A Purely Logical Approach to Imperative Program Verification

Mădălina Erașcu

Research Institute for Symbolic Computation,
Johannes Kepler University, Linz, Austria
merascu@risc.uni-linz.ac.at

Joint work with Tudor Jebelean

November 19, 2009

Outline

Outline

Approach

Our approach:

Forward symbolic execution

Functional semantics

Contributions: purely logical approach

- Program syntax, semantics, partial correctness, termination - formalized in predicate logic
- Suitable for all imperative languages
- Verification conditions (including termination condition):
  - ensure existence and the uniqueness of the program function
  - are first order logic formulae
- No necessity for a meta-theory about the program execution, semantics, etc.

Outline

Main Ingredients

[Application specific] **Object theory:**

First order logic with equality

Expresses the properties of the objects manipulated by the program

[Universal] **Meta-theory:**

Meta-terms

- the program itself
- terms and formulae from the object theory

Imperative program statements

- assignments (including recursive call)
- conditionals
- `While` loops
- abrupt statements (`Return`)

Predicates and functions for reasoning about programs

- Syntax: Π
  - usage of initialized variables
  - every branch has a `Return` statement
- Semantics: $\Sigma \rightsquigarrow$ implicit definition of the program function
- Partial correctness: Γ
  - safety verification conditions
  - functional verification conditions
- Termination: $\Theta \rightsquigarrow$ termination conditions

Outline

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\ \texttt{Pre} \longrightarrow a > 0 \wedge b > 0, \\
\ \texttt{Post} \longrightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\ x = x - y; v = v + u, \\
\qquad\ y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \longrightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Syntax: $\surd$

Syntax

Definition

1. $\Pi[P] \Leftrightarrow \Pi[\{\bar{\alpha}\}, P]$
2. $\Pi[V, \langle \texttt{Return}[t] \rangle \frown P] \Leftrightarrow \mathit{Vars}[t] \subseteq V$
3. $\Pi[V, \langle \texttt{break} \rangle \frown P] \Leftrightarrow \mathit{True}$
4. $\Pi[V, \langle v := t \rangle \frown P] \Leftrightarrow \bigwedge \begin{cases} \mathit{Vars}[t] \subseteq V \\ \Pi[V \cup \{v\}, P] \end{cases}$
5. $\Pi[V, \langle \texttt{If}[\varphi, P_T, P_F] \rangle \frown P] \Leftrightarrow \bigwedge \begin{cases} \mathit{Vars}[\varphi] \subseteq V \\ \Pi[V, P_T \frown P] \\ \Pi[V, P_F \frown P] \end{cases}$
6. $\Pi\big[V, \langle \texttt{While}[\varphi, \iota, B] \rangle \frown P\big] \Leftrightarrow \bigwedge \begin{cases} \mathit{Vars}[\varphi] \subseteq V \\ \Pi\big[V, B \frown \langle \texttt{Return}[\mathit{True}] \rangle\big] \\ \Pi\big[V, P\big] \end{cases}$
7. $\Pi[V, P] = \mathbb{F}$

Outline

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Semantics:

$$a > 0 \wedge b > 0 \wedge a = b \Rightarrow (LG[a, b] = b)$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\quad\quad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\quad\quad \texttt{If}[x > y, \\
\quad\quad\quad x = x - y; v = v + u, \\
\quad\quad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Semantics:

$$a > 0 \wedge b > 0 \wedge a = b \Rightarrow (LG[a, b] = b)$$

$$a > 0 \wedge b > 0 \wedge x \neq y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b)$$
$$\wedge x > y \Rightarrow True$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD-LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD-LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\quad\quad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\quad\quad \texttt{If}[x > y, \\
\quad\quad\quad x = x - y; v = v + u, \\
\quad\quad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD-LCM"}]]
\end{array}
$$

Semantics:

$$a > 0 \wedge b > 0 \wedge a = b \Rightarrow (LG[a, b] = b)$$

$$a > 0 \wedge b > 0 \wedge x \neq y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x > y \Rightarrow \mathit{True}$$

$$a > 0 \wedge b > 0 \wedge x \neq y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \leq y \Rightarrow \mathit{True}$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b],$$
$$\texttt{Pre} \rightarrow a > 0 \wedge b > 0,$$
$$\texttt{Post} \rightarrow \beta = LCM[a, b]]$$
$$\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b],$$
$$x = a; y = b; u = b; v = a;$$
$$\texttt{While}[x \neq y,$$
$$(GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b),$$
$$\texttt{If}[x > y,$$
$$x = x - y; v = v + u,$$
$$y = y - x; u = u + v]];$$
$$\texttt{Return}[(u + v)/2]],$$
$$\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]$$

Semantics:

$$a > 0 \wedge b > 0 \wedge a = b \Rightarrow (LG[a, b] = b)$$

$$a > 0 \wedge b > 0 \wedge x \neq y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x > y \Rightarrow True$$

$$a > 0 \wedge b > 0 \wedge x \neq y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \leq y \Rightarrow True$$

$$a > 0 \wedge b > 0 \wedge (x' = y' \wedge (GCD[x', y'] = GCD[a, b]) \wedge x' > 0 \wedge y' > 0 \wedge (x' * u' + y' * v' = 2 * a * b) \Rightarrow (LG[a, b] = \frac{u' + v'}{2}))$$

Linear Search Algorithm, v.1

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow \beta = 0 \\
\texttt{Program}[\texttt{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n + 1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{Return}[i]]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[0]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"LinSearch"}]]
\end{array}
$$

Semantics:

$$n \geq 1 \wedge 1 > n \Rightarrow LS[A, n, e] = 0$$

$$n \geq 1 \wedge i \leq n \wedge (i \leq n + 1 \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e) \wedge e = A_{[i]} \Rightarrow (LS[A, n, e] = i)$$

$$n \geq 1 \wedge i \leq n \wedge (i \leq n + 1 \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e) \wedge e \neq A_{[i]} \Rightarrow \mathit{True}$$

$$n \geq 1 \wedge i' > n \wedge (i' \leq n + 1 \Rightarrow \underset{1 \leq k < i'}{\forall} A_{[k]} \neq e) \Rightarrow (LS[A, n, e] = 0)$$

Linear Search Algorithm, v. 2

$$
\begin{array}{l}
\texttt{Specification}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \\
\texttt{Program}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n + 1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{break}]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[i]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"LinSearch"}]]
\end{array}
$$

Semantics:

$n \geq 1 \wedge (1 > n) \Rightarrow (LS[A, n, e] = 1)$

True

True

$n \geq 1 \Rightarrow i' > n \wedge (i' \leq n + 1 \Rightarrow \underset{1 \leq k < i'}{\forall} A_{[k]} \neq e) \Rightarrow (LS[A, n, e] = i'))$

Semantics

Definition

1. $\Sigma[P] = \big(I_f[\bar{\alpha}] \Rightarrow \Sigma[\{\bar{\alpha} \to \bar{\alpha}_0\},\ P]\{\bar{\alpha}_0 \to \bar{\alpha}\}\big)$
2. $\Sigma[\sigma,\ \langle \texttt{Return}[t]\rangle \frown P] = (f[\bar{\alpha}_0] = t\sigma)$
3. $\Sigma[\sigma,\ \langle \texttt{break}\rangle \frown P] = \mathit{True}$
4. $\Sigma[\sigma,\ \langle v := t\rangle \frown P] = \Sigma[\sigma\{v \to t\sigma\},\ P]$
5. $\Sigma[\sigma,\ \langle \texttt{If}[\varphi,\ P_T,\ P_F]\rangle \frown P] = \bigwedge \begin{cases} \varphi\sigma \Rightarrow \Sigma[\sigma,\ P_T \frown P] \\ \neg\varphi\sigma \Rightarrow \Sigma[\sigma,\ P_F \frown P] \end{cases}$
6. $\Sigma\big[\sigma,\ \langle\rangle\big] = \mathit{True}$
7. $\Sigma\big[\sigma,\ \langle \texttt{While}[\varphi,\ \iota,\ B]\rangle \frown P\big] = \bigwedge \begin{cases} \neg\varphi\sigma \Rightarrow\ \Sigma[\sigma,\ P] \\ \big(\varphi\sigma_0 \wedge \iota\sigma_0 \Rightarrow \Sigma\big[\sigma_0,\ B\big]\big)\{\bar{\delta}_0 \to \bar{\delta}\} \\ \neg\varphi\sigma' \wedge \iota\sigma' \ \Rightarrow\ \Sigma\big[\sigma',\ P\big] \end{cases}$

Outline

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Partial Correctness:

$$a > 0 \wedge b > 0 \wedge (a = b) \Rightarrow (LCM(a, b) = \frac{b + a}{2})$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Partial Correctness:

$$a > 0 \wedge b > 0 \wedge (a = b) \Rightarrow (LCM(a, b) = \frac{b + a}{2})$$

$$a > 0 \wedge b > 0 \Rightarrow \big(GCD[a, b] = GCD[a, b] \wedge a > 0 \wedge b > 0 \wedge a * b + b * a = 2 * a * b\big)$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Partial Correctness:

$$a > 0 \wedge b > 0 \wedge (a = b) \Rightarrow (LCM(a, b) = \frac{b + a}{2})$$

$$a > 0 \wedge b > 0 \Rightarrow \big((GCD[a, b] = GCD[a, b]) \wedge a > 0 \wedge b > 0 \wedge (a * b + b * a = 2 * a * b)\big)$$

$$\big(a > 0 \wedge b > 0 \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \neq y \wedge x > y\big)$$
$$\Rightarrow \big((GCD[x - y, y] = GCD[a, b]) \wedge x - y > 0 \wedge y > 0 \wedge ((x - y) * u + y * (v + u) = 2 * a * b)\big)$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

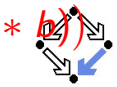
Partial Correctness:

$$a > 0 \wedge b > 0 \wedge (a = b) \Rightarrow (LCM(a, b) = \frac{b + a}{2})$$

$$a > 0 \wedge b > 0 \Rightarrow \big((GCD[a, b] = GCD[a, b]) \wedge a > 0 \wedge b > 0 \wedge (a * b + b * a = 2 * a * b)\big)$$

$$\begin{array}{l}
\big(a > 0 \wedge b > 0 \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \neq y \wedge x > y\big) \\
\qquad \Rightarrow \big((GCD[x - y, y] = GCD[a, b]) \wedge x - y > 0 \wedge y > 0 \wedge ((x - y) * u + y * (v + u) = 2 * a * b)\big)
\end{array}$$

$$\begin{array}{l}
\big(a > 0 \wedge b > 0 \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \neq y \wedge x \leq y\big) \\
\qquad \Rightarrow \big((GCD[x, y - x] = GCD[a, b]) \wedge x > 0 \wedge y - x > 0 \wedge (x * (u + v) + (y - x) * v = 2 * a * b)\big)
\end{array}$$



Example: Program computing simultaneously the GCD and the LCM of two positive integers

$\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b],$
$\texttt{Pre} \rightarrow a > 0 \wedge b > 0,$
$\texttt{Post} \rightarrow \beta = LCM[a, b]]$
$\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b],$
$x = a; y = b; u = b; v = a;$
$\texttt{While}[x \neq y,$
$(GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b),$
$\texttt{If}[x > y,$
$x = x - y; v = v + u,$
$y = y - x; u = u + v]];$
$\texttt{Return}[(u + v)/2]],$
$\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]$

Partial Correctness:

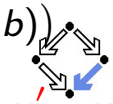
$$a > 0 \wedge b > 0 \wedge (a = b) \Rightarrow (LCM(a, b) = \frac{b + a}{2})$$

$$a > 0 \wedge b > 0 \Rightarrow \big((GCD[a, b] = GCD[a, b]) \wedge a > 0 \wedge b > 0 \wedge (a * b + b * a = 2 * a * b)\big)$$

$$\big(a > 0 \wedge b > 0 \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \neq y \wedge x > y\big)$$
$$\Rightarrow \big((GCD[x - y, y] = GCD[a, b]) \wedge x - y > 0 \wedge y > 0 \wedge ((x - y) * u + y * (v + u) = 2 * a * b)\big)$$

$$\big(a > 0 \wedge b > 0 \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge x \neq y \wedge x \leq y\big)$$
$$\Rightarrow \big((GCD[x, y - x] = GCD[a, b]) \wedge x > 0 \wedge y - x > 0 \wedge (x * (u + v) + (y - x) * v = 2 * a * b)\big)$$

$$\big(a > 0 \wedge b > 0 \wedge (GCD[x', y'] = GCD[a, b]) \wedge x' > 0 \wedge y' > 0 \wedge (x' * u' + y' * v' = 2 * a * b) \wedge x' = y'$$



Linear Search Algorithm, v.1

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow \beta = 0 \\
\texttt{Program}[\texttt{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{Return}[i]]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[0]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"LinSearch"}]]
\end{array}
$$

Partial Correctness:

$$n \geq 1 \wedge 1 > n \Rightarrow \left(\underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[0]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow 0 = 0 \right)$$

$$n \geq 1 \wedge i \leq n \wedge (i \leq n+1 \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e) \wedge e = A_{[i]} \Rightarrow \left(\underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[i]} = e \right.$$
$$\left. \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow i = 0 \right)$$

$$n \geq 1 \wedge i \leq n \wedge (i \leq n+1 \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e) \wedge e \neq A_{[i]} \Rightarrow \left(i \leq n \Rightarrow \underset{1 \leq k \leq i}{\forall} A_{[k]} \neq e \right)$$

$$n \geq 1 \wedge i' > n \wedge (i' \leq n+1 \Rightarrow \underset{1 \leq k < i'}{\forall} A_{[k]} \neq e) \Rightarrow \left(\underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[i']} = e \wedge \right.$$
$$\left. \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow i = 0 \right)$$

Linear Search Algorithm, v. 2

$$
\begin{array}{l}
\texttt{Specification}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow \beta = 0 \\
\texttt{Program}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{break}]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[i]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"LinSearch"}]]
\end{array}
$$

Partial Correctness:

$$n \geq 1 \wedge 1 > n \Rightarrow \left(\underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[1]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow 1 = 0 \right)$$

True

$$n \geq 1 \wedge i \leq n \wedge (i \leq n+1 \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e) \wedge e \neq A_{[i]} \Rightarrow \left(i \leq n \Rightarrow \underset{1 \leq k \leq i}{\forall} A_{[k]} \neq e \right)$$

$$
\begin{array}{l}
n \geq 1 \wedge i' > n \wedge (i' \leq n+1 \Rightarrow \underset{1 \leq k < i'}{\forall} A_{[k]} \neq e) \Rightarrow \left(\underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[i']} = e \right. \\
\qquad \left. \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow i' = 0 \right)
\end{array}
$$

Partial Correctness

Definition

1. $\Gamma[P] = \Gamma[\{\bar{\alpha} \to \bar{\alpha}_0\},\ I_f[\bar{\alpha}_0],\ P]\{\bar{\alpha}_0 \to \bar{\alpha}\}$
2. $\Gamma[\sigma,\ \Phi,\ \langle \texttt{Return}[\gamma]\rangle \smile P] = \langle \Phi \Rightarrow O_f[\bar{\alpha}_0, \gamma\sigma]\rangle$
3. $\Gamma[\sigma,\ \Phi,\ \langle v := \gamma\rangle \smile P] = \Gamma[\sigma\{v \to \gamma\sigma\},\ \Phi,\ P]$
4. $\Gamma[\sigma,\ \Phi,\ \langle v := h[\overline{\gamma}]\rangle \smile P] = \langle \Phi \Rightarrow I_h[\overline{\gamma}\sigma]\rangle \smile \Gamma[\sigma\{v \to h[\overline{\gamma}\sigma]\},\ \Phi \wedge I_h[\overline{\gamma}\sigma],\ P]$
5. $\Gamma[\sigma,\ \Phi,\ \langle v := g[\overline{\gamma}]\rangle \smile P] =$
   $\langle \Phi \Rightarrow I_g[\overline{\gamma}\sigma]\rangle \smile \Gamma[\sigma\{v \to c\}, \Phi \wedge I_g[\overline{\gamma}\sigma] \wedge O_g[\overline{\gamma}\sigma, c], P]$
6. $\Gamma[\sigma, \Phi, \langle \texttt{If}[\varphi, P_T, P_F]\rangle \smile P] = \Gamma[\sigma, \Phi \wedge \varphi\sigma, P_T \smile P] \smile \Gamma[\sigma, \Phi \wedge \neg\varphi\sigma, P_F \smile P]$
7. $\Gamma\big[\sigma,\ \Phi,\ \langle\rangle\big] = \langle \Phi \Rightarrow \iota\sigma\rangle$
8. $\Gamma\big[\sigma,\ \Phi,\ \langle \texttt{While}[\varphi,\ \iota,\ B]\rangle \smile P\big] = \smile \begin{cases} \Gamma\big[\sigma, \Phi \wedge \neg\varphi\sigma, P\big] \\ \langle \Phi \Rightarrow \iota\sigma\rangle \\ \Gamma\big[\sigma_0, \Phi \wedge \iota\sigma_0 \wedge \varphi\sigma_0, B\big]\{\bar{\delta}_0 \to \bar{\delta}\} \\ \Gamma\big[\sigma', \Phi \wedge \iota\sigma' \wedge \neg\varphi\sigma', P\big] \end{cases}$

Outline

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$\texttt{Specification}[\texttt{"GCD - LCM"}, LG[\downarrow a, \downarrow b],$$
$$\texttt{Pre} \longrightarrow a > 0 \wedge b > 0,$$
$$\texttt{Post} \longrightarrow \beta = LCM[a, b]]$$
$$\texttt{Program}[\texttt{"GCD - LCM"}, LG[\downarrow a, \downarrow b],$$
$$x = a; y = b; u = b; v = a;$$
$$\texttt{While}[x \neq y,$$
$$(GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b),$$
$$\texttt{If}[x > y,$$
$$x = x - y; v = v + u,$$
$$y = y - x; u = u + v]];$$
$$\texttt{Return}[(u + v)/2]],$$
$$\texttt{Specification} \longrightarrow \texttt{Specification}[\texttt{"GCD - LCM"}]]$$

Termination:

–

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\text{"GCD} - \texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \longrightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \longrightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\text{"GCD} - \texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \longrightarrow \texttt{Specification}[\text{"GCD} - \texttt{LCM"}]]
\end{array}
$$

Termination:

–

$(a > 0 \wedge b > 0 \wedge x = y) \Rightarrow \pi[x, y, u, v]$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\quad\quad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\quad\quad \texttt{If}[x > y, \\
\quad\quad\quad x = x - y; v = v + u, \\
\quad\quad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Termination:

–

$$(a > 0 \wedge b > 0 \wedge x = y) \Rightarrow \pi[x, y, u, v]$$

$$(a > 0 \wedge b > 0 \wedge x > y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b)$$
$$\wedge \pi[x - y, y, u, v + u]) \Rightarrow \pi[x, y, u, v]$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\texttt{"GCD}-\texttt{LCM"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\quad\quad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\quad\quad \texttt{If}[x > y, \\
\quad\quad\quad x = x - y; v = v + u, \\
\quad\quad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\texttt{"GCD}-\texttt{LCM"}]]
\end{array}
$$

Termination:

–

$$(a > 0 \wedge b > 0 \wedge x = y) \Rightarrow \pi[x, y, u, v]$$

$$(a > 0 \wedge b > 0 \wedge x > y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge \pi[x - y, y, u, v + u]) \Rightarrow \pi[x, y, u, v]$$

$$(a > 0 \wedge b > 0 \wedge x < y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge \pi[x, y - x, u + v, v]) \Rightarrow \pi[x, y, u, v]$$

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM[a, b]] \\
\texttt{Program}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\quad\quad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\quad\quad \texttt{If}[x > y, \\
\quad\quad\quad x = x - y; v = v + u, \\
\quad\quad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}]]
\end{array}
$$

Termination:

–

$$(a > 0 \wedge b > 0 \wedge x = y) \Rightarrow \pi[x, y, u, v]$$

$$(a > 0 \wedge b > 0 \wedge x > y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge \pi[x - y, y, u, v + u]) \Rightarrow \pi[x, y, u, v]$$

$$(a > 0 \wedge b > 0 \wedge x < y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \wedge \pi[x, y - x, u + v, v]) \Rightarrow \pi[x, y, u, v]$$

–

Example: Program computing simultaneously the GCD and the LCM of two positive integers

$$
\begin{array}{l}
\texttt{Specification}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}, LG[\downarrow a, \downarrow b], \\
\quad \texttt{Pre} \rightarrow a > 0 \wedge b > 0, \\
\quad \texttt{Post} \rightarrow \beta = LCM(a, b)] \\
\texttt{Program}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}, LG[\downarrow a, \downarrow b], \\
\quad x = a; y = b; u = b; v = a; \\
\quad \texttt{While}[x \neq y, \\
\qquad (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b), \\
\qquad \texttt{If}[x > y, \\
\qquad\quad x = x - y; v = v + u, \\
\qquad\quad y = y - x; u = u + v]]; \\
\quad \texttt{Return}[(u + v)/2]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"}\texttt{GCD}-\texttt{LCM}\text{"}]]
\end{array}
$$

Overall termination condition:

$$
\begin{array}{l}
\wedge \begin{cases}
(a > 0 \wedge b > 0 \wedge x = y) \Rightarrow \pi[x, y, u, v] \\
(a > 0 \wedge b > 0 \wedge x > y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \\
\qquad \wedge \pi[x - y, y, u, v + u]) \Rightarrow \pi[x, y, u, v] \\
(a > 0 \wedge b > 0 \wedge x < y \wedge (GCD[x, y] = GCD[a, b]) \wedge x > 0 \wedge y > 0 \wedge (x * u + y * v = 2 * a * b) \\
\qquad \wedge \pi[x, y - x, u + v, v]) \Rightarrow \pi[x, y, u, v]
\end{cases} \\
\quad \Rightarrow \pi[a, b, b, a]
\end{array}
$$

Linear Search Algorithm, v.1

$$
\begin{array}{l}
\texttt{Specification}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow \beta = 0 \\
\texttt{Program}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{Return}[i]]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[0]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"LinSearch"}]]
\end{array}
$$

Overall termination condition:

$$
\smile \left\{ \begin{array}{l} - \\ \left\langle \wedge \left\{ \begin{array}{l} n \geq 1 \wedge i > n \Rightarrow \pi[i] \\ n \geq 1 \wedge i \leq n \wedge (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e \wedge (e = A_{[i]}) \Rightarrow \pi[i] \\ n \geq 1 \wedge i \leq n \wedge (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e \wedge (e \neq A_{[i]}) \wedge \pi[i+1] \Rightarrow \pi[i] \end{array} \right\} \Rightarrow \pi[1] \right\rangle \\ - \\ - \end{array} \right.
$$

Linear Search Algorithm, v. 2

$$
\begin{array}{l}
\texttt{Specification}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Pre} \rightarrow n \geq 1, \\
\quad \texttt{Post} \rightarrow \underset{1 \leq k \leq n}{\exists} A_{[k]} = e \Rightarrow A_{[\beta]} = e \wedge \underset{1 \leq k \leq n}{\forall} A_{[k]} \neq e \Rightarrow \beta = 0 \\
\texttt{Program}[\text{"LinSearch"}, LS[\downarrow A, \downarrow n, \downarrow e], \\
\quad \texttt{Module}[\{i, k\}, \\
\quad i = 1; \\
\quad \texttt{While}[i \leq n, \\
\qquad (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e, \\
\qquad \texttt{If}[e = A_{[i]}, \\
\qquad\quad \texttt{break}]; \\
\qquad i = i + 1]; \\
\quad \texttt{Return}[i]], \\
\texttt{Specification} \rightarrow \texttt{Specification}[\text{"LinSearch"}]]
\end{array}
$$

Overall termination condition:

$$
\smile \left\{ \begin{array}{l} - \\ \left\langle \wedge \left\{ \begin{array}{l} n \geq 1 \wedge i > n \Rightarrow \pi[i] \\ n \geq 1 \wedge i \leq n \wedge (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e \wedge (e = A_{[i]}) \Rightarrow \pi[i] \\ n \geq 1 \wedge i \leq n \wedge (i \leq n+1) \Rightarrow \underset{1 \leq k < i}{\forall} A_{[k]} \neq e \wedge (e \neq A_{[i]}) \wedge \pi[i+1] \Rightarrow \pi[i] \end{array} \right\} \Rightarrow \pi[1] \right\rangle \\ - \\ - \end{array} \right.
$$

Termination

Definition

1. $\Theta[P] = \Theta[\{\bar{x} \to \bar{x_0}\},\ I_f[\bar{x}],\ P]\{\bar{x}_0 \to \bar{x}\}$
2. $\Theta[\sigma,\ \Phi,\ \langle \texttt{Return}[\bar{\delta}]\rangle \frown P] = \langle\rangle$
3. $\Theta[\sigma,\ \Phi,\ \langle \texttt{break}\rangle \frown P] = \langle\rangle$
4. $\Theta[\sigma,\ \Phi,\ \langle v := \gamma\rangle \frown P] = \Theta[\sigma\{v \to \gamma\sigma\},\ \Phi,\ P]$
5. $\Theta[\sigma,\ \Phi,\ \langle \texttt{If}[\varphi,\ P_T,\ P_F]\rangle \frown P] = \Theta[\sigma,\ \Phi \wedge \varphi\sigma,\ P_T \frown P] \wedge \Theta[\sigma,\ \Phi \wedge \neg\varphi\sigma,\ P_F \frown P]$
6. $\Theta\big[\sigma,\ \Phi,\ \langle\rangle\big] = \langle\rangle$
7. $\Theta[\sigma,\ \Phi,\ \langle \texttt{While}[\varphi,\ \iota,\ B] \frown P] =$
$$\frown \begin{cases} \Theta\big[\sigma, \Phi \wedge \neg\varphi\sigma,\ P\big] \\ \left\langle \wedge \left\{ \begin{array}{l} (\Phi \wedge \neg\varphi\sigma_0) \Rightarrow \pi[\bar{\delta}\sigma_0]\{\bar{\delta}_0 \to \bar{\delta}\} \\ \Theta'\big[\sigma_0, \Phi \wedge \varphi\sigma_0 \wedge \iota\sigma_0, B, \pi\big] \Rightarrow \pi[\bar{\delta}\sigma_0]\{\bar{\delta}_0 \to \bar{\delta}\} \end{array} \right\} \Rightarrow \pi[\bar{\delta}\sigma_I] \right\rangle \\ \Theta\big[\sigma_0, \Phi \wedge \varphi\sigma_0 \wedge \iota\sigma_0,\ B\big]\{\bar{\delta}_0 \to \bar{\delta}\} \\ \Theta\big[\sigma', \Phi \wedge \iota\sigma' \wedge \neg\varphi\sigma',\ P\big] \end{cases}$$

Contd

Definition

1. $\Theta'[\sigma,\ \Phi,\ \langle\rangle, \pi] = (\Phi \wedge \pi[\bar{\delta}\sigma])$
2. $\Theta'[\sigma,\ \Phi,\ \langle \texttt{Return}[\bar{\delta}]\rangle \frown P, \pi] = \Phi$
3. $\Theta'[\sigma,\ \Phi,\ \langle \texttt{break}\rangle \frown P, \pi] = \Phi$
4. $\Theta'[\sigma,\ \Phi,\ \langle \texttt{If}[\varphi,\ P_T,\ P_F, \pi]\rangle \frown P] = \vee \begin{cases} \Theta'[\sigma,\ \Phi,\ P_T \frown P, \pi] \\ \Theta'[\sigma,\ \Phi,\ P_F \frown P, \pi] \end{cases}$
5. $\Theta'[\sigma,\ \Phi,\ \langle \texttt{While}[\varphi,\ \iota,\ B, \pi]\rangle \frown P] = \Theta''[\sigma,\ \Phi,\ \langle \texttt{While}[\varphi,\ \iota,\ B]\rangle \frown P, \pi]$

Definition

1. $\Theta''[\sigma,\ \Phi,\ \langle\rangle, \pi] = (\Phi \wedge \pi[\sigma\delta, \pi])$
2. $\Theta''[\sigma,\ \Phi,\ \langle \texttt{break}\rangle \frown P, \pi] = \mathbb{F}$
3. $\Theta''[\sigma,\ \Phi,\ \langle \texttt{Return}[\bar{\delta}]\rangle \frown P, \pi] = \Phi$
4. $\Theta''[\sigma,\ \Phi,\ \langle \texttt{If}[\varphi,\ P_T,\ P_F]\rangle \frown P, \pi] = \vee \begin{cases} \Theta''[\sigma,\ \Phi,\ P_T \frown P, \pi] \\ \Theta''[\sigma,\ \Phi,\ P_F \frown P, \pi] \end{cases}$
5. $\Theta''[\sigma,\ \Phi,\ \langle \texttt{While}[\varphi,\ \iota,\ B]\rangle \frown P, \pi] = \vee \begin{cases} \Theta''[\sigma, \Phi \wedge \neg\varphi\sigma, P, \pi] \\ \Theta''[\sigma, \Phi \wedge \varphi\sigma \wedge \iota\sigma, B, \pi] \\ \Theta''[\sigma, \Phi \wedge \neg\varphi\sigma' \wedge \iota\sigma', P, \pi] \end{cases}$

Outline

Termination of Non-Recursive Single While Loops Programs; 2 non-nested loops

Binary Division Algorithm, by Kaldewaij

Specification["BinaryDivision", $BD[\downarrow A, \downarrow B]$,
 Pre $\longrightarrow A \geq 0 \wedge B > 0$,
 Post $\longrightarrow \beta = A\%B$]
Program["BinaryDivision", $BD[\downarrow a, \downarrow b]$,
  $q = 0; r = A; b = B;$
  While$[r \geq b,$

$$\left(\underset{k\geq 0}{\exists} b = 2^k * B\right) \wedge q = 0 \wedge r = A \wedge A \geq 0 \wedge B > 0, \quad \text{(*invariant*)}$$

  $b = 2 * b]$; (*endwhile*)
  While$[b \neq B,$
  $A = q * b + r; \wedge r \geq 0 \wedge \left(\underset{k\geq 0}{\exists} b = 2^k * B\right) \wedge B > 0 \wedge b > r, \quad \text{(*invariant*)}$
  $q = 2 * q; b = b/2;$
  If$[r \geq b,$
  $q = q + 1; r = r - b]]$; (*then,endwhile*)
  Return$[r]]$
Specification $\longrightarrow$ Specification["BinaryDivision"]]

Overall termination conditions:

$$\wedge \left\{ \begin{array}{l} (A \geq 0 \wedge B > 0 \wedge r < b) \Rightarrow \pi_1[b] \\ (A \geq 0 \wedge B > 0 \wedge r \geq b \wedge \left(\underset{k\geq 0}{\exists} b = 2^k * B\right) \wedge q = 0 \wedge r = A \wedge A \geq 0 \wedge B > 0 \wedge \pi_1[2 * b]) \Rightarrow \pi_1[b] \end{array} \right\} \Rightarrow \pi_1[B] \quad \textit{1st} \text{ While}$$

Termination of Non-Recursive Single While Loops Programs; 2 non-nested loops

Binary Division Algorithm, by Kaldewaij

```
Specification["BinaryDivision", BD[↓ A, ↓ B],
 Pre ⟶ A ≥ 0 ∧ B > 0,
 Post ⟶ β = A%B]
Program["BinaryDivision", BD[↓ a, ↓ b],
  q = 0; r = A; b = B;
  While[r ≥ b,
    (∃_{k≥0} b = 2^k * B) ∧ q = 0 ∧ r = A ∧ A ≥ 0 ∧ B > 0,  (*invariant*)
    b = 2 * b];  (*endwhile*)
  While[b ≠ B,
    A = q * b + r; ∧r ≥ 0 ∧ (∃_{k≥0} b = 2^k * B) ∧ B > 0 ∧ b > r,  (*invariant*)
    q = 2 * q; b = b/2;
    If[r ≥ b,
      q = q + 1; r = r − b]];  (*then,endwhile*)
  Return[r]]
Specification ⟶ Specification["BinaryDivision"]]
```

Overall termination conditions:

$$
\wedge \left\{ \begin{array}{l}
(A \geq 0 \wedge B > 0 \wedge A < B \wedge b = B \Rightarrow \pi_2[q, b, r] \\
(A \geq 0 \wedge B > 0 \wedge A < B \wedge b \neq B \wedge A = q * b + r \wedge r \geq 0 \wedge (\underset{k\geq 0}{\exists} b = 2^k * B) \wedge B > 0 \wedge b > r \wedge r \geq b \\
\quad \wedge \pi_2[2 * q + 1, b/2, r - b]) \Rightarrow \pi_2[q, b, r] \\
(b \neq B \wedge A = q * b + r \wedge r \geq 0 \wedge (\underset{k\geq 0}{\exists} b = 2^k * B) \wedge B > 0 \wedge b > r \wedge r < b \wedge \pi_2[2 * q, b/2, r]) \Rightarrow \pi_2[q, b, r]
\end{array} \right\} \Rightarrow \pi_2[0, B, A]
$$

$$
\wedge \left\{ \begin{array}{l}
A \geq 0 \wedge B > 0 \wedge \underset{k\geq 0}{\exists} b' = 2^k * B \wedge q = 0 \wedge r = A \wedge A \geq 0 \wedge B > 0 \wedge r < b' \wedge b = B \Rightarrow \pi_2[q, b, r] \\
(A \geq 0 \wedge B > 0 \wedge \underset{k\geq 0}{\exists} b' = 2^k * B \wedge q = 0 \wedge r = A \wedge A \geq 0 \wedge B > 0 \wedge r < b' \wedge b \neq B \wedge A = q * b + r \wedge r \geq 0 \\
\quad \wedge (\underset{k\geq 0}{\exists} b = 2^k * B) \wedge B > 0 \wedge b > r \wedge r \geq b \wedge \pi_2[2 * q + 1, b/2, r - b]) \Rightarrow \pi_2[q, b, r] \\
(A \geq 0 \wedge B > 0 \wedge \underset{k\geq 0}{\exists} b' = 2^k * Bq = 0 \wedge r = A \wedge A \geq 0 \wedge B > 0 \wedge r < b' \wedge b \neq B \wedge A = q * b + r \wedge r \geq 0 \\
\quad \wedge (\underset{k\geq 0}{\exists} b = 2^k * B) \wedge B > 0 \wedge b > r \wedge r < b \wedge \pi_2[2 * q, b/2, r]) \Rightarrow \pi_2[q, b, r]
\end{array} \right\} \Rightarrow \pi_2[0, b', A]
$$

2nd While

Termination of Non-Recursive Nested `While` Loops Programs

Program searching for an element in a matrix

```
Specification["Search", Search[↓ A, ↓ m, ↓ n, ↓ e],
 Pre ⟶ m ≥ 1 ∧ n ≥ 1,
 Post ⟶ ∃_{1≤k≤m} ∃_{1≤l≤n} A[k][l] = e ⇒ A[β1][β2] = e ∧ ∀_{1≤k≤m} ∀_{1≤l≤n} A[k][l] ≠ e ⇒ y = 0
Program["Search", Search[↓ A, ↓ m, ↓ n, ↓ e],
  i = 1; j = 1;
  While[i ≤ m,
   i ≤ m + 1 ⇒ ∀_{1≤k<i} ∀_{1≤j≤n} A[k][j] ≠ e,  (*invariant*)
   While[j ≤ n,
    j ≤ n + 1 ⇒ ∀_{1≤i≤m} ∀_{1≤l<j} A[i][l] ≠ e,  (*invariant*)
    If[e = A[i][j],
     Return[⟨i, j⟩]];  (*then, endif*)
    j = j + 1];  (*endwhile*)
   i = i + 1];  (*endwhile*)
  Return[0]],
Specification ⟶ Specification["GCD − LCM"]]
```

Overall termination condition: the invariant is omitted for readability of the formulae.

$$\wedge \left\{ \begin{array}{l} m \geq 1 \wedge n \geq 1 \wedge i > m \Rightarrow \pi_1[i,j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j > n \wedge \pi_1[i+1,j] \Rightarrow \pi_1[i,j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j \leq n \wedge A_{[i][j]} = e \Rightarrow \pi_1[i,j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j \leq n \wedge A_{[i][j]} \neq e \wedge \pi_1[i,j+1] \Rightarrow \pi_1[i,j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j' > n \wedge \pi_1[i+1,j'] \Rightarrow \pi_1[i,j] \end{array} \right\} \Rightarrow \pi_1[1,1] \quad (\textit{outer } \texttt{While})$$

$$\wedge \left\{ \begin{array}{l} m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j > n \Rightarrow \pi_2[j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j \leq n \wedge A_{[i][j]} = e \Rightarrow \pi_2[j] \\ m \geq 1 \wedge n \geq 1 \wedge i \leq m \wedge j \leq n \wedge A_{[i][j]} \neq e \wedge \pi_2[j+1] \Rightarrow \pi_2[j] \end{array} \right\} \Rightarrow \pi_2[1] \quad (\textit{inner } \texttt{While})$$

Thank you!

Questions?