow w \]

busy-wait loop

\[ \text{wait until } w \quad \Leftarrow \quad \text{if } t \geq w \text{ then ok else } t := t+1. \quad \text{wait until } w \text{ fi} \]

proof

\[ t < w \land (t := t+1. \; \text{wait until } w) \]

\[ = \quad t < w \land (t := t+1. \; t := t \uparrow w) \]
### Time Dependence

\[
\text{deadline} := t + 5 \quad \text{no problem}
\]

\[
\text{if } t < \text{deadline} \text{ then } \ldots \text{ else } \ldots \text{ fi} \quad \text{no problem}
\]

\[
t := 5 \quad \text{problem: unimplementable}
\]

\[
\text{wait until } w = t \uparrow w \quad \text{busy-wait loop}
\]

\[
\text{wait until } w \iff \text{if } t \geq w \text{ then ok else } t := t + 1. \text{ wait until } w \text{ fi}
\]

### proof

\[
t < w \land (t := t + 1. \text{ wait until } w)
\]

\[
= t < w \land (t := t + 1. \ t := t \uparrow w)
\]

\[
= t + 1 \leq w \land
\]
Time Dependence

\[\text{deadline} := t + 5\] no problem

\textbf{if} \ t < \text{deadline} \ \textbf{then} ... \ \textbf{else} ... \ \textbf{fi} no problem

\[t := 5\] problem: unimplementable

\textbf{wait until} \ w \ = \ t := t^{\uparrow}w\] busy-wait loop

\textbf{wait until} \ w \ \Leftarrow \ \textbf{if} \ t \geq w \ \textbf{then} \ ok \ \textbf{else} \ t := t+1. \ \textbf{wait until} \ w \ \textbf{fi}

\textbf{proof}

\[t < w \land (t := t+1. \ \textbf{wait until} \ w)\]

\[= t < w \land (t := t+1. \ t' = t^{\uparrow}w \land \ldots)\]

\[= t+1 \leq w \land \ldots\]
Time Dependence

deadline := t + 5

if \( t < \) deadline then ... else ... fi

\( t := 5 \)

wait until \( w = t \uparrow w \)

wait until \( w \Leftarrow \) if \( t \geq w \) then ok else \( t := t+1 \). wait until \( w \) fi

proof

\[ t < w \land (t := t+1. \text{ wait until } w) \]

\[ = t < w \land (t' = (t+1) \uparrow w \land ...) \]

\[ = t + 1 \leq w \land \]
Time Dependence

\[
\text{deadline} := t + 5 \quad \text{no problem}
\]
\[
\text{if } t < \text{deadline} \text{ then } ... \text{ else } ... \text{ fi} \quad \text{no problem}
\]
\[
t := 5 \quad \text{problem: unimplementable}
\]
\[
\text{wait until } w = t := t \uparrow w \quad \text{busy-wait loop}
\]
\[
\text{wait until } w \Leftarrow \text{ if } t \geq w \text{ then } \text{ok} \text{ else } t := t + 1. \text{ wait until } w \text{ fi}
\]

**proof**

\[
t < w \land (t := t + 1. \text{ wait until } w)
\]
\[
= t < w \land (t := t + 1. t := t \uparrow w)
\]
\[
= t + 1 \leq w \land (t := (t + 1) \uparrow w)
\]
Time Dependence

deadline := t + 5

if \( t < \) deadline then ... else ... fi

t := 5

wait until \( w = t := t \uparrow w \)

wait until \( w \iff \) if \( t \geq w \) then ok else \( t := t + 1 \). wait until \( w \) fi

proof

\[
\begin{align*}
t < w \land (t := t + 1. \text{ wait until } w) \\
= & \quad t < w \land (t := t + 1. \ t := t \uparrow w) \\
= & \quad t + 1 \leq w \land (t := (t + 1) \uparrow w) \\
= & \quad t < w \land (t := w)
\end{align*}
\]
Time Dependence

deadline := t + 5

\textbf{if} t < \textit{deadline} \textbf{then} ... \textbf{else} ... \textbf{fi}

\texttt{t := 5}

\texttt{wait until} \ w \ \ \ = \ \ \ t := t \uparrow w

\texttt{wait until} \ w \ \ \leftarrow \ \ \textbf{if} \ t \geq w \ \textbf{then} \ ok \ \textbf{else} \ t := t+1. \ \texttt{wait until} \ w \ \textbf{fi}

\textbf{proof}

\( t < w \land (t := t+1. \ \texttt{wait until} \ w) \)

\( = \)

\( t < w \land (t := t+1. \ t := t \uparrow w) \)

\( = \)

\( t + 1 \leq w \land (t := (t+1) \uparrow w) \)

\( = \)

\( t < w \land (t := w) \)

\( = \)

\( t < w \land (t := t \uparrow w) \)
Time Dependence

\[ \text{deadline}: = t + 5 \]

if \( t < \text{deadline} \) then ... else ... fi

no problem

\[ t: = 5 \]

problem: unimplementable

wait until \( w \) \( = \) \( t: = t \uparrow w \)

busy-wait loop

\[ \text{wait until } w \iff \text{if } t \geq w \text{ then ok else } t: = t + 1. \text{ wait until } w \text{ fi} \]

proof

\[ t < w \land (t: = t + 1. \text{ wait until } w) \]

\[ = \]

\[ t < w \land (t: = t + 1. \ t: = t \uparrow w) \]

\[ = \]

\[ t + 1 \leq w \land (t: = (t + 1) \uparrow w) \]

\[ = \]

\[ t < w \land (t: = w) \]

\[ = \]

\[ t < w \land (t: = t \uparrow w) \]

\[ \Rightarrow \]

\[ \text{wait until } w \]
Space Dependence
Space Dependence

\textbf{if } s < 1000000 \textbf{ then ... else ... fi } \quad \text{no problem}
Space Dependence

\[
\text{if } s < 1000000 \text{ then ... else ... fi}
\]

\[
s := 5
\]

no problem

problem
Space Dependence

\[
\begin{align*}
&\text{if } s < 1000000 \text{ then } \ldots \text{ else } \ldots \text{ fi} \\
&s := 5
\end{align*}
\]

no problem

problem

assignments to $s$ must account for space
Space Dependence

\[
\text{if } s < 1000000 \text{ then ... else ... fi}
\]
\[
s := 5
\]

no problem

problem

assignments to \( s \) must account for space

real space

implementation dependent
Assertions
assert $b$
Assertions

```plaintext
assert b

= “I believe b is true”
```
Assertions

\texttt{assert } b

\begin{align*}
\text{=} & \quad \text{“I believe } b \text{ is true”} \\
\text{=} & \quad \text{precondition } b
\end{align*}
assert $b$

$=$

“I believe $b$ is true”

$=$

precondition $b$

$=$

postcondition $b$
assert $b$

= “I believe $b$ is true”
= precondition $b$
= postcondition $b$
= invariant $b$
Assertions

assert \, b

=  “I believe \, b \, is true”

=  precondition \, b

=  postcondition \, b

=  invariant \, b

=  if \, b \, then \, ok \, else \, print \, “error: ...” \, . \, wait \, until \, ∞ \, fi
Assertions

assert $b$

= “I believe $b$ is true”

= precondition $b$

= postcondition $b$

= invariant $b$

= if $b$ then ok else print “error: ...”. wait until $\infty$ fi

redundant
Assertions

assert $b$

= “I believe $b$ is true”

= precondition $b$

= postcondition $b$

= invariant $b$

= if $b$ then ok else print “error: ...”. wait until $\infty$ fi

redundant, adds robustness
**Assertions**

```plaintext
assert \( b \)
```

= “I believe \( b \) is true”

= precondition \( b \)

= postcondition \( b \)

= invariant \( b \)

= if \( b \) then ok else print “error: …”. wait until \( \infty \) fi

redundant, adds robustness, costs execution time
Assertions

assert $b$

= “I believe $b$ is true”

= precondition $b$

= postcondition $b$

= invariant $b$

= if $b$ then ok else print “error: ...”. wait until $\infty$ fi

redundant, adds robustness, costs execution time

ensure $b$
Assertions

assert \( b \)

= “I believe \( b \) is true”

= precondition \( b \)

= postcondition \( b \)

= invariant \( b \)

= if \( b \) then ok else print “error: ...”. \textbf{wait until} \( \infty \) fi

redundant, adds robustness, costs execution time

ensure \( b \)

= “make \( b \) be true without doing anything”
assert \( b \)

= “I believe \( b \) is true”

= precondition \( b \)

= postcondition \( b \)

= invariant \( b \)

= if \( b \) then \( ok \) else print “error: ... ”. wait until \( \infty \) fi

redundant, adds robustness, costs execution time

ensure \( b \)

= “make \( b \) be true without doing anything”

= if \( b \) then \( ok \) else \( b' \land ok \) fi
Assertions

assert $b$

= “I believe $b$ is true”

= precondition $b$

= postcondition $b$

= invariant $b$

= if $b$ then ok else print “error: ...”. wait until $\infty$ fi

redundant, adds robustness, costs execution time

ensure $b$

= “make $b$ be true without doing anything”

= if $b$ then ok else $b' \land ok$ fi

= $b' \land ok$
Assertions

assert \( b \)

= “I believe \( b \) is true”

= precondition \( b \)

= postcondition \( b \)

= invariant \( b \)

= if \( b \) then ok else print “error: ...”. wait until \( \infty \) fi

redundant, adds robustness, costs execution time

ensure \( b \)

= “make \( b \) be true without doing anything”

= if \( b \) then ok else \( b' \land ok \) fi

= \( b' \land ok \)

unimplementable
Assertions

assert \( b \)

= “I believe \( b \) is true”

= precondition \( b \)

= postcondition \( b \)

= invariant \( b \)

= if \( b \) then \( \text{ok} \) else print “error: ...”. \( \text{wait until } \infty \) fi

redundant, adds robustness, costs execution time

ensure \( b \)

= “make \( b \) be true without doing anything”

= if \( b \) then \( \text{ok} \) else \( b' \land \text{ok} \) fi

= \( b' \land \text{ok} \)

unimplementable by itself, but may be used in some contexts
nondeterministic choice

\[ P \lor Q \]
nondeterministic choice (a programming notation):

\[ P \lor Q \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]
**nondeterministic choice** (a programming notation):

\[ P \text{ or } Q \quad = \quad P \lor Q \]

\[ x := 0 \text{ or } x := 1 \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q \quad = \quad P \lor Q \]

\[ x := 0 \text{ or } x := 1 \]

\[ = \quad x' = 0 \land y' = y \lor x' = 1 \land y' = y \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q \quad = \quad P \lor Q \]

\[ x := 0 \text{ or } x := 1. \quad \text{ensure } x = 1 \]

\[ = \quad x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\( x := 0 \text{ or } x := 1. \text{ ensure } x = 1 \)

\[ = \quad x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \]

\[ = \quad \exists x'', y''. \quad (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\[ x := 0 \text{ or } x := 1. \text{ ensure } x=1 \]

\[ = x' = 0 \land y' = y \lor x' = 1 \land y' = y. \text{ } x' = 1 \land x' = x \land y' = y \]

\[ = \exists x'', y''. \ (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\[ x := 0 \text{ or } x := 1. \quad \text{ensure } x = 1 \]

\[ = \]

\[ x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \]

\[ = \]

\[ \exists x'', y''. \quad (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \]
nondeterministic choice (a programming notation):

\[ P \lor Q = P \lor Q \]

\[ x := 0 \text{ or } x := 1. \quad \textbf{ensure} \ x = 1 \]

\[ = \ x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \]

\[ = \exists x'', y''. \quad (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \]

\[ = \ (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1 \]
**nondeterministic choice** (a programming notation):

\[ P \text{ or } Q \quad = \quad P \lor Q \]

\[
\begin{align*}
x &:= 0 \text{ or } x := 1. \quad \textbf{ensure} \ x = 1 \\
= & \quad x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \\
= & \quad \exists x'', y''. \ (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \\
= & \quad (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1 \\
= & \quad x' = 1 \land y' = y
\end{align*}
\]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\[ x := 0 \text{ or } x := 1. \quad \text{ensure } x = 1 \]

\[ = x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y \]

\[ = \exists x'', y''. \ (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \lor y' = y'' \]

\[ = (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1 \]

\[ = x' = 1 \land y' = y \]

\[ = x := 1 \]
**nondeterministic choice** (a programming notation):

\[
P \text{ or } Q = P \lor Q
\]

\[
x := 0 \text{ or } x := 1. \quad \textbf{ensure} \ x = 1
\]

\[
= \quad x' = 0 \land y' = y \lor x' = 1 \land y' = y. \quad x' = 1 \land x' = x \land y' = y
\]

\[
= \exists x'', y''. \ (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y''
\]

\[
= \quad (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1
\]

\[
= \quad x' = 1 \land y' = y
\]

\[
= \quad x := 1
\]

implementation: **backtracking**
nondeterministic choice (a programming notation):

\[ P \text{ or } Q \quad = \quad P \lor Q \]

\[ x := 0 \text{ or } x := 1 \quad \textbf{ensure} \quad x = 1 \]

\[ = \quad x' = 0 \land y = y \lor x' = 1 \land y = y \land x' = x \land y' = y \]

\[ = \quad \exists x'', y''. \ (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y'' = y'' \]

\[ = \quad (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1 \]

\[ = \quad x' = 1 \land y' = y \]

\[ = \quad x := 1 \]

implementation: backtracking

natural square root  Given natural \( n \) find natural \( s \) satisfying \( s^2 \leq n < (s+1)^2 \)
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\[ x := 0 \text{ or } x := 1. \text{ ensure } x = 1 \]

\[ = x' = 0 \land y' = y \lor x' = 1 \land y' = y. \land x' = 1 \land x' = x \land y' = y \]

\[ = \exists x'', y''. (x'' = 0 \land y'' = y \lor x'' = 1 \land y'' = y) \land x' = 1 \land x' = x'' \land y' = y'' \]

\[ = (x' = 0 \land y' = y \lor x' = 1 \land y' = y) \land x' = 1 \]

\[ = x' = 1 \land y' = y \]

\[ = x := 1 \]

implementation: backtracking

natural square root Given natural \( n \) find natural \( s \) satisfying \( s^2 \leq n < (s+1)^2 \)

\[ s := 0,..n+1 \]
nondeterministic choice (a programming notation):

\[ P \text{ or } Q = P \lor Q \]

\[
x := 0 \text{ or } x:= 1. \text{ ensure } x=1
\]

\[
= x'=0 \land y'=y \lor x'=1 \land y'=y. \ x'=1 \land x'=x \land y'=y
\]

\[
= \exists x'', y''. \ (x''=0 \land y''=y \lor x''=1 \land y''=y) \land x'=1 \land x'=x'' \land y'=y''
\]

\[
= (x'=0 \land y'=y \lor x'=1 \land y'=y) \land x'=1
\]

\[
= x'=1 \land y'=y
\]

\[
= x:= 1
\]

implementation: backtracking

natural square root Given natural \( n \) find natural \( s \) satisfying \( s^2 \leq n < (s+1)^2 \)

\[
s:= 0,..n+1. \text{ ensure } s^2 \leq n < (s+1)^2
\]