Kwalifikacja i implementacja systemów kompilacji z użyciem efektów algebraicznych

(Categorization and implementation of Build Systems using algebraic effects)

Jakub Mendyk

Praca licencjacka

Promotor: dr Filip Sieczkowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

4 września 2020

Streszczenie

. . .

. . .

Spis treści

1. Wprowadzenie **7**

1.1. Problemy z efektami ubocznymi 7

1.2. Radzenie sobie z efektami ubocznymi 7

1.3. Systemy kompilacji 8

1.4. O tej pracy 8

2. O efektach teoretycznie **9**

3. O systemach kompilacji (i ich klasyfikacji) **11**

4. Efekty algebraiczne i uchwyty w praktyce **13**

5. Systemy kompilacji z użyciem efektów algebraicznych i uchwytów **15**

6. Podsumowanie i wnioski **17**

Bibliografia **19**

Rozdział 1.

Wprowadzenie

1.1. Problemy z efektami ubocznymi

Programy komputerowe, dzięki możliwości interakcji z zewnętrznymi zasobami – takimi jak nośniki pamięci, sieci komputerowe czy użytkownicy oprogramowania – mogą robić istotnie więcej niż tylko zadane wcześniej obliczenia. W ten sposób przebieg programu i jego końcowy wynik staje się jednak zależny od tegoż świata zewnętrznego, a sam program nie tylko serią czystych obliczeń ale także towarzyszących im efektów ubocznych.

Efekty uboczne powodują jednak, że rozumowanie i wnioskowanie o sposobie oraz prawidłowości działania programów staje się znacznie trudniejsze, a w konsekwencji ogranicza ich modularność i prowadzi do częstszych pomyłek ze strony autorów. Chcąc tego uniknąć, dąży się do wydzielania w programie jak największej części, która składa się z czystych obliczeń. Jednak to, czy jakiś moduł oprogramowania wykonuje obliczenia z efektami ubocznymi nie koniecznie jest jasne i często musimy zaufać autorowi, że w istocie tak jest.

1.2. Radzenie sobie z efektami ubocznymi

Jednym z rozwiązań tego problemu, jest zawarcie informacji o posiadaniu efektów ubocznych w systemie typów. Możemy wykorzystać wtedy dedukcji i weryfikacji typów do automatycznej identyfikacji modułów zawierających efekty uboczne. Programista może łatwo wyczytać z sygnatury funkcji, że w czasiej jej działania występują efekty uboczne. Znanym przykładem takiego rozwiązania jest wykorzystanie monad w języku programowania Haskell. Niestety, jednoczesne użytkowanie dwóch niezależnych zasób reprezentowanych przez różne monady nie jest możliwe i wymaga dodatkowych struktur, takich jak transformery monad, które niosą ze sobą dodatkowe problemy. Problem modularności został jedynie przesunięty w inny obszar.

Nowym, konkurencyjnym podejściem do ujarzmienia efektów ubocznych przez wykorzystanie systemu typów są efekty algebraiczne z uchwytami. Powierzchownie, zdają się być podobne do konstrukcji obsługi wyjątków w językach programowania lub wywołań systemowych w systemach operacyjnych. Dzięki rozdziałowi między definicjami operacji związanych z efektami ubocznymi, a ich sematyką oraz interesującemu zastosowaniu kontynuacji, dają łatwość myślenia i wnioskowania o programach ich używających. Ponadto, w przeciwieństwie do monad, można je bezproblemowo składać.

1.3. Systemy kompilacji

Przykładami programów, których głównym zadaniem jest interakcja z zewnętrznymi zasobami są systemy kompilacji, w których użytkownik opisuje proces wytwarzania wyniku jako zbiór wzajemnie-zależnych zadań, wraz z informacją jak zadania są wykonywane w oparciu o wyniki innych podzadań, a system jest odpowiedzialny za poprawne uporządkowanie i wykonanie otrzymanych zadań. W czasie działania, system agreguje wyniki obliczeń (np. na dysku lub w pamięci) i decyduje, która zadania powinny być obliczone ponownie – np. system Make lub popularne narzędzie biurowe Excel.

W publikacjach pod tytułem „Build systems à la carte" [5] [4], autorzy przedstawiają sposób klasyfikacji systemów kompilacji w oparciu o to jak determinują one kolejność w jakiej zadania zostaną obliczone oraz jak wyznaczają, które z zadań wymagają ponownego obliczenia. Uzyskana klasyfikacja prowadzi autorów do skonstruowania platformy umożliwiającej konstrukcję systemów kompilacji o oczekiwanych właściwościach. Platforma ta okazuje się być łatwa w implementacji w języku Haskell, a klasy typów Applicative oraz Functor odpowiadać mocy języka opisywnia zależności między zadaniami do obliczenia.

1.4. O tej pracy

Celem tej pracy jest zapoznanie czytelnika, który miał dotychczas kontakt z językiem Haskell oraz podstawami języków funkcyjny, z nowatorskim rozwiązaniem jakim są efekty algebraiczne oraz zademonstrowanie – idąc śladami Makhov i innych [5] – implementacji systemów kompilacji z wykorzystaniem efektów algebraicznych i uchwytów w języku programowania Helium. Jak się okazuje, wykorzystanie tych narzędzi daje schludną implementację ale także prowadzi do problemów w implementacji systemów z topologicznym planistą.

Rozdział 2.

O efektach teoretycznie

Rozdział 3.

O systemach kompilacji (i ich klasyfikacji)

Rozdział 4.

Efekty algebraiczne i uchwyty w praktyce

Rozdział 5.

Systemy kompilacji z użyciem efektów algebraicznych i uchwytów

Rozdział 6.

Podsumowanie i wnioski

...

Bibliografia

[1] D. Biernacki, M. Piróg, P. Polesiuk, and F. Sieczkowski. Handle with care: relational interpretation of algebraic effects and handlers. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–30, 2017.

[2] D. Biernacki, M. Piróg, P. Polesiuk, and F. Sieczkowski. Abstracting algebraic effects. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–28, 2019.

[3] D. Biernacki, M. Piróg, P. Polesiuk, and F. Sieczkowski. Binders by day, labels by night: effect instances via lexically scoped handlers. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–29, 2019.

[4] A. Mokhov, N. Mitchell, and S. P. Jones. Build systems à la carte: Theory and practice. *Journal of Functional Programming*, 30, 2020.

[5] A. Mokhov, N. Mitchell, and S. Peyton Jones. Build systems à la carte. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29, 2018.

[6] M. Pretnar. An introduction to algebraic effects and handlers. invited tutorial paper. *Electronic notes in theoretical computer science*, 319:19–35, 2015.