ZPP – NianioLang

Cotygodniowa prezentacja
23.11.2017

Co miało być zrobione

- Modyfikacja przydzielania rejestrów, tak żeby uniknąć niepoprawnych typów w kodzie C
- Dodanie indeksowania tablic intami
- Dokończenie zadań z zeszłego tygodnia:
  - Integracja intów
  - Implementacja przypisań na intach
  - Implementacja operacji arytmetycznych na intach

Modyfikacja rejestrów – główny cel tygodnia

- Bardzo duża zmiana
- Spowodowała ponad 400 błędów kompilacji (10% kodu translatora do zmiany)
- A wygląda tak niewinnie:

```
def nlasm::reg_t() {
-       return ptd::sim();
+       return ptd::var({
+               im => ptd::sim(),
+               int => ptd::sim(),
+               string => ptd::sim(),
+               bool => ptd::sim()
+       });
}
```

Modyfikacja rejestrów – główny cel tygodnia

- Bardzo duża zmiana
- Spowodowała ponad 400 błędów kompilacji (10% kodu translatora do zmiany)
- A wygląda tak niewinnie:

```
def nlasm::reg_t() {
-       return ptd::sim();
+       return ptd::var({
+               im => ptd::sim(),
+               int => ptd::sim(),
+               string => ptd::sim(),
+               bool => ptd::sim()
+       });
}
```

- Tylko co się tutaj stało?

Rejestry nlasm

nlasm – język pośredni podczas kompilacji NianioLanga

- Formalnie maszyna rejestrowa
- Mamy do dyspozycji nieskończenie wiele rejestrów i podstawowe operacje na nich
- Przykładowy program dla `var x = 11 + 22`:

```
registers => [0, 1, 2]
cmds => [load_const(dest => 1, val => 11),
         load_const(dest => 2, val => 22),
         bin_op(dest => 0, left => 1,
                right => 2, op => '+')]
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

{ `#reg = 0`

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
        {
                var b;              #reg = 2, a: 0, b: 1
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
        {
                var b;              #reg = 2, a: 0, b: 1
        }                           #reg = 1, a: 0
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
        {
                var b;              #reg = 2, a: 0, b: 1
        }                           #reg = 1, a: 0
        var c;                      #reg = 2, a: 0, c: 1
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
        {
                var b;              #reg = 2, a: 0, b: 1
        }                           #reg = 1, a: 0
        var c;                      #reg = 2, a: 0, c: 1
}                                   #reg = 0
```

Przydzielanie rejestrów

- Każda zmienna otrzymuje własny rejestr
- Kiedy zmienna wychodzi z zasięgu, jej rejestr jest zwalniany i może być ponownie użyty

```
{                                   #reg = 0
        var a;                      #reg = 1, a: 0
        {
                var b;              #reg = 2, a: 0, b: 1
        }                           #reg = 1, a: 0
        var c;                      #reg = 2, a: 0, c: 1
}                                   #reg = 0
```

- Problem pojawia się, kiedy zmienne `b` i `c` są różnych typów

Rozwiązanie

- Wszyskie rejestry `void*`
- **Oddzielna pula rejestrów dla każdego typu**
- **Rezygnacja z optymalizacji liczby rejestrów (GCC sobie poradzi)**

Efekt – Przed

```
void* ___nl__0 = NULL;
void* ___nl__1 = NULL;
#line 5 // var i = 11;

c_rt_lib0move(&___nl__1,___get_global_const(57));
c_rt_lib0copy(&___nl__0, ___nl__1);
c_rt_lib0clear(&___nl__1);

#line 6 // i = i + i;
c_rt_lib0move(&___nl__1, c_rt_lib0add(___nl__0, ___nl__0));
c_rt_lib0copy(&___nl__0, ___nl__1);
c_rt_lib0clear(&___nl__1);
c_rt_lib0clear(&___nl__0);
```

Efekt – Po

```
INT ___nl__int__0 = 0;
INT ___nl__int__1 = 0;

#line 5 // var i = 11;
___nl__int__1 = 11;
___nl__int__0 = ___nl__int__1;

#line 6 // i = i + i
___nl__int__1 = ___nl__int__0 + ___nl__int__0;
___nl__int__0 = ___nl__int__1;
```

Plany na najbliższy tydzień

- Doprowadzenie do skompilowania się istniejących testów
- Dodanie obsługi intów jako argumentów i wartości zwracanych funkcji
- Początek pracy nad kompilacją rekordów i tablic
- Zapisywanie odgadniętych przez type checker typów w drzewie AST