Seminarium: programowanie w teorii typów

1. Proste przykłady zastosowań typów zależnych w Agdzie

- Przykłady funkcji spełniających proste specyfikacje (np. filter, o której przez konstrukcję wiemy, że zwraca podlistę listy wejściowej [1])
- Przykłady struktur danych zachowujących proste niezmienniki, które normalnie istniałyby tylko w głowie implementującego (np. drzewa BST [2])
- Ilustracja mocy wyrazu typów zależnych, która pozwala na napisanie programu ładniejszego w porównaniu do tego, który byłby napisany w Haskellu lub MLu [3].

Literatura:

1. Dependently Typed Programming in Agda - Ulf Norell
2. Dependent Types at Work - Ana Bove and Peter Dybjer
3. CPDT. Chapter 7. More Dependent Types. A Tagless Interpreter - Adam Chlipała (tutaj dostarczymy tłumaczenie programu na Agdę).

2. Piękno i elegancja typów zależnych

- Pokazanie przykładów z klasycznych prac, które demonstrują ważne i praktyczne zastosowania typów zależnych ([1], [2]). Głównym trzonem prezentacji powinna być raczej praca [1] i należałoby przedstawić pełną konstrukcję otrzymanego ostatecznego programu razem z rozumowaniem/ideą, która za nią stoi.

Literatura:

1. Why Dependent Types Matter - McBride, McKinna, Swierstra
2. The Power of pi - Swierstra, Oury

3. Gdy rekursja strukturalna może nie wystarczać

Prezentacja problemu i kilku technik radzenia sobie sobie w sytuacjach, gdy naiwna implementacja nie spełnia rygorystycznych wymagań języka co do dozwolonych schematów rekursji:

- dodanie dodatkowego argumentu - limitu rekurencyjnych wywołań [1]
- indukcja względem indeksu indeksowanej struktury danych [2]
- odkrycie struktury względem której faktycznie chcielibyśmy zrobić indukcję i zmaterializowanie jej w postaci typu indukcyjnego [1,2,3,4]
- dobrze ufundowana indukcja: teoria i praktyka [1,5,6]
- (opcjonalnie) wspomnienie o ogólnej rekursji w oparciu o prace Any Bove (chociaż sam przykład użycia indukcji-rekursji jako rozszerzenia jednej z powyższych technik)

Literatura:

1. Coq'Art. Chapter 15. General recursion[1].
2. Unification by structural induction - McBride
3. Why Dependent Types Matter - McBride, McKinna, Swierstra
4. Strong Functional Programming - Turner
5. Slajdy o dobrze ufundowanej indukcji w Agdzie Erica Mertensa
6. Biblioteka standardowa Agdy - Induction.WellFounded
7. General Recursion in Type Theory - A. Bove
8. Constructing Recursion Operators in Intuitionistic Type Theory

[1] w tej książce pojawia się magiczne uniwersum Prop. Na nasze potrzeby możemy udawać, że Prop w Coqu = Set w Agdzie

4. Koindukcja

- Czym jest koindukcja?
- Przykłady wnioskowań przed koindukcję i programów definiowanych przez korekursję.
- Używanie koindukcji w Agdzie[2]. Guarded coinduction. Implementacja klasycznych koindukcyjnych struktur danych - strumienie, kolisty, nieskończone drzewa binarne.
- Techniki pozwalające radzić sobie, gdy naturalna (naiwna) implementacja nie jest akceptowana przez Agdę, gdyż wykracza poza schemat guarded coinduction. [3,5]
- Definicje indukcyjno-koindukcyjne: przykłady zastosowań [3,4]

Literatura:

1. Strong Functional Programming - David Turner
2. Artykuł o koinducji na Agda wiki:
   `http://wiki.portal.chalmers.se/agda/pmwiki.php?n=ReferenceManual.Codatatypes`
3. Mixing Induction and Coinduction - Nils Anders Danielsson, Thorsten Altenkirch
4. Total Parser Combinators - Nils Anders Danielsson
5. Beating the Productivity Checker Using Embedded Languages - Nils Anders Danielsson

Literatura dodatkowa:

1. General Recursion via Coinductive Types - Venanzio Capretta
2. Certified Programming with Dependent Types (Chapter 5. Infinite Data and Proofs) - Adam Chlipała
3. Coq'Art (Rozdział 13.) - Yves Bertot, Pierre Castéran.

5. Widoki

- Pojęcie widoku w ujęciu "klasycznym", czyli w kontekście języków funkcyjnych [1].
- Przedstawienie widoków jako rozszerzenia kompilatora GHC dla Haskella [2].
- Wytłumaczenie, dlaczego widoki w kontekście programowania z typami zależnymi nabierają rumieńców [3].
- Przedstawienie różnych przykładów struktur danych i widoków dla nich [3,4,5].

Literatura:

1. Views: A way for pattern matching to cohabit with data abstraction - Wadler (1986)
2. A new view of guards - Peyton Jones (www, 1997)
3. The view from the left - McBride, McKinna
4. Strong Functional Programming - Turner
5. Why Dependent Types Matter - McBride, McKinna, Swierstra

6. Reprezentacja języków z wiązaniem zmiennych

- Różne podejścia do reprezentacji wiązań.
- Dlaczego nie wystarcza nam naiwna reprezentacja (np. zmienne reprezentowane poprzez typ string)?
- (Opcjonalnie) jak wygląda sprawa dowodzenia twierdzeń na temat języka z konstrukcjami wiążącymi zmienne (np. lambda abstrakcje)?

[2]Chodzi nam o Agdę 2.2.10 i nowsze. We wcześniejszych wersjach używało się słowa kluczowego codata - chcemy posłuchać o aktualnym sposobie (wbudowane operacje SHARP i FLAT, patrz [2], [3], [4]).

Literatura:

1. CPDT - Adam Chlipala
2. Parametric Higher-Order Abstract Syntax for Mechanized Semantics - Adam Chlipala
3. Nameless, Painless - Nicolas Pouillard
4. A fresh look at programming with names and binders - Nicolas Pouillard, François Pottier
5. Strongly-typed term representations in Coq - N. Benton, Ch.-K. Hur, A. Kennedy, C. McBride
6. Name binding blog:
   `http://namebinding.wordpress.com/`

7. Typy zależne w Haskellu

- Omówienie (tak powszechnie używanych, że już częściowo zastąpionych przez nowsze pomysły) rozszerzeń Haskella 98 [1], które potrzebne będą, aby zrozumieć pracę [2].
- Pokazanie jak można symulować typy zależne w Haskellu za pomocą wieloparametrycznych klas typów i zależności funkcyjnych [2].
- Przedstawienie uogólnionych typów algebraicznych (GADT). Porównanie ich możliwości z rodzinami indukcyjnymi z Agdy.
- Przedstawienie SHE, czyli próby dorobienia pewnych elementów typów zależnych do Haskella (język... tłumaczony do Haskella!) [3,4]

Literatura:

1. Type classes with functional dependencies - Mark P. Jones (2000)
2. Faking It: Simulating Dependent Types in Haskell - Conor McBride (2001)
3. The Strathclyde Haskell Enhancement
4. Wykład 1. z kursu Dependently Typed Programming używający SHE jako ilustracji
5. Materiały internetowe, m.in.
   `http://en.wikibooks.org/wiki/Haskell/GADT`

8. Programowanie generyczne

- Użycie uniwersów do pisania algorytmów niezależących od konkretnego typu danych.
- Uniwersum dla prostych typów indukcyjnych, wraz z przykładowymi programami jak generyczna równość, generyczny map.
- Bardziej zaawansowane kodowania: rodziny indukcyjne, kontenery, zippery itp.
- Pisanie funkcji z różną arnością (eliminacja problemu z powtarzaniem schematu zipWith, zipWith3, itd) [5].
- (Opcjonalnie) Zastosowanie mechanizmu w praktyce: Generyczny i certyfikowany algorytm diff [1].

Literatura:

1. Type-safe Diff for Families of Datatypes - E. Lempsink, S. Leather, A. Loh
2. Generic programming with Dependent Types - T. Altenkirch, C. McBride, P. Morris
3. Generic programming with Indexed Functors - A. Loh, J. P. Magalhaes
4. Constructing Universes for Generic Programming - P. Morris
5. Arity-Generic Datatype-Generic Programming - S. Weirich, C.Casinghino
6. Exploring the regular tree type -

1 XXX

Opis:
Literatura:

x

-

Literatura:

1.