Wstęp do Programowania potok funkcyjny

Marcin Kubica

2016/2017

Outline

Pojęcie specyfikacji

Definition

Specyfikacja opis celu,
może być w postaci warunku początkowego i końcowego.

Implementacja realizacja w języku programowania.

Weryfikacja dowód, że implementacja spełnia specyfikację.

Warunek początkowy dozwolone wartości argumentów.

Warunek końcowy jak wynik zależy od argumentów.
(Dla argumentów spełniających warunek wstępny muszą istnieć wyniki spełniające warunek końcowy.)

Pojęcie specyfikacji

Definition

Specyfikacja	opis celu, może być w postaci warunku początkowego i końcowego.
Implementacja	realizacja w języku programowania.
Weryfikacja	dowód, że implementacja spełnia specyfikację.
Warunek początkowy	dozwolone wartości argumentów.
Warunek końcowy	jak wynik zależy od argumentów. (Dla argumentów spełniających warunek wstępny muszą istnieć wyniki spełniające warunek końcowy.)

Pojęcie specyfikacji

Definition

Specyfikacja	opis celu, może być w postaci warunku początkowego i końcowego.
Implementacja	realizacja w języku programowania.
Weryfikacja	dowód, że implementacja spełnia specyfikację.
Warunek początkowy	dozwolone wartości argumentów.
Warunek końcowy	jak wynik zależy od argumentów. (Dla argumentów spełniających warunek wstępny muszą istnieć wyniki spełniające warunek końcowy.)

Pojęcie specyfikacji

Definition

Specyfikacja	opis celu, może być w postaci warunku początkowego i końcowego.
Implementacja	realizacja w języku programowania.
Weryfikacja	dowód, że implementacja spełnia specyfikację.
Warunek początkowy	dozwolone wartości argumentów.
Warunek końcowy	jak wynik zależy od argumentów. (Dla argumentów spełniających warunek wstępny muszą istnieć wyniki spełniające warunek końcowy.)

Pojęcie specyfikacji

Definition

Specyfikacja	opis celu, może być w postaci warunku początkowego i końcowego.
Implementacja	realizacja w języku programowania.
Weryfikacja	dowód, że implementacja spełnia specyfikację.
Warunek początkowy	dozwolone wartości argumentów.
Warunek końcowy	jak wynik zależy od argumentów. (Dla argumentów spełniających warunek wstępny muszą istnieć wyniki spełniające warunek końcowy.)

Pojęcie specyfikacji, c.d.

- Specyfikacja może dopuszczać wiele implementacji
  n.p. $\forall x : |f\ x| = |x|$ nie określa znaku $f\ x$.
- Implementacja może dawać określone wyniki, nawet jeżeli dane nie spełniają warunku początkowego.
- Specyfikacje podajemy dla wszystkich tworzonych procedur, w tym pomocniczych.
  Weryfikacja procedur pomocniczych przypomina dowody lematów.
- To, że program spełnia specyfikację można wyrazić formułą:

$$\text{warunek początkowy}(\text{dane}) \implies \text{warunek końcowy}(\text{dane}, \text{wynik})$$

Pomija ona kwestie związane z zapętleniem lub błędami w obliczeniach.

Pojęcie specyfikacji, c.d.

- Specyfikacja może dopuszczać wiele implementacji
  n.p. $\forall x : |f\ x| = |x|$ nie określa znaku $f\ x$.
- Implementacja może dawać określone wyniki, nawet jeżeli dane nie spełniają warunku początkowego.
- Specyfikacje podajemy dla wszystkich tworzonych procedur, w tym pomocniczych.
  Weryfikacja procedur pomocniczych przypomina dowody lematów.
- To, że program spełnia specyfikację można wyrazić formułą:

$$\text{warunek początkowy}(\text{dane}) \implies \text{warunek końcowy}(\text{dane}, \text{wynik})$$

Pomija ona kwestie związane z zapętleniem lub błędami w obliczeniach.

Pojęcie specyfikacji, c.d.

- Specyfikacja może dopuszczać wiele implementacji
  n.p. $\forall x : |f\ x| = |x|$ nie określa znaku $f\ x$.
- Implementacja może dawać określone wyniki, nawet jeżeli dane nie spełniają warunku początkowego.
- Specyfikacje podajemy dla wszystkich tworzonych procedur, w tym pomocniczych.
  Weryfikacja procedur pomocniczych przypomina dowody lematów.
- To, że program spełnia specyfikację można wyrazić formułą:

$$\text{warunek początkowy(dane)} \implies \text{warunek końcowy(dane, wynik)}$$

  Pomija ona kwestie związane z zapętleniem lub błędami w obliczeniach.

Pojęcie specyfikacji, c.d.

- Specyfikacja może dopuszczać wiele implementacji
  n.p. $\forall x : |f\ x| = |x|$ nie określa znaku $f\ x$.
- Implementacja może dawać określone wyniki, nawet jeżeli dane nie spełniają warunku początkowego.
- Specyfikacje podajemy dla wszystkich tworzonych procedur, w tym pomocniczych.
  Weryfikacja procedur pomocniczych przypomina dowody lematów.
- To, że program spełnia specyfikację można wyrazić formułą:

$$\text{warunek początkowy(dane)} \implies \text{warunek końcowy(dane, wynik)}$$

Pomija ona kwestie związane z zapętleniem lub błędami w obliczeniach.

Pełna i częściowa poprawność

Definition

Własność stopu

$$\text{warunek początkowy(dane)} \implies \text{wynik jest określony}$$

Częściowa poprawność

$$\text{warunek początkowy(dane)} \wedge \text{wynik jest określony} \implies$$
$$\implies \text{warunek końcowy(dane, wynik)}$$

Pełna poprawność = częściowa poprawność + własność stopu.

Pełna i częściowa poprawność

Definition

Własność stopu

$$\text{warunek początkowy(dane)} \implies \text{wynik jest określony}$$

Częściowa poprawność

$$\text{warunek początkowy(dane)} \wedge \text{wynik jest określony} \implies$$

$$\implies \text{warunek końcowy(dane, wynik)}$$

Pełna poprawność = częściowa poprawność + własność stopu.

Pełna i częściowa poprawność

Definition

Własność stopu

$$\text{warunek początkowy(dane)} \implies \text{wynik jest określony}$$

Częściowa poprawność

$$\text{warunek początkowy(dane)} \wedge \text{wynik jest określony} \implies$$
$$\implies \text{warunek końcowy(dane, wynik)}$$

Pełna poprawność = częściowa poprawność + własność stopu.

Niezmienniki iteracji

Definition (Niezmiennik iteracji)

Niezmiennik iteracji, to warunek, który:

- musi być spełniony przed rozpoczęciem iteracji,
- każdy krok iteracji zachowuje niezmiennik,
- po zakończeniu iteracji jest nadal spełniony.

W przypadku iteracyjnych procedur rekurencyjnych niezmienniki zwykle pokrywają się z warunkami początkowymi.

Niezmienniki iteracji

Definition (Niezmiennik iteracji)

Niezmiennik iteracji, to warunek, który:

- musi być spełniony przed rozpoczęciem iteracji,
- każdy krok iteracji zachowuje niezmiennik,
- po zakończeniu iteracji jest nadal spełniony.

W przypadku iteracyjnych procedur rekurencyjnych niezmienniki zwykle pokrywają się z warunkami początkowymi.

Dowodzenie częściowej poprawności

Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wynik jest określony i zgodny z warunkiem końcowym.

Dowodzenie częściowej poprawności

Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wynik jest określony i zgodny z warunkiem końcowym.

Dowodzenie częściowej poprawności

Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wynik jest określony i zgodny z warunkiem końcowym.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Dowodzenie własności stopu

W przypadku procedur rekurencyjnych należy udowodnić, że ich wywołania (spełniające warunki początkowe) nie zapętlają się.
Schemat dowodu:

- Zakładamy, że argumenty wywołania spełniają warunek początkowy.
- Określamy funkcję miary φ : argumenty $\rightarrow \mathbb{N}$.
  Jest to górne ograniczenie na głębokość rekurencji.
- Zakładamy, że wszystkie wywołania procedur spełniają specyfikacje.
- Pokazujemy, że wywołania procedur spełniają odpowiednie warunki początkowe.
- Dla wszystkich wywołań rekurencyjnych pokazujemy, że funkcja miary maleje.

Algorytm Euklidesa (wersja z odejmowaniem)

Example

Warunek początkowy: $x, y > 0$
Warunek końcowy: $(\texttt{nwd}\ x\ y) = NWD(x, y)$

```
let rec nwd x y =
  if x = y then x else
    if x > y then nwd (x - y) y else nwd x (y - x);;
```

Dowód.

Funkcja miary $f = \max(x, y)$.
Jeśli $x = y$, to oczywiste. Załóżmy, że $x > y$.
Dla każdego d takiego, że $d|x$ i $d|y$ mamy $d|(x - y)$.
I odwrotnie, jeśli $d|(x - y)$ i $d|y$, to $d|(x - y + y) = x$.
Stąd $\text{nwd}(x, y) = \text{nwd}(x - y, y)$. Dla $x < y$ analogicznie. □

Dowód własności stopu okazał się łatwiejszy niż częściowej poprawności.

Algorytm Euklidesa (wersja z odejmowaniem)

Example

Warunek początkowy: $x, y > 0$
Warunek końcowy: $(\texttt{nwd}\ x\ y) = NWD(x, y)$

```
let rec nwd x y =
  if x = y then x else
    if x > y then nwd (x - y) y else nwd x (y - x);;
```

Dowód.

Funkcja miary $f = \max(x, y)$.
Jeśli $x = y$, to oczywiste. Załóżmy, że $x > y$.
Dla każdego d takiego, że $d|x$ i $d|y$ mamy $d|(x - y)$.
I odwrotnie, jeśli $d|(x - y)$ i $d|y$, to $d|(x - y + y) = x$.
Stąd $\text{nwd}(x, y) = \text{nwd}(x - y, y)$. Dla $x < y$ analogicznie. □

Dowód własności stopu okazał się łatwiejszy niż częściowej poprawności.

Algorytm Euklidesa (wersja z odejmowaniem)

Example

Warunek początkowy: $x, y > 0$
Warunek końcowy: $(\texttt{nwd}\ x\ y) = NWD(x, y)$

```
let rec nwd x y =
  if x = y then x else
    if x > y then nwd (x - y) y else nwd x (y - x);;
```

Dowód.

Funkcja miary $f = \max(x, y)$.
Jeśli $x = y$, to oczywiste. Załóżmy, że $x > y$.
Dla każdego d takiego, że $d|x$ i $d|y$ mamy $d|(x - y)$.
I odwrotnie, jeśli $d|(x - y)$ i $d|y$, to $d|(x - y + y) = x$.
Stąd $\text{nwd}(x, y) = \text{nwd}(x - y, y)$. Dla $x < y$ analogicznie. □

Dowód własności stopu okazał się łatwiejszy niż częściowej poprawności.

Silnia i przykład niezmiennika

Example

Warunek początkowy: $num > 0$
Warunek końcowy: `silnia` $num = num!$

```
let silnia num =
  let rec silnia_pom a k =
    if k = num then a
    else silnia_pom (a * (k+1)) (k+1)
  in silnia_pom 1 1;;
```

Warunek początkowy `silnia_pom` = niezmiennik iteracji:
$1 \leq k \leq num \ \wedge \ a = k!$
Trywialnie spełniony dla $a = k = 1$.
Dla $k = num$ otrzymujemy warunek końcowy:
`silnia_` $a\ k = num!$.

Bisekcja i przykład niezmiennika

Example

Warunek początkowy: $n \geq 0$.

Warunek końcowy: sqrt $n = \lfloor \sqrt{n} \rfloor$.

```
let sqrt num =
  let rec bisekcja lewy prawy =
    if lewy = prawy then lewy else
      let srodek = (lewy + prawy) / 2
      in
        if square (srodek + 1) > num then
          bisekcja lewy srodek
        else bisekcja (srodek + 1) prawy
  in bisekcja 0 num;;
```

Warunek początkowy = niezmiennik iteracji:

$lewy \leq \sqrt{num} < prawy + 1$.

Trywialnie spełniony dla $lewy = 0$ i $prawy = num$.

Dla $lewy = prawy$ otrzymujemy warunek końcowy:

bisekcja $lewy\ prawy = \sqrt{num}$.

Metoda Newtona liczenia pierwiastków

Example

- Specyfikacja:
  Warunek początkowy: $x \geq 0$
  Warunek końcowy: $\left|\texttt{sqrt}\ x - \sqrt{x}\right| \leq \varepsilon$
  Wynikiem procedury `sqrt` jest przybliżenie pierwiastka z dokładnością do ε.
- Implementacja:

```
let epsilon = ...;;
let sqrt x =
  let rec sqrt_iter g =
    if good g x then g
    else sqrt_iter (improve g x)
  in
    sqrt_iter (guess x);;
```

Metoda Newtona liczenia pierwiastków

Example

- Specyfikacja:
  Warunek początkowy: $x \geq 0$
  Warunek końcowy: $\left|\texttt{sqrt}\ x - \sqrt{x}\right| \leq \varepsilon$
  Wynikiem procedury `sqrt` jest przybliżenie pierwiastka z dokładnością do ε.
- Implementacja:

```
let epsilon = ...;;
let sqrt x =
  let rec sqrt_iter g =
    if good g x then g
    else sqrt_iter (improve g x)
  in
    sqrt_iter (guess x);;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Predykat good decyduje o tym, kiedy przybliżenie jest dobre.
- Specyfikacja good g x:
  warunek początkowy: $x \geq 0 \ \wedge \ g \geq 0$
  warunek końcowy: good g $x \implies \left|g - \sqrt{x}\right| \leq \varepsilon$
- Trywialnie zapewnia częściową poprawność algorytmu.
- Jak zaimplementować good nie znając dokładnej wartości $\sqrt{x}$?

```
let good g x =
  abs_float ((g *. g) -. x) <= epsilon *. g;;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Predykat good decyduje o tym, kiedy przybliżenie jest dobre.
- Specyfikacja good g x:
  warunek początkowy: $x \geq 0 \ \wedge \ g \geq 0$
  warunek końcowy: good g $x \implies \left|g - \sqrt{x}\right| \leq \varepsilon$
- Trywialnie zapewnia częściową poprawność algorytmu.
- Jak zaimplementować good nie znając dokładnej wartości $\sqrt{x}$?

```
let good g x =
  abs_float ((g *. g) -. x) <= epsilon *. g;;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Predykat good decyduje o tym, kiedy przybliżenie jest dobre.
- Specyfikacja good g x:
  warunek początkowy: $x \geq 0 \ \wedge \ g \geq 0$
  warunek końcowy: good g $x \implies \left|g - \sqrt{x}\right| \leq \varepsilon$
- Trywialnie zapewnia częściową poprawność algorytmu.
- Jak zaimplementować good nie znając dokładnej wartości $\sqrt{x}$?

```
let good g x =
  abs_float ((g *. g) -. x) <= epsilon *. g;;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Predykat good decyduje o tym, kiedy przybliżenie jest dobre.
- Specyfikacja good g x:
  warunek początkowy: $x \geq 0 \ \wedge \ g \geq 0$
  warunek końcowy: good g $x \implies \left|g - \sqrt{x}\right| \leq \varepsilon$
- Trywialnie zapewnia częściową poprawność algorytmu.
- Jak zaimplementować good nie znając dokładnej wartości $\sqrt{x}$?

```
let good g x =
  abs_float ((g *. g) -. x) <= epsilon *. g;;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

Oznaczmy $e = g - \sqrt{x}$.

$$
\begin{aligned}
\left|g^2 - x\right| &= \left|\left(g - \sqrt{x}\right)\left(g + \sqrt{x}\right)\right| = |e|\left(g + \sqrt{x}\right) \\
|e| &= \frac{\left|g^2 - x\right|}{g + \sqrt{x}} \\
\text{good } g\ x &\Leftrightarrow \left|g^2 - x\right| \leq \varepsilon \cdot g \implies \\
\left|g^2 - x\right| &\leq \varepsilon\left(g + \sqrt{x}\right) \Leftrightarrow \frac{\left|g^2 - x\right|}{g + \sqrt{x}} \leq \varepsilon \Leftrightarrow \\
|e| &\leq \varepsilon
\end{aligned}
$$

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Częściowa poprawność `sqrt` jest zapewniona, o ile tylko `guess` i `improve` zwracają liczby nieujemne.
- Zapewnienie własności stopu jest trudniejsze.
- Czego oczekujemy od `guess` i `improve`?
  Chcemy, aby nasza iteracja kiedyś się zakończyła:

$$\exists_{n \in \mathbb{N}}\ \mathtt{good}\,((\mathtt{fun}\ g \rightarrow \mathtt{improve}\ g\ x)^n\ g)\ x$$

- Jeżeli $g > 0$, to $\sqrt{x}$ leży gdzieś między g, a $\frac{x}{g}$.
  Dobrym kandydatem jest więc średnia arytmetyczna tych liczb.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Częściowa poprawność `sqrt` jest zapewniona, o ile tylko `guess` i `improve` zwracają liczby nieujemne.
- Zapewnienie własności stopu jest trudniejsze.
- Czego oczekujemy od `guess` i `improve`? Chcemy, aby nasza iteracja kiedyś się zakończyła:

$$\exists_{n \in \mathbb{N}}\ \texttt{good}\,((\texttt{fun}\ g \rightarrow \texttt{improve}\ g\ x)^n\ g)\ x$$

- Jeżeli $g > 0$, to $\sqrt{x}$ leży gdzieś między g, a $\frac{x}{g}$. Dobrym kandydatem jest więc średnia arytmetyczna tych liczb.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Częściowa poprawność `sqrt` jest zapewniona, o ile tylko `guess` i `improve` zwracają liczby nieujemne.
- Zapewnienie własności stopu jest trudniejsze.
- Czego oczekujemy od `guess` i `improve`? Chcemy, aby nasza iteracja kiedyś się zakończyła:

$$\exists_{n \in \mathbb{N}}\ \texttt{good}\,((\texttt{fun}\ g \to \texttt{improve}\ g\ x)^n\ g)\ x$$

- Jeżeli $g > 0$, to $\sqrt{x}$ leży gdzieś między g, a $\frac{x}{g}$. Dobrym kandydatem jest więc średnia arytmetyczna tych liczb.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Specyfikacja `improve`:
  Warunek początkowy: $g > 0 \ \wedge \ x \geq 0$
  Warunek końcowy:
  `improve` $g\ x > 0 \ \wedge \ \mu($`improve` $g\ x) < \mu(g)$,
  gdzie μ jest taką funkcją miary, że $\mu(g) = 0 \implies$ good $g\ x$.
- Implementacja `improve`:

```
let average x y = (x +. y) /. 2.0;;
let improve g x = average g (x /. g);;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Specyfikacja `improve`:
  Warunek początkowy: $g > 0 \ \wedge \ x \geq 0$
  Warunek końcowy:
  `improve` $g\ x > 0 \ \wedge \ \mu(\texttt{improve}\ g\ x) < \mu(g)$,
  gdzie μ jest taką funkcją miary, że $\mu(g) = 0 \implies \text{good}\ g\ x$.
- Implementacja `improve`:

```
let average x y = (x +. y) /. 2.0;;
let improve g x = average g (x /. g);;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Warunek początkowy `improve` pociąga `guess` $x > 0$.
- Specyfikacja `guess`:
  Warunek początkowy: $x \geq 0$
  Warunek końcowy: `guess` $x > 0$
- Implementacja `guess`:

```
let guess x = 1.0;;
```

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Jaka powinna być funkcja miary μ?
- Kolejne przybliżenia pierwiastka są dodatnie.
- Błąd kolejnego przybliżenia wynosi:

$$\begin{aligned}
\frac{g+\frac{x}{g}}{2}-\sqrt{x} &= \frac{\sqrt{x}+e+\frac{x}{\sqrt{x}+e}}{2}-\sqrt{x}=\frac{\sqrt{x}+e+\frac{x}{\sqrt{x}+e}-2\sqrt{x}}{2}= \\
&= \frac{e-\sqrt{x}+\frac{x}{\sqrt{x}+e}}{2}=\frac{(e-\sqrt{x})(e+\sqrt{x})+x}{2(\sqrt{x}+e)}= \\
&= \frac{e^2-x+x}{2(\sqrt{x}+e)}=\frac{e^2}{2(\sqrt{x}+e)}= \\
&= \frac{e}{2}\cdot\frac{1}{1+\frac{\sqrt{x}}{e}}
\end{aligned}$$

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Jeśli $e < 0$, to $-\sqrt{x} < e < 0$, oraz $1 + \frac{\sqrt{x}}{e} < 0$.
  W następnym kroku błąd jest dodatni.
- Jeśli $e > 0$, to $0 < \frac{1}{1+\frac{\sqrt{x}}{e}} < 1$.

  Błąd kolejnego przybliżenia jest dodatni i przynajmniej dwukrotnie mniejszy.
- Jeśli $e = 0$ to wychodzimy z pętli.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Jeśli $e < 0$, to $-\sqrt{x} < e < 0$, oraz $1 + \frac{\sqrt{x}}{e} < 0$.
  W następnym kroku błąd jest dodatni.
- Jeśli $e > 0$, to $0 < \frac{1}{1+\frac{\sqrt{x}}{e}} < 1$.
  Błąd kolejnego przybliżenia jest dodatni i przynajmniej dwukrotnie mniejszy.
- Jeśli $e = 0$ to wychodzimy z pętli.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Jeśli $e < 0$, to $-\sqrt{x} < e < 0$, oraz $1 + \frac{\sqrt{x}}{e} < 0$.
  W następnym kroku błąd jest dodatni.
- Jeśli $e > 0$, to $0 < \frac{1}{1+\frac{\sqrt{x}}{e}} < 1$.
  Błąd kolejnego przybliżenia jest dodatni i przynajmniej dwukrotnie mniejszy.
- Jeśli $e = 0$ to wychodzimy z pętli.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Funkcja miary:

$$\mu(g) = \begin{cases} \max\left(\left\lceil \log_2 \frac{2\cdot(g-\sqrt{x})}{\varepsilon} \right\rceil, 0\right) & \text{dla } g > \sqrt{x} \\ 1 + \mu\left(\frac{g+\frac{x}{g}}{2}\right) & \text{dla } g < \sqrt{x} \\ 0 & \text{dla } g = \sqrt{x} \end{cases}$$

- Jeżeli $\mu(g) = 0$, to $g \geq \sqrt{x}$ oraz $g - \sqrt{x} \leq \frac{\varepsilon}{2}$.

$$|g^2 - x| = (g - \sqrt{x})(g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot (g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot 2g = \varepsilon \cdot g$$

$$|g^2 - x| \leq \varepsilon \cdot g \implies \text{good } g\ x$$

- W wywołaniu rekurencyjnym wartość μ jest mniejsza.

Dowód własności stopu okazał się trudniejszy niż częściowej poprawności.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Funkcja miary:

$$\mu(g) = \begin{cases} \max\left(\left\lceil \log_2 \frac{2\cdot(g-\sqrt{x})}{\varepsilon} \right\rceil, 0\right) & \text{dla } g > \sqrt{x} \\ 1 + \mu\left(\frac{g+\frac{x}{g}}{2}\right) & \text{dla } g < \sqrt{x} \\ 0 & \text{dla } g = \sqrt{x} \end{cases}$$

- Jeżeli $\mu(g) = 0$, to $g \geq \sqrt{x}$ oraz $g - \sqrt{x} \leq \frac{\varepsilon}{2}$.

$$|g^2 - x| = (g - \sqrt{x})(g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot (g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot 2g = \varepsilon \cdot g$$

$$|g^2 - x| \leq \varepsilon \cdot g \implies \text{good } g\ x$$

- W wywołaniu rekurencyjnym wartość μ jest mniejsza.

Dowód własności stopu okazał się trudniejszy niż częściowej poprawności.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Funkcja miary:

$$\mu(g) = \begin{cases} \max\left(\left\lceil \log_2 \frac{2\cdot(g-\sqrt{x})}{\varepsilon} \right\rceil, 0\right) & \text{dla } g > \sqrt{x} \\ 1 + \mu\left(\frac{g+\frac{x}{g}}{2}\right) & \text{dla } g < \sqrt{x} \\ 0 & \text{dla } g = \sqrt{x} \end{cases}$$

- Jeżeli $\mu(g) = 0$, to $g \geq \sqrt{x}$ oraz $g - \sqrt{x} \leq \frac{\varepsilon}{2}$.

$$|g^2 - x| = (g - \sqrt{x})(g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot (g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot 2g = \varepsilon \cdot g$$

$$|g^2 - x| \leq \varepsilon \cdot g \implies \text{good } g\ x$$

- W wywołaniu rekurencyjnym wartość μ jest mniejsza.

Dowód własności stopu okazał się trudniejszy niż częściowej poprawności.

Metoda Newtona liczenia pierwiastków, c.d.

Example

- Funkcja miary:

$$\mu(g) = \begin{cases} \max\left(\left\lceil \log_2 \frac{2\cdot(g-\sqrt{x})}{\varepsilon} \right\rceil, 0\right) & \text{dla } g > \sqrt{x} \\ 1 + \mu\left(\frac{g+\frac{x}{g}}{2}\right) & \text{dla } g < \sqrt{x} \\ 0 & \text{dla } g = \sqrt{x} \end{cases}$$

- Jeżeli $\mu(g) = 0$, to $g \geq \sqrt{x}$ oraz $g - \sqrt{x} \leq \frac{\varepsilon}{2}$.

$$\begin{aligned} |g^2 - x| &= (g - \sqrt{x})(g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot (g + \sqrt{x}) \leq \frac{\varepsilon}{2} \cdot 2g = \varepsilon \cdot g \\ |g^2 - x| &\leq \varepsilon \cdot g \implies \text{good } g\ x \end{aligned}$$

- W wywołaniu rekurencyjnym wartość μ jest mniejsza.

Dowód własności stopu okazał się trudniejszy niż częściowej poprawności.

Podział problemu na podproblemy

Jak rozbiliśmy problem na podproblemy:

- iterowanie przybliżeń, aż są dobre: `sqrt`, `sqrt_iter`,
- pierwsze przybliżenie: `guess`,
- kiedy przybliżenie jest dobre: `good`,
- poprawienie przybliżenia: `improve`,
- średnia: `average`,
- wartość bezwzględna: `float_abs`.

Podział problemu na podproblemy

- Jak dzielić problem na podproblemy?
  Trudna i ważna umiejętność.
- *Zasada dekompozycji*:
  Podzielić problem trudniejszy na kilka prostszych „zgodnie z jego strukturą".
  Tak, aby dało się prosto wyrazić rozwiązanie całego problemu za pomocą rozwiązań podproblemów.
  Tak powstające definicje są proste i łatwe do ogarnięcia.
- *Zasada abstrakcji proceduralnej*:
  Tak formułować podproblemy, aby były jak najbardziej ogólne.
  Możliwość wykorzystania jednego rozwiązania w kilku miejscach.

Podział problemu na podproblemy

- Jak dzielić problem na podproblemy?
  Trudna i ważna umiejętność.
- *Zasada dekompozycji*:
  Podzielić problem trudniejszy na kilka prostszych „zgodnie z jego strukturą".
  Tak, aby dało się prosto wyrazić rozwiązanie całego problemu za pomocą rozwiązań podproblemów.
  Tak powstające definicje są proste i łatwe do ogarnięcia.
- *Zasada abstrakcji proceduralnej*:
  Tak formułować podproblemy, aby były jak najbardziej ogólne.
  Możliwość wykorzystania jednego rozwiązania w kilku miejscach.

Podział problemu na podproblemy

- Jak dzielić problem na podproblemy?
  Trudna i ważna umiejętność.
- *Zasada dekompozycji*:
  Podzielić problem trudniejszy na kilka prostszych „zgodnie z jego strukturą".
  Tak, aby dało się prosto wyrazić rozwiązanie całego problemu za pomocą rozwiązań podproblemów.
  Tak powstające definicje są proste i łatwe do ogarnięcia.
- *Zasada abstrakcji proceduralnej*:
  Tak formułować podproblemy, aby były jak najbardziej ogólne.
  Możliwość wykorzystania jednego rozwiązania w kilku miejscach.

Podział problemu na podproblemy

- *Zasada pobożnych życzeń*:
  Tworząc rozwiązanie problemu zakładamy na chwilę, że podproblemy są już rozwiązane.
  Dopiero potem przystępujemy do rozwiązywania podproblemów.
- *Spychologia stosowana* (na ćwiczeniach):
  Rozwiązanie zadania dzielimy między kilka osób.
  Można wyłonić podproblemy i wyspecyfikować je, pozostawiając rozwiązanie komu innemu.
- *Kierunek top-down* — od ogółu do szczegółu:
  Wyłaniamy problemy do rozwiązania dzieląc większe problemy na prostsze, a nie odwrotnie.

Podział problemu na podproblemy

- *Zasada pobożnych życzeń*:
  Tworząc rozwiązanie problemu zakładamy na chwilę, że podproblemy są już rozwiązane.
  Dopiero potem przystępujemy do rozwiązywania podproblemów.
- *Spychologia stosowana* (na ćwiczeniach):
  Rozwiązanie zadania dzielimy między kilka osób.
  Można wyłonić podproblemy i wyspecyfikować je, pozostawiając rozwiązanie komu innemu.
- *Kierunek top-down* — od ogółu do szczegółu:
  Wyłaniamy problemy do rozwiązania dzieląc większe problemy na prostsze, a nie odwrotnie.

Podział problemu na podproblemy

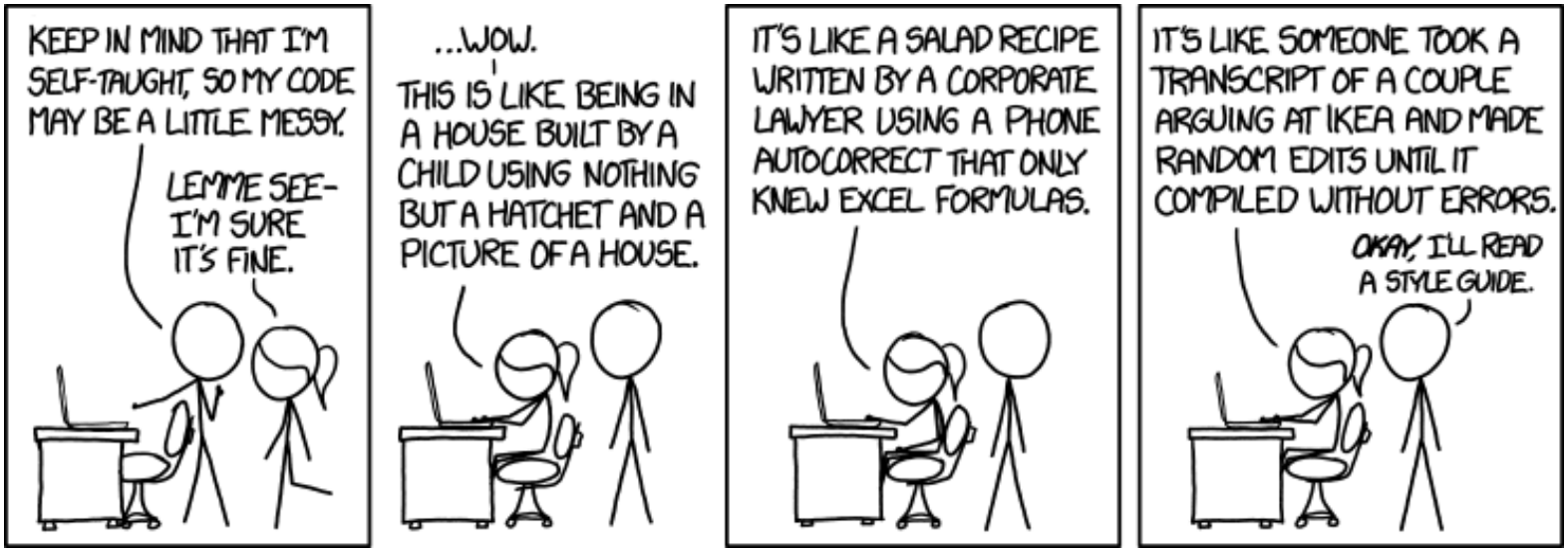
- *Zasada pobożnych życzeń*:
  Tworząc rozwiązanie problemu zakładamy na chwilę, że podproblemy są już rozwiązane.
  Dopiero potem przystępujemy do rozwiązywania podproblemów.
- *Spychologia stosowana* (na ćwiczeniach):
  Rozwiązanie zadania dzielimy między kilka osób.
  Można wyłonić podproblemy i wyspecyfikować je, pozostawiając rozwiązanie komu innemu.
- *Kierunek top-down* — od ogółu do szczegółu:
  Wyłaniamy problemy do rozwiązania dzieląc większe problemy na prostsze, a nie odwrotnie.

Kilka innych dobrych zasad

- *Black-box approach*:
  Polegamy tylko na specyfikacji, a nie na sposobie implementacji.
- *Information hiding*:
  Ukrywamy przed użytkownikiem elementy implementacji, które nie powinny go interesować.
  Definicje lokalne, moduły (w przyszłości).
- *Separation of concerns*:
  Na raz zajmujemy się tylko jednym problemem.
  Sposoby rozwiązania różnych problemów są niezależne.
  Nie chcemy, nie możemy i nie zajmujemy się nimi równocześnie.

Te zasady to trzy różne strony tej samej monety. :−)

Deser



http://xkcd.com/1513/