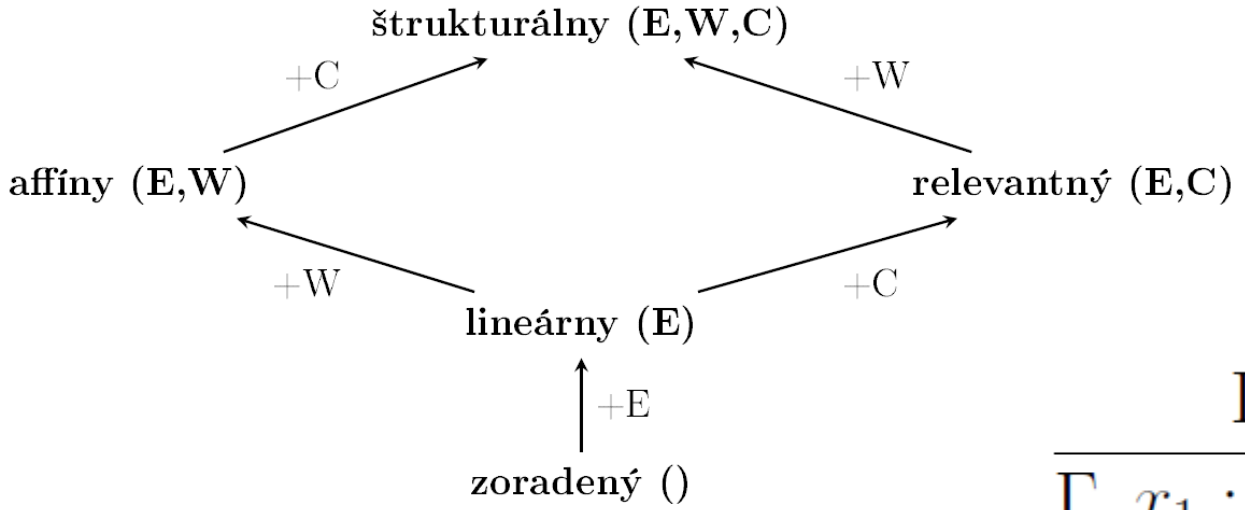
Subštrukturálne typové systémy v praxi

BC. PATRIK GRMAN

ŠKOLITEĽ: RNDR. RICHARD OSTERTÁG, PHD.

On the usefulness of linear types for correct nonce use enforcement during compile time

- kontrola korektného používania v čase kompilácie
- Oslabenie (W), Výmena (E), Kontrakcia (C)
- popísané v jazyku Rust



$$\frac{\Gamma \vdash e : \tau}{\Gamma, x : \tau_x \vdash e : \tau}$$

$$\frac{\Gamma_1, x : \tau_x, y : \tau_y, \Gamma_2 \vdash e : \tau}{\Gamma_1, y : \tau_y, x : \tau_x, \Gamma_2 \vdash e : \tau}$$

$$\frac{\Gamma, x_2 : \tau_x, x_3 : \tau_x \vdash e : \tau}{\Gamma, x_1 : \tau_x \vdash [x_2 \mapsto x_1, x_3 \mapsto x_1]e : \tau}$$

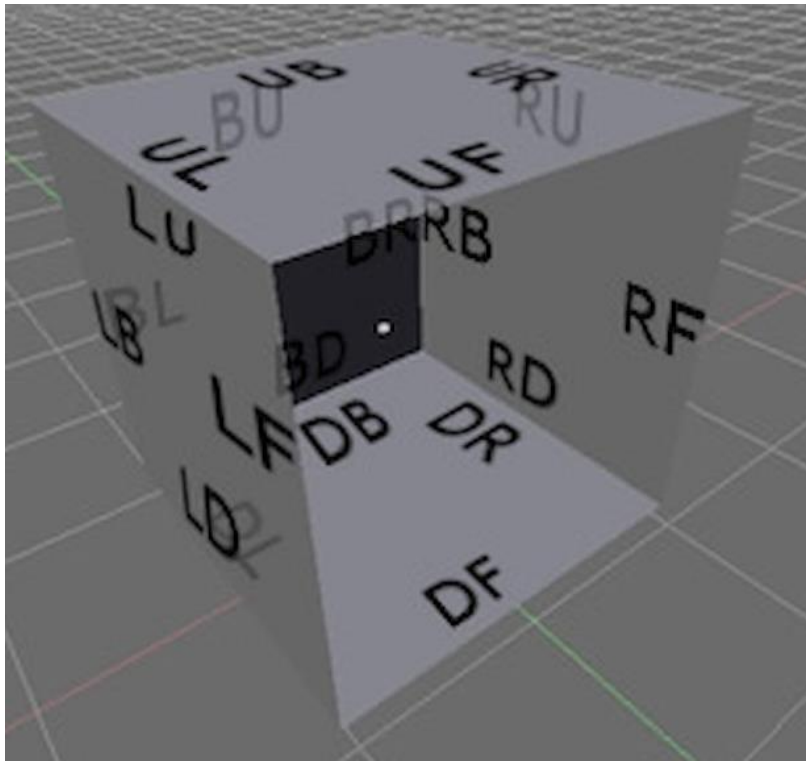
Príležitostné slovo

- požiadavka jednorazového použitia
- typická forma je reťazec alebo číslo fixnej veľkosti
- zvyčajne časová pečiatka alebo dostatočný počet náhodných bitov
- rôzne ďalšie požiadavky podľa použitia
  - nepredikovateľnosť, pseudonáhodnosť
  - ochrana proti replay útokom; deduplikácia správ
  - inicializačné vektory šifier
  - súkromný kľúč Lamportovej jednorazovej schémy
  - proof-of-work systémy

Ďalšie príklady použitia subštrukt.

- Practical Linearity in a higher-order polymorphic language — POPL 2018
  - popisujú správu systémových zdrojov
  - otvorené súbory – určite uzatvoriť a už nepoužiť.
  - dynamicky alokovaná pamäť – určite uvoľniť a už nemeniť
  - základná idea je obmedzenie použitia „handle“ na spravovaný objekt
- vzťah s kvantovým počítaním
  - veta o zákaze klonovania je neplatnosť pravidla o kontrakcii
  - veta o nemožnosti zmazania všeobecného stavu je neplatnosť pravidla o oslabení
  - je to teda lineárny typový systém (ak hardvér umožňuje výmenu)

- Making non-manifold models unrepresentable — Samuel Gélineau 2015
  - nápad nie je úplne formálne prezentovaný
  - automatická kontrola korektnosti 3D objektov
    - chýbajúce steny
  - založené na kontrole počtu použití hrany
    - práve 2, pre 2 steny, ktoré spája
  - autor priznáva, že neodhalí všetky typy problémov



Naše príbuzné pravidlá

- pre potenciálnu simuláciu systémov
- oslabenie, kontrakcia

$$\frac{\Gamma \vdash e : \tau}{\Gamma, x : \tau_x \vdash dump(x); e : \tau}$$

$$\frac{\Gamma, x_2 : \tau_x, x_3 : \tau_x \vdash e : \tau}{\Gamma, x_1 : \tau_x \vdash (x_2, x_3) := duplicate(x_1); e : \tau}$$

Správanie substrukturálnych systémov

- zoradené
  - zásobník s premennými, použitie odstráni a odobratie použije
- lineárne (výmena)
  - ľubovoľné prelínanie použitia, všeobecne každá premenná práve raz
- afínne (výmena, oslabenie)
  - podobné, ale nenúti použiť, čo môže byť postačujúce
  - pri práci so zdrojom nevyžaduje explicitné ukončenie, ale bráni použitiu po ukončení
- relevantné (výmena, kontrakcia)
  - pravidlo umožňuje 2 použitia, priamy dôsledok je ľubovoľne veľa opakovaných použití

Lineárne typy v Rust-e

- koncept vlastníctva, práve jedna premenná vlastní objekt/hodnotu
- umožňuje obmedziť používanie objektu tým, že preberie a nevráti vlastníctvo

```
mod nonce {
    // A public struct with a private random value of type u128
    pub struct Nonce {
        val: u128,
    }

    impl Nonce {
        // A public constructor method
        pub fn new() -> Nonce {
            use rand::prelude::*;
            Nonce { val: random() }
        }

        // A public getter method
        pub fn get(&self) -> u128 {
            self.val
        }
    }
}
```

```rust
fn need_new_random_u128_every_time(nonce: nonce::Nonce) {
    let _tmp = nonce.get();
    println!("Nonce param value: {}", nonce.get());
    println!("Nonce param value: {}", *nonce);
}
```

```rust
fn main() {
    // Structs with private fields can be created only using public constructors
    let mut nonce = nonce::Nonce::new();
    need_new_random_u128_every_time(nonce);

    nonce = nonce::Nonce::new();
    need_new_random_u128_every_time(nonce);

    need_new_random_u128_every_time(nonce::Nonce::new());
}
```

- priamy dôsledok konceptu vlastníctva je neplatnosť pravidla o kontrakcii
  - ak použitie premenných odovzdáva vlastníctvo, potrebuje 2 s rovnakou hodnotou
- pravidlo oslabenia platí, pokus o zakázanie nepoužitých premenných nestačí
  - nekontroluje všetky vetvy, nezískame teda vynútenie použitia
- podobne pravidlo výmeny platí
- systém je teda afínny

Lineárne typy v Haskell-i

- definuje špeciálny typ: lineárna funkcia
- funkcia je lineárna: ak výsledok sa skonzumuje práve 1 krát, tak argument sa skonzumuje práve 1 krát
- lineárne:
  - fun x = x
  - fun (x, y) = (y, x)
  - fun x = linearna (dalsia_linearna x)
- nelineárne:
  - fun x = 0
  - fun x = (x, x)
  - fun x = nelinearna (linearna x)

Practical Linearity in a higher-order polymorphic language — POPL 2018

- v čase, keď lineárne typy ešte neboli v hlavnej verzii GHC
- spomína dôvod ich zavedenia, hlavné myšlienky implementácie
- uvádza príklad rozhrania mutovateľných polí pomocou lineárnych typov
- špeciálny typ „Unrestricted“ umožňuje obídenie lineárnej kontroly

- operácie pre vytvorenie, zápis, čítanie a konverzia na imutabilné + iterácia
- demonštračný príklad: zoznam dvojíc (index, hodnota) na pole s tými hodnotami na príslušných indexoch (predpokladáme valídne)

```
newMArray     :: Int -> ST s (MArray s a)
read          :: MArray s a -> Int -> ST s a
write         :: MArray s a -> (Int, a) -> ST s ()
unsafeFreeze  :: MArray s a -> ST s (Array a)
forM          :: (Monad m) => [a] -> (a -> m ()) -> m ()
runST         :: (forall a. ST s a) -> a

array :: Int -> [(Int, a)] -> Array a
array size pairs = runST $ do
    ma <- newMArray size
    forM pairs (write ma)
    return $ unsafeFreeze ma
```

```
newMArray :: Int -> (MArray a %1-> Ur b) %1-> b
read      :: MArray a %1-> Int -> (MArray a, Ur a)
write     :: MArray a %1-> (Int, a) -> MArray a
freeze    :: MArray a %1 -> Ur (Array a)
foldl     :: (a %1-> b %1-> a) -> a %1-> [b] %1-> a

array :: Int -> [(Int, a)] -> Array a
array size pairs = newMArray size $ \ma ->
    freeze (foldl write ma pairs)
```

System.IO.Resource.Linear

- RIO – „Resource I/O“ monáda, rieši správu otvorených súborov

```
linearWriteToFile :: IO ()
linearWriteToFile = Linear.run $ Control.do
    handle1 <- Linear.openFile "/home/user/test.txt" Linear.
        WriteMode
    handle2 <- Linear.hPutStrLn handle1 (Text.pack "hello␣
        there")
    () <- Linear.hClose handle2
    Control.return (Ur ())
```

Lineárny Haskell v praxi

- štandardná knižnica nie je lineárne otypovaná
  - fun x = x + 2 – nie lineárna lebo + nie je lineárne
  - podobne „$“, „.“ nezachováva linearitu
- treba použiť knižnicu linear-base, ktorá je lineárne otypovaná
  - fun x = x Data.Num.Linear.+ 2 -- dá sa schovať vhodným importovaním

linear-base/examples/Pure.hs

- obsahuje jednoduché príklady lineárnych a nelineárnych funkcií

- (#.) :: (b %1 -> c) -> (a %1 -> b) -> (a %1 -> c)
- g #. f = \a -> g (f a)

- linearSwap (x, y) = (y, x)

Jednoduchý lineárny Nonce — Haskell

```
data IntNonce = Nonce Int


newIntNonce' :: Prelude.IO IntNonce
newIntNonce' = do
    gen <- getStdGen
    let (val::Int, gen') = uniform gen
    setStdGen gen'
    return (Nonce val)
```

```
newIntNonce :: System.IO.Linear.IO IntNonce
newIntNonce = System.IO.Linear.fromSystemIO newIntNonce'

putIntNonce' :: IntNonce -> Prelude.IO ()
putIntNonce' (Nonce val) = do
    putStrLn $ show val
```

```
needs_new_random_every_time :: IntNonce %1 -> System.IO.Linear
    .IO ()
needs_new_random_every_time nonce = System.IO.Linear.
    fromSystemIO (putIntNonce'' nonce)

unwrapIntNonce' :: IntNonce -> Ur Int
unwrapIntNonce' (Nonce val) = Ur val
unwrapIntNonce :: IntNonce %1 -> Ur Int
unwrapIntNonce = Unsafe.Linear.coerce unwrapIntNonce'
```

```
printNewNonce :: System.IO.Linear.IO (Ur ())
printNewNonce = Control.do
    nonce <- newIntNonce -- vytvor
    () <- needs_new_random_every_time nonce -- print
    --() <- needs_new_random_every_time nonce -- print again
    Control.return (Ur ())
main :: Prelude.IO ()
main = System.IO.Linear.withLinearIO $ printNewNonce
```

Unsafe.Linear.coerce

- :: a %1 -> b
- použitá v dokumentácii pre linear knižnicu
- použitá v implementácii RIO monády

```
>>> import qualified Unsafe.Linear as Unsafe
>>> import qualified Data.Time as Time
>>> let getCurrentTime = fromSystemIO (Unsafe.coerce Time.getCurrentTime)
>>> S.print $ S.replicateM 2 getCurrentTime
2015-08-18 00:57:36.124508 UTC
2015-08-18 00:57:36.124785 UTC
```

- výmena premenných medzi argumentami funkcií je povolená, pravidlo platí
  - lineárne aj všeobecné štandardne
  - pre kombinovanú je korektný všeobecný swap (dá sa úvahou)
- oslabenie aj kontrakcia sú v jasnom rozpore s podmienkou linearity
  - pomocou coerce sa dá definovať operácie pre alternatívne verzie pravidiel

```
swapLinear1 :: (a %1 -> b -> c) %1 -> (b -> a %1 -> c)
swapLinear1 f = \b a -> f a b
swapLinear2 :: (a -> b %1 -> c) %1 -> (b %1 -> a -> c)
swapLinear2 f = \b a -> f a b
```

Lineárne typy v C++

- Linear types can save the API — Ivan Čukić
- založené na „move semantics“ (C++11)
- nesmie sa dať kopírovať, smie sa len presúvať, teda podobne ako vlastníctvo pre Rust
- object, z ktorého bolo presunuté
  - je vo valídnom (dá sa dealokovať)
  - ale nešpecifikovanom stave (nemá sa používať)
- kontrola počas kompilácie vyžaduje zapnutie use-after-move
  - clang, clang-tidy
- dá sa definovať koncept pre lineárny typ (C++20)

Koncept pre lineárny (afínny) typ

```
template <typename T, typename U>
constexpr bool linear_usable_as =
std::is_nothrow_constructible_v<T, U> and
std::is_nothrow_assignable_v<T&, U> and
std::is_nothrow_convertible_v<U, T>;

template <typename T, typename U>
constexpr bool linear_unusable_as =
not std::is_constructible_v<T, U> and
not std::is_assignable_v<T&, U> and
not std::is_convertible_v<U, T>;
```

```
template <typename T>
concept Linear =
std::is_nothrow_destructible_v<T> and
linear_usable_as<T, T> and
linear_usable_as<T, T&&> and
linear_unusable_as<T, T&> and
linear_unusable_as<T, const T&> and
linear_unusable_as<T, const T>;
```

Jednoduchý lineárny Nonce — C++

- pre zmenu demonštrujeme prístup so vzorom factory

```
random_device rd;
mt19937 rng(rd());
IntNonce new_IntNonce(){
    int32_t val = rng();
    IntNonce in(val);
    return in;
}
```

```
class IntNonce {
private:
    int32_t val;
public:
    IntNonce(int32_t value): val(value){};

    IntNonce(const IntNonce& in) = delete;
    IntNonce(IntNonce&& in) = default;
    IntNonce& operator=(const IntNonce&) = delete;
    IntNonce& operator=(IntNonce&&) = default;
    ~IntNonce(){};

    int32_t getValue() && noexcept {
        return val;
    }
};
```

```
void needs_new_random_every_time(IntNonce&& nonce){
    int32_t val = std::move(nonce).getValue();
    cout<<"Nonce: "<<val<<endl;
    cout<<"That is "<<val<<endl;
}

int main(){
    IntNonce b = new_IntNonce(); // create
    needs_new_random_every_time(std::move(b)); // print
    // needs_new_random_every_time(std::move(b));
    // needs_new_random_every_time(b);
}
```

- pravidlo kontrakcie neplatí, ak je aktívna kontrola použitia po presunutí
  - inak môže mať program nedefinované správanie
- pravidlo výmeny štandardne platí
- pravidlo oslabenia platí, pokus o zakázanie nepoužitých premenných nestačí
  - nekontroluje všetky vetvy, nezískame teda vynútenie použitia
- systém je teda afínny

Testovanie dopadu na výkon

- kompilácia aj beh
- vo všetkých prípadoch s dostatočnou optimalizáciou odstráni nadbytočný obal
  - za behu sme preto nemerali
- čas kompilácie je relevantný pre vývoj
  - väčšina výsledkov slušná
- meranie škálované počtom použití rozhrania
  - 1, 3, 6, 10, 30, 60, 100, 300, 600, 1000, 3000, 6000, 10 000, 20 000, 40 000, 60 000, 80 000 a 100 000
- merané s opakovaním do zlyhania, samostatne lineárny a jednoduchý kód, čas (a pamäť) pomocou GNU time
- G++, Clang, MSVC; Rustc
- osi časových grafov logaritmické

Od optimalizácie úrovne 2

- napríklad MSVC

```
1356    main    PROC                                            ; COMDAT
1357    $LN34:
1358            mov     QWORD PTR [rsp+8], rbx
1359            mov     QWORD PTR [rsp+16], rsi
1360            push    rdi
1361            sub     rsp, 48                                 ; 00000030H
1362            lea     rcx, OFFSET FLAT:std::mersenne_twister_engine<unsigned int,32,624,397,31,2567483615,11,429
1363            call    unsigned int std::mersenne_twister<unsigned int,32,624,397,31,2567483615,11,7,2636928640,1
1364            mov     ebx, eax
1365            lea     rdx, OFFSET FLAT:`string'
1366            lea     rcx, OFFSET FLAT:std::basic_ostream<char,std::char_traits<char> > std::cout ; std::cout
1367            call    std::basic_ostream<char,std::char_traits<char> > & std::operator<<<std::char_traits<char>
1368            mov     rcx, rax
1369            mov     edx, ebx
1370            call    std::basic_ostream<char,std::char_traits<char> > & std::basic_ostream<char,std::char_trait
1371            mov     rdi, rax
1372            mov     rcx, QWORD PTR [rax]
```

- Clang

```
62  main:                                   # @main
63          push    rax
64          lea     rdi, [rip + rng]
65          call    std::mersenne_twister_engine<unsigned long,
66          mov     dword ptr [rsp + 4], eax
67          lea     rdi, [rsp + 4]
68          call    needs_fresh(IntNonce&&)
69          xor     eax, eax
70          pop     rcx
71          ret
```

```
19  needs_fresh(IntNonce&&):                # @needs_fresh(IntNonce&&)
20          push    r14
21          push    rbx
22          push    rax
23          mov     ebx, dword ptr [rdi]
24          mov     r14, qword ptr [rip + std::cout@GOTPCREL]
25          lea     rsi, [rip + .L.str]
26          mov     edx, 7
27          mov     rdi, r14
28          call    std::basic_ostream<char, std::char_traits<char> >& std::__ostream_insert
29          mov     rdi, r14
30          mov     esi, ebx
31          call    std::basic_ostream<char, std::char_traits<char> >::operator<<(int)@PLT
32          mov     rcx, qword ptr [rax]
```

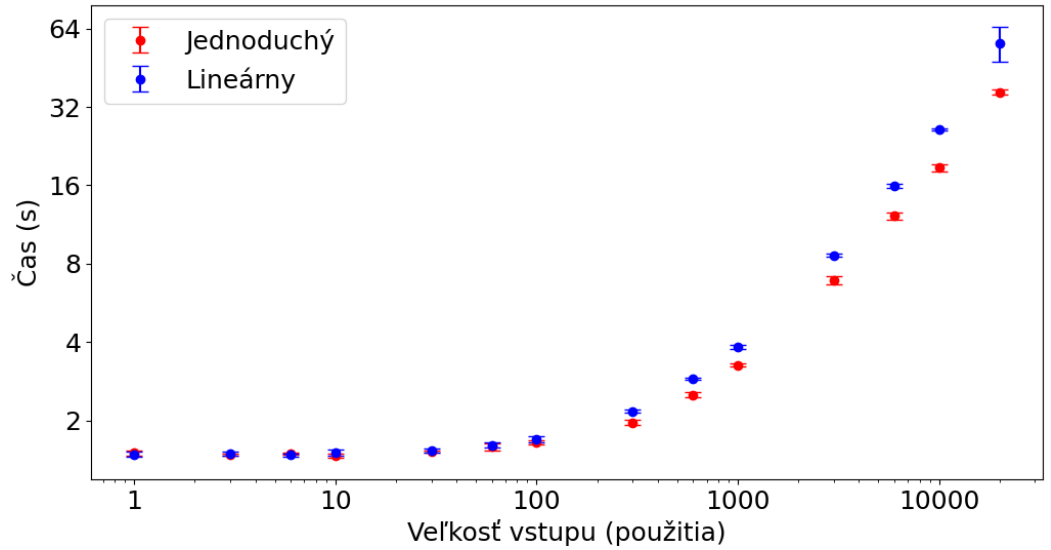
- rustc

```
78  example::Nonce::new::hd1dd94632ef96955:
79          push    r15
80          push    r14
81          push    r13
82          push    r12
83          push    rbx
84          call    qword ptr [rip + rand::rngs::thread::thread_rng
178 .LBB1_32:
179         or      r15, r12
180         mov     rax, r14
181         mov     rdx, r15
182         pop     rbx
183         pop     r12
184         pop     r13
185         pop     r14
186         pop     r15
187         ret
```

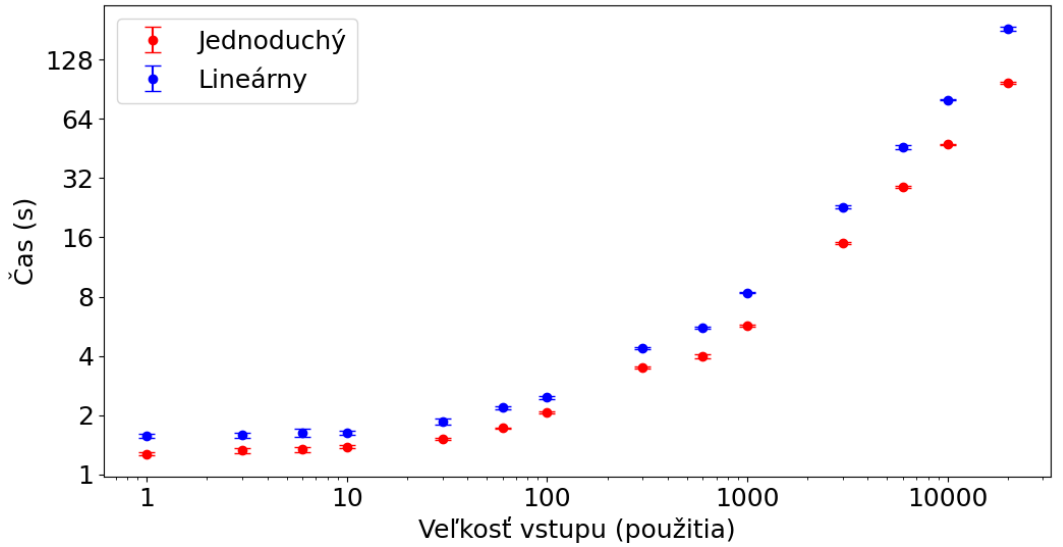
```
1   extern crate rand;
2
3   pub struct Nonce {
4       val: u128,
5   }
6
7   impl Nonce {
8   pub fn new() -> Nonce {
9       use rand::prelude::*;
10      Nonce {
11          val: random()
12      }
13  }
14
15  pub fn get(&self) -> u128 {
16      self.val
17  }
18  }
19
20  fn needs_fresh(nonce: Nonce) {
21      let tmp = nonce.get();
22      println!("Nonce: {}", tmp);
23  }
24
25  pub fn main() {
26      needs_fresh(Nonce::new());
27  }
```

G++

- O0

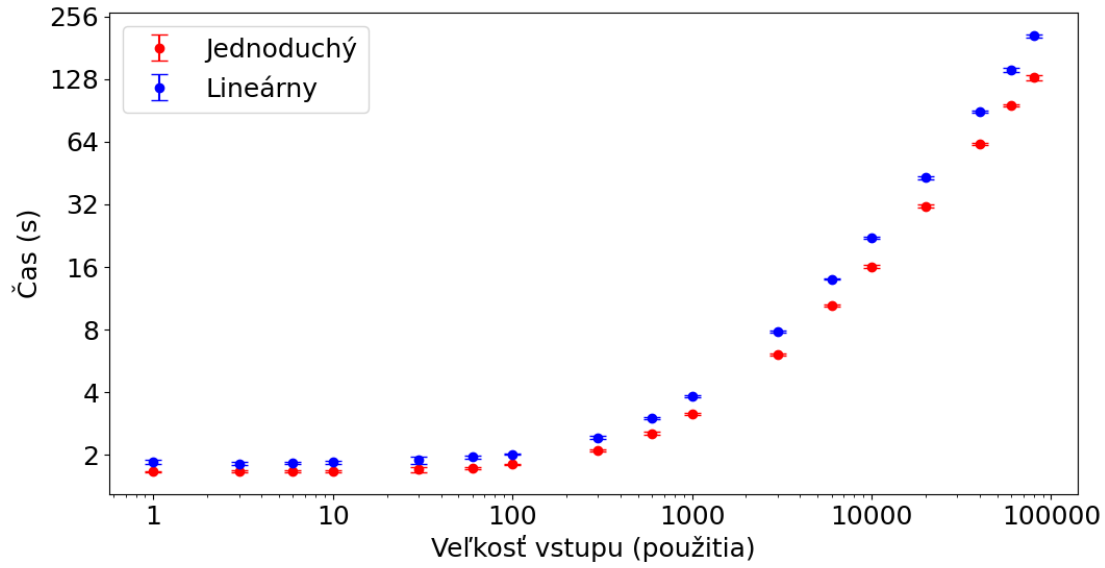

Jednoduchý
Lineárny
Čas (s)
64
32
16
8
4
2
1
10
100
1000
10000
Veľkosť vstupu (použitia)

O2

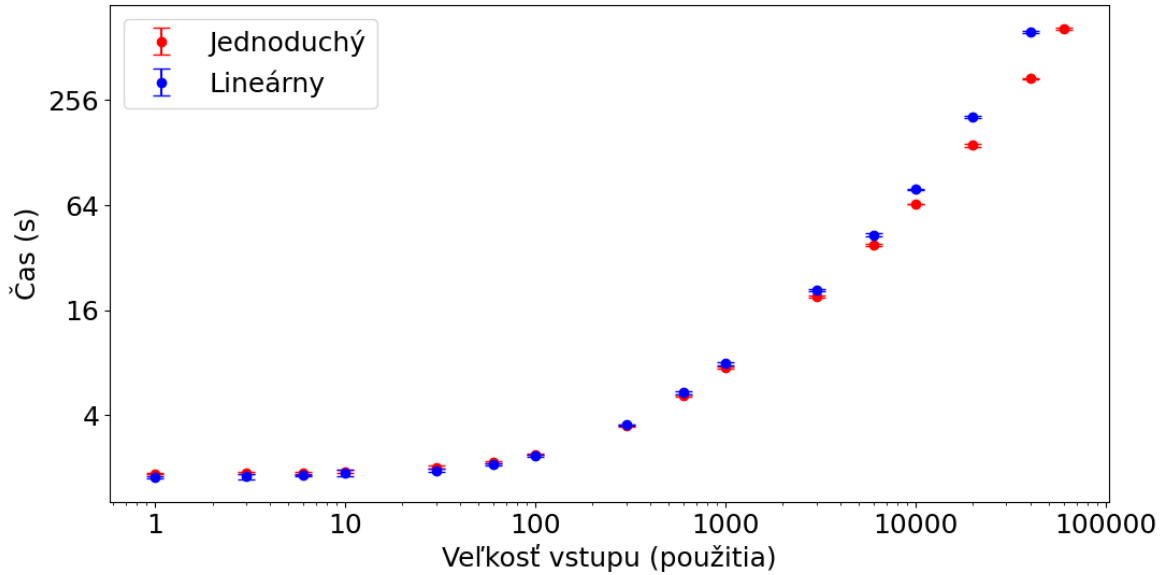

Jednoduchý
Lineárny
Čas (s)
128
64
32
16
8
4
2
1
10
100
1000
10000
Veľkosť vstupu (použitia)

Clang

- O0

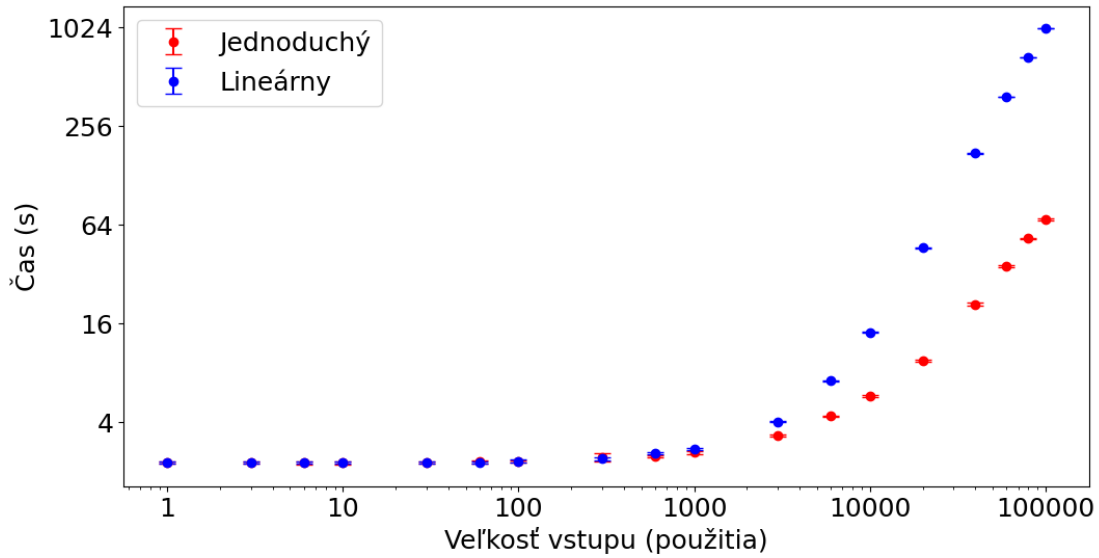

Jednoduchý
Lineárny
Čas (s)
Veľkosť vstupu (použitia)
256
128
64
32
16
8
4
2
1
10
100
1000
10000
100000

O2

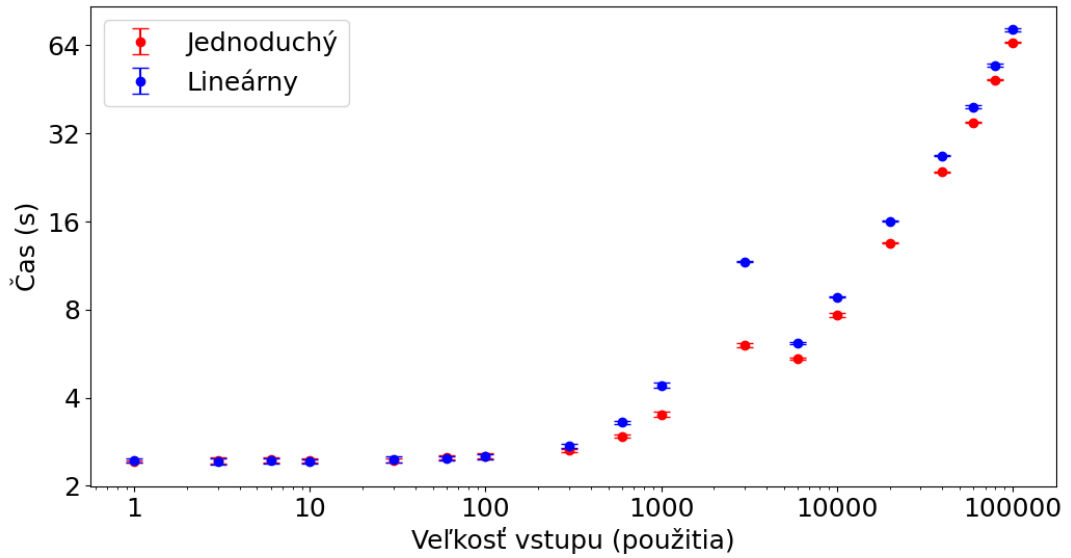

Jednoduchý
Lineárny
Čas (s)
Veľkosť vstupu (použitia)
256
64
16
4
1
10
100
1000
10000
100000

MSVC

- O0


Jednoduchý
Lineárny
Čas (s)
1024
256
64
16
4
1
10
100
1000
10000
100000
Veľkosť vstupu (použitia)

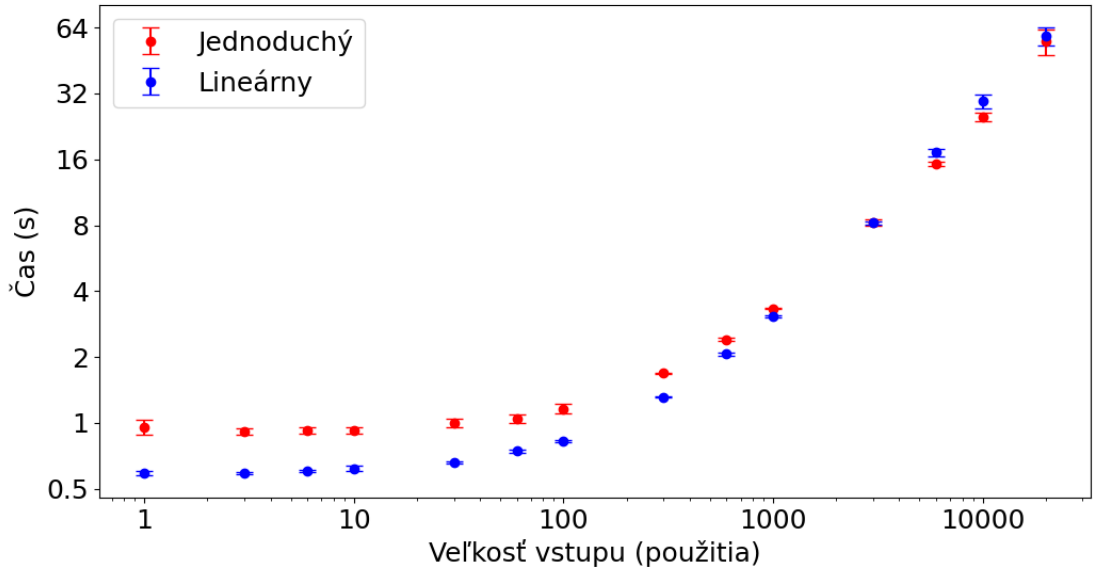
O2

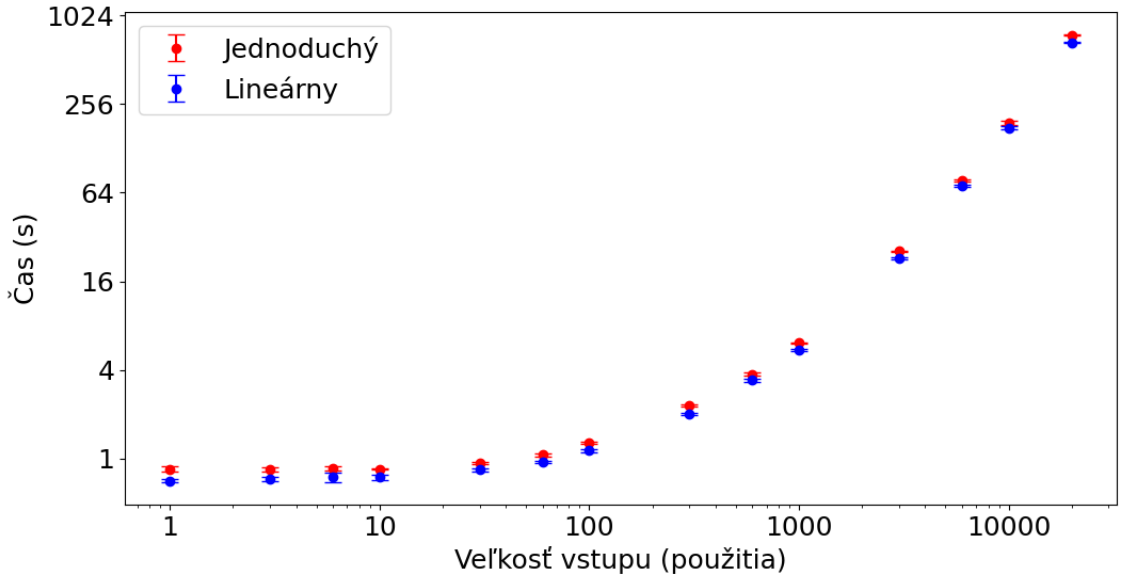

Jednoduchý
Lineárny
Čas (s)
64
32
16
8
4
2
1
10
100
1000
10000
100000
Veľkosť vstupu (použitia)

rustc

- O0

O2


Jednoduchý
Lineárny
Čas (s)
64
32
16
8
4
2
1
0.5
1
10
100
1000
10000
Veľkosť vstupu (použitia)


Jednoduchý
Lineárny
Čas (s)
1024
256
64
16
4
1
1
10
100
1000
10000
Veľkosť vstupu (použitia)

Ďakujem za pozornosť

2. Iný kód v teste

- napr. strana 32 (z textu posudku)
- kapitola 6

  Dopad lineárnych typov na výkon
- Appendix A:

  Súbory base_cislo.cpp resp. base_cislo.rs obsahujú jednoduchý kód a lin_cislo.cpp resp. lin_cislo.rs lineárny.

```cpp
#include <iostream>
#include <type_traits>
#include <memory>
#include <random>

using namespace std;

class IntNonce {
private:
    int32_t val;
public:
    IntNonce(int32_t value): val(value){};
    IntNonce(const IntNonce& in) = delete;
    IntNonce(IntNonce&& in) = default;
    IntNonce& operator=(const IntNonce&) = delete;
    IntNonce& operator=(IntNonce&&) = default;
    ~IntNonce(){};
    int32_t getValue() && noexcept {
        return val;
    }
};

random_device rd;
mt19937 rng(rd());
IntNonce new_IntNonce(){
    int32_t val = rng();
    IntNonce in(val);
    return in;
}

void needs_fresh(IntNonce&& nonce){
    int32_t val = std::move(nonce).getValue();
    cout<<"Nonce: "<<val<<endl;
}

int main(){
    needs_fresh(new_IntNonce());
}
```

```cpp
#include <iostream>
#include <type_traits>
#include <memory>
#include <random>

using namespace std;

random_device rd;
mt19937 rng(rd());
int32_t new_nonce(){
int32_t val = rng();
return val;
}
void needs_fresh_580b55fe(int32_t nonce){
cout<<"Nonce: "<<(nonce+1477137918)<<endl;
}
int main(){
needs_fresh_580b55fe(new_nonce());
}
```

```cpp
class IntNonce {
private:
int32_t val;
public:
IntNonce(int32_t value): val(value){};
IntNonce(const IntNonce& in) = delete;
IntNonce(IntNonce&& in) = default;
IntNonce& operator=(const IntNonce&) = delete;
IntNonce& operator=(IntNonce&&) = default;
~IntNonce(){};
int32_t getValue() && noexcept {
return val;
}
};
random_device rd;
mt19937 rng(rd());
IntNonce new_IntNonce(){
int32_t val = rng();
IntNonce in(val);
return in;
}
void needs_fresh_580b55fe(IntNonce&& nonce){
int32_t val = std::move(nonce).getValue();
cout<<"Nonce: "<<(val+1477137918)<<endl;
}
int main(){
needs_fresh_580b55fe(new_IntNonce());
}
```

```
8  class IntNonce {
9  private:
10     int32_t val;
11 public:
12     IntNonce(int32_t value): val(value){};
13     IntNonce(const IntNonce& in) = delete;
14     IntNonce(IntNonce&& in) = default;
15     IntNonce& operator=(const IntNonce&) = delete;
16     IntNonce& operator=(IntNonce&&) = default;
17     ~IntNonce(){};
18     int32_t getValue() && noexcept {
19         return val;
20     }
21 };
22
23 random_device rd;
24 mt19937 rng(rd());
25 IntNonce new_IntNonce(){
26     int32_t val = rng();
27     IntNonce in(val);
28     return in;
29 }
30
31 void needs_fresh(IntNonce&& nonce){
32     int32_t val = std::move(nonce).getValue();
33     cout<<"Nonce: "<<val<<endl;
34 }
35
36 int main(){
37     needs_fresh(new_IntNonce());
38 }
```

```
8  class IntNonce {
9  private:
10 int32_t val;
11 public:
12 IntNonce(int32_t value): val(value){};
13 IntNonce(const IntNonce& in) = delete;
14 IntNonce(IntNonce&& in) = default;
15 IntNonce& operator=(const IntNonce&) = delete;
16 IntNonce& operator=(IntNonce&&) = default;
17 ~IntNonce(){};
18 int32_t getValue() && noexcept {
19 return val;
20 }
21 };
22 random_device rd;
23 mt19937 rng(rd());
24 IntNonce new_IntNonce(){
25 int32_t val = rng();
26 IntNonce in(val);
27 return in;
28 }
29 void needs_fresh_580b55fe(IntNonce&& nonce){
30 int32_t val = std::move(nonce).getValue();
31 cout<<"Nonce: "<<(val+1477137918)<<endl;
32 }
33 int main(){
34 needs_fresh_580b55fe(new_IntNonce());
35 }
```

1. Čo iné použitie

- test predpokladá jednoduché korektné použitie
  - potrebujem nonce -> vyrobím -> použijem
- porovnanie priameho int bez kontroly vs. subštrukturálny prístup
- bolo by zaujímavé preskúmať komplikované prípady
  - prehľadávanie cez gramatiky
  - zaujíma nás spomalenie pre korektné prípady, skôr rýchlosť odhalenia pre nekorektné
  - nejak vhodne definovať gramatiku (syntax) aby určite bola alebo nebola splnená podmienka príležitostného slova (sémantika) a pri tom dosť veľká množina zaujímavých prípadov

```
29  void needs_fresh_30666ed7(IntNonce&& nonce){
30  int32_t val = std::move(nonce).getValue();
31  cout<<"Nonce: "<<(val+812019415)<<endl;
32  }
33  void needs_fresh_37c8fe59(IntNonce&& nonce){
34  int32_t val = std::move(nonce).getValue();
35  cout<<"Nonce: "<<(val+935919193)<<endl;
36  }
37  void needs_fresh_5b7fde8c(IntNonce&& nonce){
38  int32_t val = std::move(nonce).getValue();
39  cout<<"Nonce: "<<(val+1535106700)<<endl;
40  }
41  int main(){
42  needs_fresh_30666ed7(new_IntNonce());
43  needs_fresh_37c8fe59(new_IntNonce());
44  needs_fresh_5b7fde8c(new_IntNonce());
45  }
```

```
29  void needs_fresh_30666ed7(IntNonce&& nonce){
30  int32_t val = std::move(nonce).getValue();
31  cout<<"Nonce: "<<(val+812019415)<<endl;
32  }
33  void needs_fresh_37c8fe59(IntNonce&& nonce){
34  int32_t val = std::move(nonce).getValue();
35  cout<<"Nonce: "<<(val+935919193)<<endl;
36  }
37  void needs_fresh_5b7fde8c(IntNonce&& nonce){
38  int32_t val = std::move(nonce).getValue();
39  cout<<"Nonce: "<<(val+1535106700)<<endl;
40  }
41  int main(){
42  needs_fresh_30666ed7(new_IntNonce());
43  needs_fresh_37c8fe59(new_IntNonce());
44  needs_fresh_5b7fde8c(new_IntNonce());
45  }
```

```
29  void needs_fresh_30666ed7(IntNonce&& nonce){
30  int32_t val = std::move(nonce).getValue();
31  cout<<"Nonce: "<<(val+812019415)<<endl;
32  }
33  void needs_fresh_37c8fe59(IntNonce&& nonce){
34  int32_t val = std::move(nonce).getValue();
35  cout<<"Nonce: "<<(val+935919193)<<endl;
36  }
37  void needs_fresh_5b7fde8c(IntNonce&& nonce){
38  int32_t val = std::move(nonce).getValue();
39  cout<<"Nonce: "<<(val+1535106700)<<endl;
40  }
41  int main(){
42  needs_fresh_5b7fde8c(new_IntNonce());
43  needs_fresh_37c8fe59(new_IntNonce());
44  needs_fresh_30666ed7(new_IntNonce());
45  }
```

3. Meranie pre Haskell

- z dokumentácie k lineárnym typom v jazyku Haskell (aj v latest z 10. mája)
  - **Status:** Experimental
  - **This extension is currently considered experimental, expect bugs, warts, and bad error messages; everything down to the syntax is subject to change.**
- iná dizajnová filozofia
  - C++, Rust systémové jazyky
  - Haskell výskumný, známe motto „avoid $ success at all costs“
- Godbolt nepodporuje knižnice pre Haskell
  - https://github.com/compiler-explorer/compiler-explorer/issues/3681 linear-base
  - Hi, thanks for the request. As you might have noticed, we currently dont support libraries for Haskell, at all.