Konstrukcja kompilatorów

Projekt nr 1

Termin oddawania: 9 listopada 2018

UWAGA! Przed przystąpieniem do zadania proszę zapoznać się ze stronami w systemie SKOS poświęconymi pracowni. Znajdziecie tam Państwo między innymi opis języka oraz instrukcje obsługi kompilatora.

W pierwszym projekcie na pracownię z konstrukcji kompilatorów zaimplementują Państwo lekser i parser do języka Xi. Nie chcieliśmy aby Państwo skupiali się na technikaliach związanych z obsługą wygenerowanego leksera/parsera i postanowiliśmy Państwa odciążyć w tej kwestii. W katalogu «`source/mod_student`» znajdują się przygotowane przez nas pliki «`lexer.mll`» oraz «`parser.mly`» będace szablonami plików wejściowych dla generatorów. Zaznaczyliśmy w nich miejsca, które Państwo muszą uzupełnić. Dostarczony przez nas kod wtyczki jest już dostosowany do tego, aby kompilator mógł posłużyć się Państwa lekserem i parserem.
W udostępnionym przez nas kodzie w katalogu «`tests/pracownia1`» znajdą Państwo pliki «`parse_ok.xi`» oraz «`parse_error.xi`» zawierające w sobie przykłady składniowo poprawnych i niepoprawnych programów. Wszelkie niejasności specyfikacji powinni Państwo rozstrzygnąć obserwując jak radzi sobie nasza wtyczka oraz korzystając z powyższych plików testowych. Proszę zwrócić uwagę na priorytety i łączność operatorów!

Dla ciekawskich Parsery LR zostały wymyślone już w latach 60 [1]. Zaproponowany tam algorytm konstrukcji automatu LR(1) był jednak niepraktyczny z powodu rozmiaru wyniku. Jeszcze w tej samej dekadzie zaproponowano parsery LALR[2] będące pewnym kompromisem pomiędzy możliwościami automatu, a jego rozmiarem. Wiele generatorów parserów LR, w tym «`ocamlyacc`», bazuje do dnia dzisiejszego na tej technice.
Automaty LALR nie satysfakcjonują wielu użytkowników. Moc automatu albo bywa niewystarczająca albo praca z konfliktami w gramatyce bywa zbyt uciążliwa. Stąd w ostatnich latach można zaobserwowac pojawianie się generatorów parserów bazujących na pracach [3, 4], gdzie zaproponowano lepszą konstrukcję automatu LR(1). Jednym z takich narzędzi jest właśnie «`menhir`».
Wadą parsowania bottom-up, a w tym także LR, jest słaba obsługa błędów. Wiele kompilatorów z powodów pragmatycznych przyjmuje taką strategię, aby jak najwięcej błędów zgłosić użytkownikowi. Przy tym podejściu pożądanym zachowaniem jest pominięcie tego fragmentu pliku, który jest niepoprawny składniowo, a na całej reszcie uruchomić mimo wszystko mechanizm sprawdzania typów. Wiele generatorów parserów LR udostępnia mechanizm zwany *error recovery*, za pomocą którego użytkownik może dopisać dodatkowe reguły gramatyczne pomagające automatowi poradzić sobie w przypadku błędu składni. Użycie tego mechanizmu wykracza poza zakres przedmiotu, lecz zachęcamy Państwa do zapoznania się z nim.
Parsery z rodziny LR nie potrafią sobie radzić z językami, których gramatyki bezkontekstowe są niedeterministyczne, jak np C++. W ostatnich latach również widzimy odpowiedź na ten problem ze strony generatorów parserów. Niektóre z nich, jak «`bison`», posługują się algorytmem Generalised LR [5] potrafiący sobie poradzić z każdą gramatyką.

Literatura

[1] Knuth, D. On the translation of languages from left to right, Information and Control, Volume 8, Issue 6, 1965.

[2] DeRemer F. Pennello T. J. Efficient computation of LALR(1) look-ahead sets. SIGPLAN 1979

[3] Pager, D. A practical general method for constructing LR(k) parsers. Acta Informatica, Volume 7 Issue 3, 1977.

[4] Pager, D. The lane-tracing algorithm for constructing LR (k) parsers and ways of enhancing its efficiency, Information Sciences, Volume 12, Issue 1, 1977.

[5] Economopoulos, G. Generalised LR Parsing Algorithms. Phd 2006.