Time Dependence

Time Dependence

$deadline := t + 5$

Time Dependence

$deadline := t + 5$ no problem

Time Dependence

$deadline := t + 5$ no problem

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

Time Dependence

deadline := $t + 5$	no problem
if $t < \mathit{deadline}$ **then** ... **else** ... **fi**	no problem

Time Dependence

deadline:= *t* + 5	no problem
if *t* < *deadline* **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable

Time Dependence

deadline:= t + 5	no problem
if t < *deadline* **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable
wait until w	

Time Dependence

deadline:= $t + 5$	no problem
if $t <$ *deadline* **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable
wait until w **=** $t := t \uparrow w$	

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

Time Dependence

deadline:= $t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable
wait until w = t:= $t{\uparrow}w$	busy-wait loop

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

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w = $t := t \uparrow w$	busy-wait loop

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

proof

Time Depemdence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$t \geq w \land ok$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$$t \geq w \wedge ok$$

$$= \quad t \geq w \wedge (t := t)$$

Time Depedence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$$t \ge w \wedge ok$$

$$= \quad t \ge w \wedge (t := t)$$

$$= \quad t \ge w \wedge (t := t \uparrow w)$$

Time Dependence

deadline:= $t + 5$	no problem
if $t <$ *deadline* **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable
wait until w = t:= $t{\uparrow}w$	busy-wait loop

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

proof

$t \geq w \land ok$

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

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

⇒ **wait until** w

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

Time Dependence

deadline:= t + 5	no problem
if t < *deadline* **then** ... **else** ... **fi**	no problem
t:= 5	problem: unimplementable
wait until w = t:= $t\uparrow w$	busy-wait loop

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

proof

$t<w \land$ (t:= t+1. **wait until** w)

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$t<w \wedge (t := t+1.$ **wait until** $w)$

$= \quad t<w \wedge (t := t+1.\ t := t \uparrow w)$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$t<w \land (t := t+1.$ **wait until** $w)$

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

$= \quad t+1 \le w \land$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $\;=\;$ $t := t \uparrow w$	busy-wait loop

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

proof

$t<w \land (t:= t+1.$ **wait until** $w)$

$=$ $t<w \land (t:= t+1.\ t' = t\uparrow w \ \land\ ...\ )$

$=$ $t+1 \le w \land$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$t<w \land (t := t+1.$ **wait until** $w)$

$= \quad t<w \land (t' = (t+1) \uparrow w \;\land\; \ldots)$

$= \quad t+1 \le w \land$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$$t<w \wedge (t:= t+1.\ \ \textbf{wait until } w)$$

$$= \quad t<w \wedge (t:= t+1.\ \ t:= t \uparrow w)$$

$$= \quad t+1 \leq w \wedge (t:= (t+1) \uparrow w)$$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$$t<w \wedge (t := t+1.\ \textbf{wait until}\ w)$$

$$= \quad t<w \wedge (t := t+1.\ t := t \uparrow w)$$

$$= \quad t+1 \le w \wedge (t := (t+1) \uparrow w)$$

$$= \quad t<w \wedge (t := w)$$

Time Dependence

$deadline := t + 5$	no problem
if $t < deadline$ **then** ... **else** ... **fi**	no problem
$t := 5$	problem: unimplementable
wait until w $=$ $t := t \uparrow w$	busy-wait loop

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

proof

$$t<w \wedge (t := t+1.\ \textbf{wait until}\ w)$$

$$= \quad t<w \wedge (t := t+1.\ t := t \uparrow w)$$

$$= \quad t+1 \le w \wedge (t := (t+1) \uparrow w)$$

$$= \quad t<w \wedge (t := w)$$

$$= \quad t<w \wedge (t := t \uparrow w)$$

Time Dependence

$deadline := t + 5$ — no problem

if $t < deadline$ **then** ... **else** ... **fi** — no problem

$t := 5$ — problem: unimplementable

wait until w $=$ $t := t \uparrow w$ — busy-wait loop

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

proof

$t<w \wedge (t := t+1.$ **wait until** $w)$

$= \quad t<w \wedge (t := t+1.\ t := t \uparrow w)$

$= \quad t+1 \leq w \wedge (t := (t+1) \uparrow w)$

$= \quad t<w \wedge (t := w)$

$= \quad t<w \wedge (t := t \uparrow w)$

$\Rightarrow$ **wait until** w

Space Dependence

Space Dependence

if $s < 1000000$ **then** ... **else** ... **fi** no problem

Space Dependence

if $s < 1000000$ **then** ... **else** ... **fi**	no problem
$s := 5$	problem

Space Dependence

if $s < 1000000$ **then** ... **else** ... **fi**	no problem
$s := 5$	problem

assignments to s must account for space

Space Dependence

if $s < 1000000$ **then** ... **else** ... **fi**	no problem
$s := 5$	problem

assignments to s must account for space

real space	implementation dependent

Assertions

Assertions

assert b

Assertions

assert b

$=$ "I believe b is true"

Assertions

	assert b
=	"I believe b is true"
=	**precondition** b

Assertions

assert b

= “I believe b is true”

= **precondition** b

= **postcondition** b

Assertions

	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** ∞ **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** ∞ **fi**

redundant, adds robustness

Assertions

assert b

= "I believe b is true"

= **precondition** b

= **postcondition** b

= **invariant** b

= **if** b **then** *ok* **else** *print* "error: ... ". **wait until** ∞ **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** ∞ **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: ... ". **wait until** ∞ **fi**

redundant, adds robustness, costs execution time

ensure b

= "make b be true without doing anything"

Assertions

assert b

= “I believe b is true”

= **precondition** b

= **postcondition** b

= **invariant** b

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

redundant, adds robustness, costs execution time

ensure b

= “make b be true without doing anything”

= **if** b **then** ok **else** $b' \wedge ok$ **fi**

= $b' \wedge ok$

Assertions

assert b

$=$ "I believe b is true"

$=$ **precondition** b

$=$ **postcondition** b

$=$ **invariant** b

$=$ **if** b **then** ok **else** $print$ "error: ... ". **wait until** ∞ **fi**

redundant, adds robustness, costs execution time

ensure b

$=$ "make b be true without doing anything"

$=$ **if** b **then** ok **else** $b' \wedge ok$ **fi**

$=$ $b' \wedge ok$

unimplementable

Assertions

assert b

= "I believe b is true"

= **precondition** b

= **postcondition** b

= **invariant** b

= **if** b **then** ok **else** *print* "error: ... ". **wait until** ∞ **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 by itself, but may be used in some contexts

nondeterministic choice

$$P \lor Q$$

nondeterministic choice (a programming notation):

$$P \vee Q$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \vee Q$$

nondeterministic choice (a programming notation):

P **or** Q $\quad = \quad$ $P \lor Q$

$x := 0$ **or** $x := 1$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1$$

$$= \qquad x'{=}0 \land y'{=}y \ \lor \ x'{=}1 \land y'{=}y$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ \ x'{=}1 \land x'{=}x \land y'{=}y$$

$$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ \ x'{=}1 \land x'{=}x \land y'{=}y$$

$$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \textbf{ensure } x{=}1$$

$$=\quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$$

$$=\quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}y) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$$

$$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

$$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$x{:=}\ 0$ **or** $x{:=}\ 1$. **ensure** $x{=}1$

$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$

$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$

$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$

$= \quad x'{=}1 \land y'{=}y$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\,0 \textbf{ or } x{:=}\,1.\ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$$

$$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

$$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$$

$$= \quad x'{=}1 \land y'{=}y$$

$$= \quad x{:=}\,1$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \ \textbf{ensure } x{=}1$$

$$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ \ x'{=}1 \land x'{=}x \land y'{=}y$$

$$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$$

$$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$$

$$= \quad x'{=}1 \land y'{=}y$$

$$= \quad x{:=}\ 1$$

implementation: **backtracking**

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \ \textbf{ensure}\ x{=}1$$

$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ \ x'{=}1 \land x'{=}x \land y'{=}y$

$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$

$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$

$= \quad x'{=}1 \land y'{=}y$

$= \quad x{:=}\ 1$

implementation: **backtracking**

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

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \vee Q$$

$$x{:=}\ 0 \textbf{ or } x{:=}\ 1.\ \textbf{ensure } x{=}1$$

$= \quad x'{=}0 \wedge y'{=}y \ \vee\ x'{=}1 \wedge y'{=}y.\ \ x'{=}1 \wedge x'{=}x \wedge y'{=}y$

$= \quad \exists x'', y''\cdot\ (x''{=}0 \wedge y''{=}y \ \vee\ x''{=}1 \wedge y''{=}\mathrm{y}) \wedge x'{=}1 \wedge x'{=}x'' \wedge y'{=}y''$

$= \quad (x'{=}0 \wedge y'{=}y \ \vee\ x'{=}1 \wedge y'{=}\mathrm{y}) \wedge x'{=}1$

$= \quad x'{=}1 \wedge y'{=}y$

$= \quad x{:=}\ 1$

implementation: **backtracking**

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

$$s{:=}\ 0,..n+1$$

nondeterministic choice (a programming notation):

$$P \textbf{ or } Q \quad = \quad P \lor Q$$

$x{:=}\ 0$ **or** $x{:=}\ 1.$ **ensure** $x{=}1$

$= \quad x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}y.\ x'{=}1 \land x'{=}x \land y'{=}y$

$= \quad \exists x'', y''\cdot\ (x''{=}0 \land y''{=}y \ \lor\ x''{=}1 \land y''{=}\mathrm{y}) \land x'{=}1 \land x'{=}x'' \land y'{=}y''$

$= \quad (x'{=}0 \land y'{=}y \ \lor\ x'{=}1 \land y'{=}\mathrm{y}) \land x'{=}1$

$= \quad x'{=}1 \land y'{=}y$

$= \quad x{:=}\ 1$

implementation: **backtracking**

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

$s{:=}\ 0,..n+1.$ **ensure** $s^2 \le n < (s+1)^2$