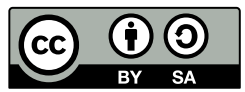
Programų teisingumo įrodymai

Saulius Gražulis

2011 ruduo

Vilnius University, Faculty of Mathematic and Informatics
Institute of Informatics



Programų teisingumą tegalime įrodyti, kaip matematikos teoremą

... correctness proofs for programs can be — and should be! —just as beautiful, fascinating and convincing as any other piece of mathematics.

Edsger W.Dijkstra
http://www.cs.utexas.edu/users/EWD/transcriptions/EWD05xx/EWD566.html
2009.11.09

Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.

C. A. R. Hoare
An axiomatic basis for computer programming.
Communications of the ACM, 12(10):576-585, October 1969.

Hoaro (Hoare) logika

Formali sistema programų tesingumui įrodyti

- Hoaro “trejetukas” (Hoare triple):

$$\{P\}\ S\ \{Q\}$$

- Formalios sistemos aksioma:

$$\frac{}{\{P\}\ S\ \{Q\}}$$

- Formalios sistemos išvedimo taisyklė:

$$\frac{\{P\}\ S\ \{Q\}}{\{P'\}\ S'\ \{Q'\}}$$

Hoaro (Hoare) logika

- Aksioma:

$$\frac{}{\{Q[E/id]\}\ id := E\ \{Q\}}$$

  - Trejetuko pavyzdys: $\{y + 7 > 10\}\ x := y + 7\ \{x > 10\}$

- Išvedimo taisyklė – nuoseklaus įvykdymo taisyklė:

$$\frac{\{P\}\ S1\ \{R\};\ \{R\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$

- Pradinės sąlygos sugriežtinimas, galutinės sąlygos susilpninimas:

$$\frac{P \Rightarrow P';\ \{P'\}\ S\ \{Q'\};\ Q' \Rightarrow Q}{\{P\}\ S\ \{Q\}}$$

Sąlyginiai operatoriai ir ciklai

- Išvedimo taisyklė – sąlyginis operatorius:

$$\frac{\{E\&P\}\ S\ \{Q\};\ \{!E\&P\}\ R\ \{Q\}}{\{P\}\ \textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ R\ \textbf{fi}\ \{Q\}}$$

- Išvedimo taisyklė – `while` ciklas:

$$\frac{\{E\&P\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ E\ \textbf{do}\ S\ \textbf{done}\ \{!E\&P\}}$$

  - P yra **ciklo invariantas**

Deikstros (Dijkstra) **silpniausia pradinė sąlyga**
(weakest precondition)

- $\{wp(S, Q)\}\ S\ \{Q\}$

  $wp(S, Q)$ yra predikatas, kuris teisingas didžiausiai programos būsenų aibei, garantuojančiai Q po S įvykdymo. $wp(\cdot, \cdot)$ yra predikatų funkcija (predicate transformer).
- Deikstros $wp(\cdot, \cdot)$ predikatų algebra:
  - $wp(x := E, Q) = Q[E/x]$
  - $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
  - $wp(\textbf{if}\ E\ \textbf{then}\ S\ \textbf{else}\ R, Q) = E \Rightarrow wp(S, Q)\ \&\ !E \Rightarrow wp(R, Q)$
  - $wp(\textbf{while}\ E\ \textbf{do}\ S, Q) = \exists k \geq 0 : H_k, \text{kur}\ \ H_0 = !E\ \&\ Q,\ H_{k+1} = H_0\ \|\ wp(S, H_k))$
- $wp(\cdot, \cdot)$ panaudojimas įrodymuose:
  $\{P\}\ S\ \{Q\}$ iff $P \Rightarrow wp(S, Q)$ iff $sp(P, S) \Rightarrow Q$

Euklido algoritmas DBD rasti

GCD – Greatest Common Divisor

- Duoti du natūriniai skaičiai. Raskime didžiausią sveiką (natūrinį) skaičių, abu juos dalinantį be liekanos (DBD, didžiausią bendrą daliklį):
- Algoritmą paminėjo Euklidas prieš ≈ 2300 metų
- Šiais laikais mes jį galime užrašyti taip:

`tests/2009.11.09/gcd.perl`:

```
sub gcd($$)
{
    my ($x, $y) = @_;

    while( $x != $y ) {
        if( $x > $y ) {
            $x -= $y;
        } else {
            $y -= $x;
        }
    }
    return $x;
}
```

Euklido algoritmas pasibaigia:

tests/2009.11.09/gcd-finitness-proof.perl:

```
sub gcd($$)
{
    my ($x, $y) = @_;
    # inv: $x is int && $y is int
    # pre: $x > 0 && $y > 0
    while( $x != $y ) {
        # pre: $x > 0 && $y > 0 && $x != $y
        if( $x > $y ) {
            # pre: $x > $y
            $x -= $y;
            # post: $x > 0
        } else {
            # pre: $x < $y ( <== $x != $y && !$x > $y )
            $y -= $x;
            # post: $y > 0
        }
        # inv: $x > 0 && $y > 0 => max( $x, $y ) > 0
        # post: max( new $x, new $y ) < max( old $x, old $y )
    }
    # post: $x == $y && $x > 0 && $y > 0
    return $x;
}
```

Euklido algoritmas tikrai duoda DBD:

tests/2009.11.09/gcd-correctness-proof.perl:

```
sub gcd($$)
{
    my ($x, $y) = @_;
    # assume: $X0 == initial $x, $Y0 == initial $y
    # pre: GCD( $x, $y ) == GCD( $X0, $Y0 )
    while( $x != $y ) {
        if( $x > $y ) {
            # pre:  GCD( $x, $y ) == GCD( $X0, $Y0 );
            $x -= $y;
            # post: GCD( $x, $y ) == GCD( $X0, $Y0 );
        } else {
            # pre:  GCD( $x, $y ) == GCD( $X0, $Y0 );
            $y -= $x;
            # post: GCD( $x, $y ) == GCD( $X0, $Y0 );
        }
        # post: GCD( new $x, new $y ) == GCD( old $x, old $y )
        # inv: GCD( $x, $y ) == GCD( $X0, $Y0 )
    }
    # post: $x == $y
    # post: $x == GCD($x,$x) == GCD($x,$y) == GCD($X0,$Y0)
    return $x;
}
```

Pradinių sąlygų užtikrinimas

tests/2009.11.09/gcd-with-asserts.perl:

```
sub gcd($$)
{
    my ($x, $y) = @_;

    assert( $x > 0 );
    assert( $y > 0 );

    while( $x != $y ) {
        if( $x > $y ) {
            $x -= $y;
        } else {
            $y -= $x;
        }
    }
    return $x;
}
```

Įdedame paprogramę į modulį

tests/2009.11.09/GCD.pm:

```
#-----------------------------------------------------------------
#$Author: saulius $
#$Date: 2009-11-09 16:07:25 +0200 (Mon, 09 Nov 2009) $
#$Revision: 573 $
#$Id: GCD.pm 573 2009-11-09 14:07:25Z saulius $
#-----------------------------------------------------------------
#*
#  Euclid's GCD (Greatest Common Divisor) algorithm.
#**
package GCD;
use strict;
use warnings;
use Carp::Assert;

sub gcd($$)
{
    my ($x, $y) = @_;
    assert( $x > 0 );
    assert( $y > 0 );
    while( $x != $y ) {
        if( $x > $y ) {
            $x -= $y;
        } else {
            $y -= $x;
        }
    }
    return $x;
}
1;
```

Pagrindinė programa

tests/2009.11.09/euclid-gcd.pl:

```
#! /bin/sh
#!perl -w # -*- Perl -*-
eval 'exec perl5 -x $0 ${1+"$@"}'
    if 0;
#-------------------------------------------------------------------------
#$Author: saulius $
#$Date: 2009-11-09 15:23:40 +0200 (Mon, 09 Nov 2009) $
#$Revision: 571 $
#$Id: euclid-gcd.pl 571 2009-11-09 13:23:40Z saulius $
#-------------------------------------------------------------------------
#*
#  Find GCD (Greatest Common Divider) using Euclid's algorithm.
#**

use strict;
use GCD;

while(<>) {
    my ($x,$y) = split;
    print GCD::gcd( $x, $y ), "\n";
}
```

Bėda – kur klaida galėjo pasislėpti?

tests/2009.11.09/GCDbad.pm:

```
#-----------------------------------------------------------------
#$Author: saulius $
#$Date: 2009-11-09 16:07:25 +0200 (Mon, 09 Nov 2009) $
#$Revision: 573 $
#$Id: GCDbad.pm 573 2009-11-09 14:07:25Z saulius $
#-----------------------------------------------------------------
#*
#  Euclid's GCD (Greatest Common Divisor) algorithm (hopelessly
#  broken!).
#**
package GCD;
use strict;
use warnings;
use Carp::Assert;

sub gcd($$)
{
    my ($x, $y) = @_;
    assert( $x > 0 );
    assert( $y > 0 );
    while( $x != $y ) {
        if( $x > $y ) {
            $y -= $x;
        } else {
            $x -= $y;
        }
    }
    return $x;
}
1;
```

Ar klaida algoritme, ar įrodyme?

`tests/2009.11.09/gcd-bad.perl`:

```
sub gcd($$)
{
    my ($x, $y) = @_;
    # assume: $X0 == initial $x
    # assume: $Y0 == initial $y
    while( $x != $y ) {
        if( $x > $y ) {
            # GCD( $x, $y ) == GCD( $X0, $Y0 );
            $y -= $x;
            # GCD( $x, $y ) == GCD( $X0, $Y0 );
        } else {
            # GCD( $x, $y ) == GCD( $X0, $Y0 );
            $x -= $y;
            # GCD( $x, $y ) == GCD( $X0, $Y0 );
        }
        # post: GCD( new $x, new $y ) == GCD( old $x, old $y )
        #                             == GCD( $X0, $Y0 )
        # inv: GCD( $x, $y ) == GCD( $X0, $Y0 )
    }
    # post: $x == GCD($x,$x) == GCD($x,$y) == GCD($X0,$Y0)
    return $x;
}
```