Idris

Język funkcyjny z typami zależnymi

7 czerwca 2016

Idris

- rozwijany od około 2008 roku
- http://www.idris-lang.org/
- główny projektant i wykonawca to Edwin Brady z University of St Andrews.
- autor powstającej na bieżąco książki “Type-driven development with Idris”
- Idris (chyba) jeszcze nie ma dobrze sprawdzonych podstaw teoretycznych

Idris

- rozwijany od około 2008 roku
- http://www.idris-lang.org/
- główny projektant i wykonawca to Edwin Brady z University of St Andrews.
- autor powstającej na bieżąco książki “Type-driven development with Idris”
- Idris (chyba) jeszcze nie ma dobrze sprawdzonych podstaw teoretycznych

Idris

- rozwijany od około 2008 roku
- http://www.idris-lang.org/
- główny projektant i wykonawca to Edwin Brady z University of St Andrews.
- autor powstającej na bieżąco książki "Type-driven development with Idris"
- Idris (chyba) jeszcze nie ma dobrze sprawdzonych podstaw teoretycznych

Idris

- rozwijany od około 2008 roku
- http://www.idris-lang.org/
- główny projektant i wykonawca to Edwin Brady z University of St Andrews.
- autor powstającej na bieżąco książki “Type-driven development with Idris”
- Idris (chyba) jeszcze nie ma dobrze sprawdzonych podstaw teoretycznych

Idris

- rozwijany od około 2008 roku
- http://www.idris-lang.org/
- główny projektant i wykonawca to Edwin Brady z University of St Andrews.
- autor powstającej na bieżąco książki “Type-driven development with Idris”
- Idris (chyba) jeszcze nie ma dobrze sprawdzonych podstaw teoretycznych

Typy zależne i zalezny pattern-matching

- typy zależą od wartości
- nie ma syntaktycznej różnicy między typem i wartością
- typy są wartościami
- można pisać funkcje, dla których typ wyniku zależy od wartości argumentu

Typy zależne i zalezny pattern-matching

- typy zależą od wartości
- nie ma syntaktycznej różnicy między typem i wartością
- typy są wartościami
- można pisać funkcje, dla których typ wyniku zależy od wartości argumentu

Typy zależne i zalezny pattern-matching

- typy zależą od wartości
- nie ma syntaktycznej różnicy między typem i wartością
- typy są wartościami
- można pisać funkcje, dla których typ wyniku zależy od wartości argumentu

Typy zależne i zalezny pattern-matching

- typy zależą od wartości
- nie ma syntaktycznej różnicy między typem i wartością
- typy są wartościami
- można pisać funkcje, dla których typ wyniku zależy od wartości argumentu

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak “solving a puzzle”
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające “size change principle”)
- przy generowaniu programu wykonywalnego kompilator “odrzuca” części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - czysty język funkcyjny

- rozbudowany system typów służy specyfikowaniu i lepszemu/łatwiejszemu implementowaniu funkcji
- znajdowanie programu spełniającego specyfikację jest jak "solving a puzzle"
- prawdziwy język programowania (można kompilować i uruchamiać programy:)
- składniowo dosyć podobny do Haskella, ale są zamienione znaczenia : i ::, type classes nazywają się interfejsami ...
- gorliwa strategia obliczania, choć możliwe jest wprowadzenie leniwości
- z powodu typów zależnych potrzebna jest ewaluacja przy sprawdzaniu typów (powiedzmy taka jak w Coqu w konwersji)
- funkcje potrzebne do typowania muszą być całkowite (total) i terminujące (spełniające "size change principle")
- przy generowaniu programu wykonywalnego kompilator "odrzuca" części potrzebne tylko do typowania

Idris - sprawdzanie typów, kompilacja

- wejście do repl poprzez idris plik.idr
- tam można używać komend, np :t, :q (:? to ich lista)
- kompilacja porzez idris -o plik plik.idr
- lub w repl :c plik i :exec

Idris - sprawdzanie typów, kompilacja

- wejście do repl poprzez idris plik.idr
- tam można używać komend, np :t, :q (:? to ich lista)
- kompilacja porzez idris -o plik plik.idr
- lub w repl :c plik i :exec

Idris - sprawdzanie typów, kompilacja

- wejście do repl poprzez idris plik.idr
- tam można używać komend, np :t, :q (:? to ich lista)
- kompilacja porzez idris -o plik plik.idr
- lub w repl :c plik i :exec

Idris - sprawdzanie typów, kompilacja

- wejście do repl poprzez idris plik.idr
- tam można używać komend, np :t, :q (:? to ich lista)
- kompilacja porzez idris -o plik plik.idr
- lub w repl :c plik i :exec

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr

Idris i typy zależne - przegląd możliwości

- hello.idr
- basic2.idr
- plus_theorem.idr
- parity.idr
- Parity.v in Coq
- binary.idr
- interp-tutorial.idr
- Interp.v in Coq
- removeElem.idr