AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Projekt języka programowania wspierającego przetwarzanie rozproszone na platformach heterogenicznych.

Kajetan Rzepecki

Wydział EAIiIB
Katedra Informatyki Stosowanej

28 września 2015

Problem - Na czym się skupiamy

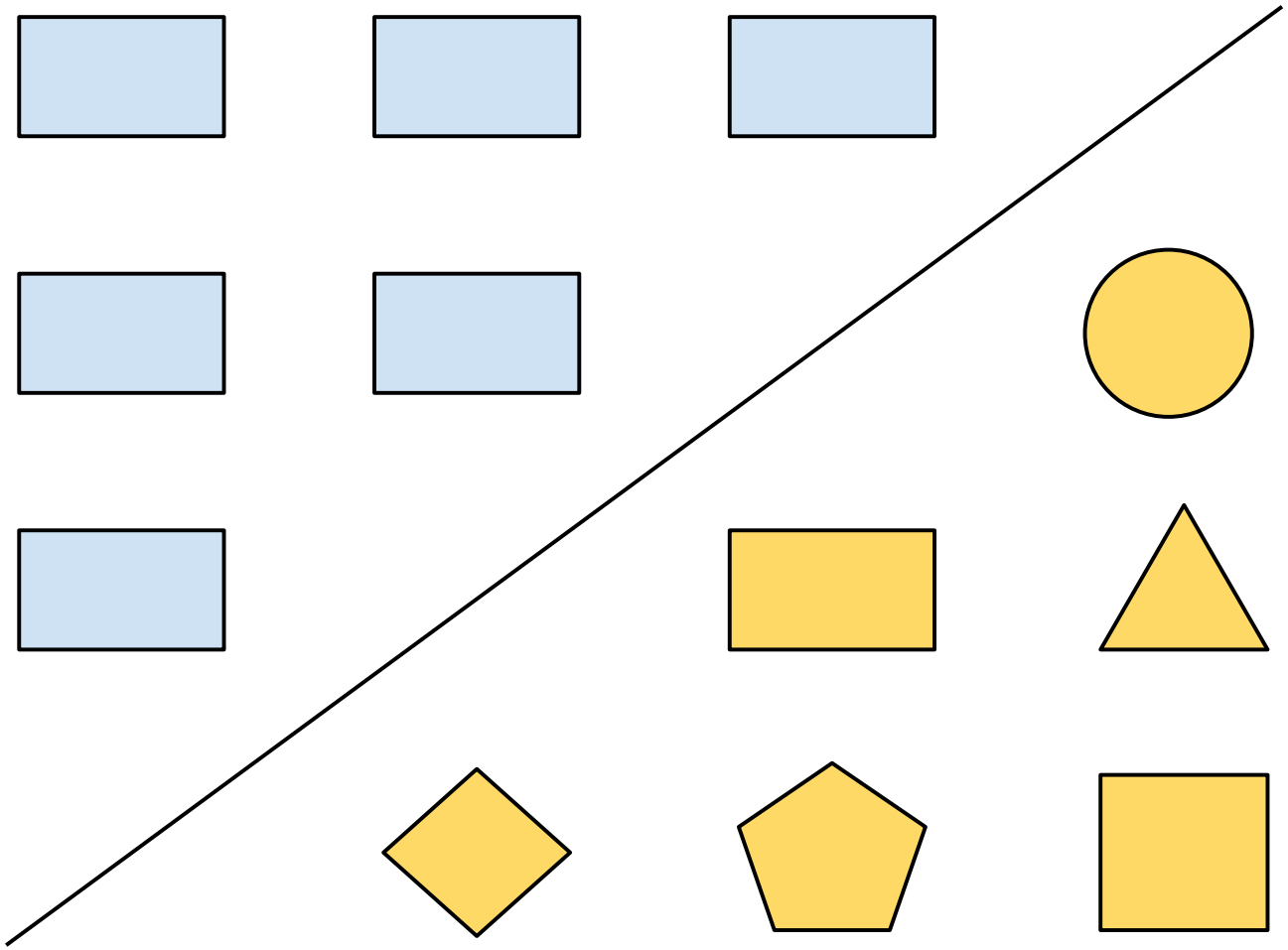
- Skalowalność
- Dynamiczność
- Niezawodność
- Konfiguracja

Problem - Na czym się skupiamy

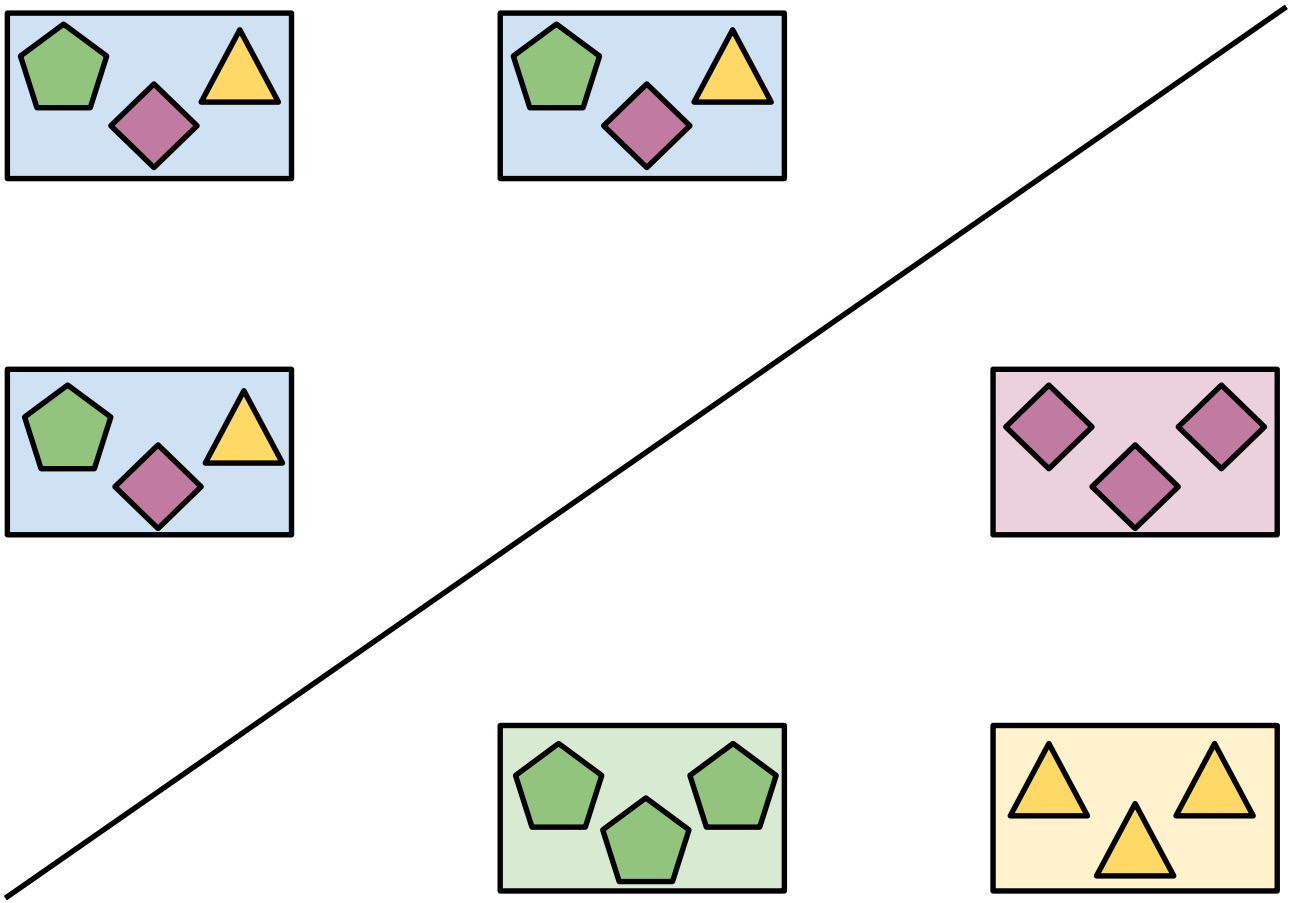
- Skalowalność
- Dynamiczność
- Niezawodność
- Konfiguracja

- Heterogeniczność?
- Świadomość platformy?

Problem - Heterogeniczność



Problem - Heterogeniczność



Problem - Heterogeniczność

Problem - State of the Art

Problem - State of the Art

- Heterogeniczność?
- Świadomość platformy?

Problem - Cel pracy

- Rozwiązanie problemu heterogeniczności

Problem - Cel pracy

- Rozwiązanie problemu heterogeniczności
- Projekt języka programowania

Problem - Cel pracy

- Rozwiązanie problemu heterogeniczności
- Projekt języka programowania
- Kompilator
- Środowisko uruchomieniowe wspierające przetwarzanie rozproszone
- System modułowy

Problem - Cel pracy

- Rozwiązanie problemu heterogeniczności
- Projekt języka programowania
- Kompilator
- Środowisko uruchomieniowe wspierające przetwarzanie rozproszone
- System modułowy
- Integracja z wieloma platformami

Projekt - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami

Projekt - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

Projekt - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

- Kontynuacje:

```
(+ 1 (reset (* 2 (shift k (k (k 4)))))) ; 17 wtf!?
```

Projekt - Prosty rdzeń języka

- Przenośny i integrowalny z wieloma platformami
- λ calculus - funkcje:

```
(lambda (x) x)
```

- Kontynuacje:

```
(+ 1 (reset (* 2 (shift k (k (k 4)))))) ; 17 wtf!?
```

- Model Aktorowy - procesy:

```
(send (spawn do-something) 'message)
```

Projekt - System modułowy

- Moduł dostarcza funkcjonalność vs. moduł wymaga funkcjonalności:

```
(module FProvider
  (provide f)
  (function (f) ...))
(module FUser
  (require FProvider))
```

Projekt - System modułowy

- Moduł dostarcza funkcjonalność vs. moduł wymaga funkcjonalności:

```
(module FProvider
  (provide f)
  (function (f) ...))
(module FUser
  (require FProvider))
```

```
(module FProvider
  (function (f) ...))
(module FUser
  (require ?m
    (and (module ?m)
         (provides ?m f))))
```

Projekt - System modułowy

- Moduł dostarcza funkcjonalność vs. moduł wymaga funkcjonalności:

```
(module FProvider
  (provide f)
  (function (f) ...))
(module FUser
  (require FProvider))
```

```
(module FProvider
  (function (f) ...))
(module FUser
  (require ?m
    (and (module ?m)
         (provides ?m f))))
```

- Oparty o reguły:

```
(whenever (and (module ?m) (provides ?m feature))
  (start-using ?m))
```

Projekt - Inżynieria wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                    (module Foo)
  (function (bar) ...))        (provides Foo bar)
```

Projekt - Inżynieria wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                    (module Foo)
  (function (bar) ...))        (provides Foo bar)
```

- Heterogeniczność i świadomość platformy

```
(@ O-time "N^2")               (@ O-time "NlogN")
(@ O-space "1")                (@ O-space "N")
(function (insertion-sort xs)  (function (merge-sort xs)
  ...)                           ...)
```

Projekt - Inżynieria wiedzy

- Automatyczne odkrywanie i propagacja wiedzy

```
(module Foo                    (module Foo)
  (function (bar) ...))        (provides Foo bar)
```

- Heterogeniczność i świadomość platformy

```
(@ O-time "N^2")               (@ O-time "NlogN")
(@ O-space "1")                (@ O-space "N")
(function (insertion-sort xs)  (function (merge-sort xs)
  ...)                           ...)
```

- Dynamiczność, niezawodność i skalowalność

Implementacja - Kompilator

- Makroekspansja:

```
;; Przed makroekspansją
(and A B)
(or A B)
```

```
;; Po makroekspansji
(if A B false)
(if A true B)
```

Implementacja - Kompilator

- Makroekspansja:

```
;; Przed makroekspansją
(and A B)
(or A B)
```

```
;; Po makroekspansji
(if A B false)
(if A true B)
```

- Transformacja *Continuation Passing Style*:

```
;; Przed transformacją
(lambda (x)
  (* 2 x))
```

```
;; Po transformacji
(lambda (x cont)
  (%MULT 2 x cont))
```

Implementacja - Kompilator

- Makroekspansja:

```
;; Przed makroekspansją
(and A B)
(or A B)
```

```
;; Po makroekspansji
(if A B false)
(if A true B)
```

- Transformacja *Continuation Passing Style*:

```
;; Przed transformacją
(lambda (x)
  (* 2 x))
```

```
;; Po transformacji
(lambda (x cont)
  (%MULT 2 x cont))
```

- Wynikowy kod jest podzbiorem języka Scheme

Implementacja - Środowisko uruchomieniowe

- Integracja ze środowiskiem uruchomieniowym języka Scheme

Implementacja - Środowisko uruchomieniowe

- Integracja ze środowiskiem uruchomieniowym języka Scheme
- Model Aktorowy:

```
(spawn fun)
(self)
(send pid message)
(recv)
(sleep time)
```

Implementacja - Środowisko uruchomieniowe

- Integracja ze środowiskiem uruchomieniowym języka Scheme
- Model Aktorowy:

```
(spawn fun)
(self)
(send pid message)
(recv)
(sleep time)
```

- Algorytm *Completely Fair Scheduler*

Implementacja - System regułowy

- Algorytm Rete z autorskimi rozszerzeniami

Implementacja - System regułowy

- Algorytm Rete z autorskimi rozszerzeniami
- Wnioskowanie w przód:

```
(whenever (and (module ?m)
               (provides ?m ?f))
  (list ?m ?f))
```

Implementacja - System regułowy

- Algorytm Rete z autorskimi rozszerzeniami
- Wnioskowanie w przód:

```
(whenever (and (module ?m)
               (provides ?m ?f))
  (list ?m ?f))
```

- Wnioskowanie wstecz:

```
(select (?m ?f)
        (and (module ?m)
             (provides ?m ?f)))
```

Podsumowanie - Co udało się wykonać

- Projekt języka programowania

Podsumowanie - Co udało się wykonać

- Projekt języka programowania
- Kompilator
- Środowisko uruchomieniowe
- System modułowy

Podsumowanie - Co udało się wykonać

- Projekt języka programowania
- Kompilator
- Środowisko uruchomieniowe
- System modułowy
- Inżynieria wiedzy

Podsumowanie - Co udało się wykonać

- Projekt języka programowania
- Kompilator
- Środowisko uruchomieniowe
- System modułowy
- Inżynieria wiedzy
- Rozwiązanie problemu heterogeniczności i świadomości platformy

Podsumowanie - Przyszłe kierunki rozwoju

- Kompilacja z wykorzystaniem LLVM

Podsumowanie - Przyszłe kierunki rozwoju

- Kompilacja z wykorzystaniem LLVM
- Lepsza integracja systemu regułowego

Podsumowanie - Przyszłe kierunki rozwoju

- Kompilacja z wykorzystaniem LLVM
- Lepsza integracja systemu regułowego
- Rozszerzenie wykorzystania inżynierii wiedzy

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Dziękuję za uwagę.

Kajetan Rzepecki