INSTYTUT PODSTAW INFORMATYKI POLSKIEJ AKADEMII NAUK

Marcin Woliński

Komputerowa weryfikacja gramatyki Świdzińskiego

Rozprawa doktorska
przygotowana pod kierunkiem dr. hab. Janusza S. Bienia, prof. UW

WARSZAWA 2004

Spis treści

Wstęp

Cel, założenia i zakres pracy

Niniejsza praca dotyczy automatycznej analizy składniowej wypowiedzeń w języku polskim. Celem automatycznej analizy składniowej jest sprawdzenie, czy wypowiedzenie (dostępne jako napis) jest akceptowane przez daną gramatykę formalną, i określenie jego struktury. W wypadku niejednoznaczności interpretacji, powinny zostać określone wszystkie struktury, mogące odpowiadać danemu wypowiedzeniu.

Gramatyką realizowaną komputerowo w tej pracy jest formalna gramatyka (podzbioru) języka polskiego autorstwa prof. Marka Świdzińskiego (Świdziński 1992). Mimo iż gramatyka ta zapisana jest w sposób bardzo formalny, możliwość jej realizacji komputerowej nie była celem jej Autora. W związku z tym znane z literatury techniki analizy składniowej nie dały się bezpośrednio zastosować. Opracowałem więc algorytm uwzględniający specyficzne potrzeby wynikające z charakteru gramatyki.

Celem pracy jest efektywna realizacja komputerowa pełnej gramatyki Świdzińskiego i jej weryfikacja. Zasadniczym elementem pracy jest napisany przeze mnie program komputerowy *Świgra*. Program ten, zrealizowany w języku Prolog, jest analizatorem składniowym działającym według reguł gramatyki Świdzińskiego.

Analizator składniowy nie pracuje na wypowiedzeniu w formie napisu, ale na jego wstępnie zinterpretowanej postaci — podzielonej na fragmenty, nazywane segmentami, które zostały zanalizowane morfologicznie (fleksyjnie). W programie *Świgra* to zadanie wypełnia analizator morfologiczny *Morfeusz*, także mojego autorstwa.

Istotny element zadania stanowiło opracowanie sposobu połączenia analizy morfologicznej ze składniową, ponieważ w książce Świdzińskiego reguły najniższego poziomu są z założenia niekompletne, w szczególności pewne jednostki nie są definiowane. Opis Świdzińskiego wymagał więc niejako domknięcia na poziomie morfologicznym. Punktem wyjścia były tu wyniki wcześniejszych prac przedstawionych w p. 1.3.

Dzięki zastosowaniu analizatora morfologicznego program *Świgra* pozwala operować szerokim zakresem słownictwa. W zestawieniu z bogatym repertuarem konstrukcji składniowych uwzględnionym w gramatyce Świdzińskiego otwiera to interesujące możliwości automatycznego przetwarzania tekstów polskich.

Uważam, że ze względu na dużą złożoność przedstawionej przez Świdzińskiego gramatyki, jej komputerowa realizacja w postaci możliwie bliskiej oryginału jest jedynym sposobem faktycznej weryfikacji opisu. Dlatego gramatykę Świdzińskiego traktuję, na ile to możliwe, jako zamknięte dzieło i staram się zrealizować ją wprowadzając jak najmniej zmian. Pewne modyfikacje były wszakże nie do uniknięcia. Na przykład niektóre z warunków, którymi opatrzone są reguły gramatyczne, wymagały przeformułowania, w celu zapewnienia ich poprawnego obliczania. Zapewnienie efektywności analizy wymagało też

wprowadzenia pewnych zmian w samych regułach gramatyki. Zmiany te były wprowadzane w taki sposób, aby nie naruszyć lingwistycznej warstwy opisu.

Oczywiście, jak każda gramatyka formalna, opis Świdzińskiego jest zarówno niepełny — odrzuca pewne konstrukcje poprawne, jak i nadmiarowy — dopuszcza pewne konstrukcje niepoprawne lub przypisuje konstrukcjom poprawnym nadmiarowe, nieadekwatne struktury. Analizę tych zagadnień można znaleźć w niniejszej pracy — w ten sposób rozumiem weryfikację gramatyki.

Wybór Prologu do realizacji gramatyki Świdzińskiego sam się narzuca: formalizm gramatyk metamorficznych, w którym jest ona zdefiniowana, został zaproponowany dla tego właśnie języka i w nim w najbardziej naturalny sposób wyraża się unifikacyjny charakter gramatyki.

Prologowe sformułowanie efektywnego algorytmu analizy składniowej w programie *Świgra* jest nowatorskie — w literaturze nie natrafiłem na tego rodzaju algorytm dla gramatyk metamorficznych.

Zawartość pracy

Pierwszy rozdział pracy stanowi nieformalne wprowadzenie do gramatyki Świdzińskiego. Ponieważ opisująca ją książka jest trudno dostępna, w punkcie 1.2 staram się przybliżyć tę gramatykę Czytelnikowi, który nie miał okazji się z nią zetknąć. Przedstawiam na przykładach, jaki z grubsza zakres wypowiedzeń obejmuje gramatyka i jakie przypisuje im interpretacje. Z konieczności prezentacja jest bardzo pobieżna. W punkcie 1.3 omawiam pokrótce historię zmagań z problemem realizacji komputerowej tej gramatyki.

Dalsze rozdziały można zgrupować w dwie części. Rozdział 2 i 3 odnosi się w ogólności do gramatyk zapisanych w podobnym formalizmie jak gramatyka Świdzińskiego (niekoniecznie gramatyk języka polskiego). Natomiast rozdziały 4–6 dotyczą polszczyzny i gramatyki Świdzińskiego.

W rozdziale 2 przedstawiam formalizm gramatyczny stosowany przez Świdzińskiego i jego wariant stosowany w programie *Świgra*. Rozdział 3 jest w większości poświęcony zastosowanemu algorytmowi analizy składniowej.

W rozdziale 4 omawiam analizę morfologiczną, która w przypadku języka polskiego stanowi konieczny krok wstępny przed analizą składniową. Przedstawiam też, w jakiej formie wyniki analizy morfologicznej stają się danymi dla analizatora składniowego. Rozdział 5 jest w całości poświęcony problemom, jakich nastręcza komputerowa realizacja gramatyki Świdzińskiego. Wyniki doświadczeń z programem *Świgra* zestawiłem w rozdziale 6.

Informacje zawarte w dodatku A powinny wystarczyć czytelnikowi do przeprowadzenia eksperymentów z programem. W dodatku B opisany jest szczegółowo format prezentacyjny drzew analizy. Dodatek C zawiera przykłady wyników pracy programu *Świgra*.

Konwencje notacyjne

Przykładowe wypowiedzenia, frazy, segmenty są składane kursywą. Kapitalikami oznaczono identyfikatory leksemów.

W przykładach gwiazdka * na początku sygnalizuje zdanie niepoprawne. Gwiazdka i wykrzyknik $*^{!}$ — zdanie niepoprawne akceptowane przez GFJP. Znak zapytania $^{?}$ — zdanie, którego poprawność budzi wątpliwości (autora niniejszego tekstu).

Fragmenty programu w języku Prolog (w szczególności reguły gramatyki) są przytaczane pismem bezszeryfowym, nazwy stałych prostym, a zmiennych — pochyłym, na przykład

f(a, X). Nazwy jednostek gramatycznych są wyróżniane przez pogrubienie, np. **wypowiedzenie**. Składnia reguł gramatycznych jest opisana w punkcie 2.1. Reguły gramatyczne Świdzińskiego, które zostały zmienione w programie są, w swojej oryginalnej postaci, wyróżniane szarym tłem. Ich wersja użyta w programie jest podawana bez takiego tła, podobnie jak reguły, które w książce i w programie są identyczne.

Podziękowania

Dziękuję mojemu promotorowi, profesorowi Januszowi S. Bieniowi za wytrwałość w zachęcaniu mnie do doprowadzenia tej pracy do końca. Podziękowania kieruję również do profesora Zygmunta Saloniego, któremu w znacznej mierze zawdzięczam zainteresowanie językoznawstwem uprawianym w sposób ścisły. Winien jestem wdzięczność profesorowi Markowi Świdzińskiemu za wiele wyjaśnień i inspirujących dyskusji o jego gramatyce. Chciałbym także serdecznie podziękować mojej rodzinie, przyjaciołom i współpracownikom za udzielone mi wsparcie, bez którego zapewne nie starczyłoby mi wytrwałości, by napisać tę pracę (osobom należącym do przecięć tych zbiorów dziękuję odpowiednio po wielokroć).

1. *Gramatyka formalna języka polskiego* Marka Świdzińskiego

Opis języka polskiego przedstawiony przez Marka Świdzińskiego wywodzi się z prac nieformalnego zespołu lingwistów i informatyków, uczestniczących w seminarium „Formalny opis języka naturalnego" prowadzonym przez Zygmunta Saloniego w latach od 1977 do 1990 w Instytucie Informatyki UW (por. Świdziński 1992, s. 52, Bień 1996).

Zespół ten rozwijał tematykę dystrybucyjnego opisu języka polskiego. Zwięzłe określenie istoty podejścia dystrybucyjnego znajdujemy u Szpakowicza (1983):

> Dystrybucja tworu składniowego jest to zbiór poprawnych językowo kontekstów, w których twór ten może wystąpić. Twory mające tę samą dystrybucję pozostają w relacji równoważności. Reguły składniowe odwołują się najczęściej nie do wyrazów, lecz do klas abstrakcji tej relacji.

Kierowanie się przy konstruowaniu opisu składniowego przede wszystkim dystrybucją opisywanych tworów wydaje się dobrym sposobem uniknięcia uwikłań semantycznych. Porównywanie kontekstów w jakich mogą wystąpić dane konstrukcje nie wymaga podawania od razu interpretacji tych kontekstów ani konstrukcji. Taka więc, bliska niezinterpretowanego tekstu, metoda badawcza pozwala na zbudowanie czysto składniowego opisu przy zachowaniu zadowalającego stopnia ścisłości.

Oparta na zasadach dystrybucyjnych klasyfikacja gramatyczna leksemów polskich jest tematem artykułu Saloniego (1974). Koncepcję bardziej rygorystycznego niż tradycyjny opisu faktów językowych przedstawił Saloni w swojej pracy habilitacyjnej (1976). Koncepcję formalnego opisu składni polskiej przedstawił Szpakowicz w rozprawie doktorskiej (1978, 1983). Praca ta zawiera gramatykę formalną obszernego podzbioru polszczyzny. Stanowi ona punkt wyjścia dalszych prac, przedstawionych w artykułach Szpakowicza i Świdzińskiego (1981b, 1982, 1986, 1990) oraz Świdzińskiego (1983, 1986).

Świdziński kontynuował samodzielnie ten nurt badań. Główny wynik stanowi jego praca habilitacyjna (1987) i jej wersja książkowa *Gramatyka formalna języka polskiego* (1992, dalej GFJP). Świdziński postanowił przedstawić w tym dziele kompletną gramatykę, obejmującą wszystkie poziomy konstrukcji składniowych. Ta właśnie gramatyka jest obiektem mojego zainteresowania, jej komputerową realizację stanowi program *Świgra*.

Punktem wyjścia rozważań Świdzińskiego jest krytyka szkolnego opisu składniowego (pod którą to nazwą rozumie on przede wszystkim ujęcie Z. Klemensiewicza). Autor podaje w wątpliwość podział zdań na pojedyncze i złożone wskazując, że podział taki jest oparty na bardzo niejasnych kryteriach semantycznych i trudnym do uzasadnienia przekonaniu o różnicy stopnia złożoności tych konstrukcji. Podział taki sprawia, że budowa zdań „pojedynczych" i „złożonych" jest opisywana zupełnie innymi pojęciami.

Jak pisze Świdziński, szkolny opis gramatyczny nie spełnia postulatu pełności i jawności: „Składnia tradycyjna nie dostarcza po prostu ani recept analizy, ani syntezy" (GFJP, s. 39).

Książka Świdzińskiego składa się z zasadniczej części opisowej i aneksu zawierającego reguły gramatyczne wyrażone w formalizmie zbliżonym do gramatyk metamorficznych (zob. p. 2.1). W praktyce te dwie części książki trzeba czytać równolegle. W aneksie można zobaczyć, jak zostały sformalizowane intencje wyrażone w części opisowej. Jednocześnie aneks bez części opisowej jest trudno zrozumiały właśnie ze względu na brak informacji o intencji takiego a nie innego sformułowania poszczególnych reguł.

1.1. Metodologiczne podstawy GFJP

Celem przedstawionego przez Świdzińskiego opisu składniowego jest „zdanie sprawy w sposób wyczerpujący i jawny ze wszystkich związków i zależności między składnikami konstrukcji językowych różnych poziomów, a więc z budowy, składu i dyspozycji zewnętrznych tych konstrukcji" (GFJP, s. 52).

Opis jest powierzchniowy, co oznacza, że nie uwzględnia ograniczeń semantycznych, a tylko składniowe. Można to zilustrować następującymi zdaniami:

(1) *Pies czyta gazetę.*
(2) **Szukam książkę do historii.*

Pierwsze z nich jest niemożliwe w naszym świecie: psy nie czytają. Jest to typowe ograniczenie semantyczne. Natomiast składniowo zdanie jest poprawne i możliwe są do pomyślenia „inne światy", w których opisana sytuacja się wydarzy[1]. Drugie ze zdań narusza ograniczenia składniowe: SZUKAĆ można *czego* a nie *co*. Tego zdania nie zaakceptujemy w polskojęzycznym opisie żadnego świata (mimo że wiemy, o co chodziło nadawcy tego komunikatu).

Przedstawione rozróżnienie wydaje się oczywiste. Jednak „test innych światów" bywa użyteczny. Na przykład niektórzy użytkownicy języka są skłonni odrzucić zdanie

(3) *Dobrze spałom.*

Twierdzą oni, że niemożliwa jest nijaka forma pierwszej (i drugiej) osoby czasownika. Jednak w świecie baśni łatwo sobie wyobrazić mówiące słoneczko lub źródło. Wypada więc uznać, że nie ma fleksyjnych ani składniowych podstaw do odrzucenia tego zdania.

Metodą opisu w GFJP jest analiza na składniki bezpośrednie. Polega ona na hierarchicznej dekompozycji badanego tworu na ciąg tworów prostszych. W wyniku tego procesu tworowi językowemu przypisuje się reprezentację (lub reprezentacje) w postaci drzew składników bezpośrednich. Każdy wierzchołek odpowiada pewnemu spójnemu fragmentowi badanego tworu interpretowanemu jako pewna jednostka składniowa, a jego wierzchołki potomne odpowiadają bezpośrednim składnikom tego tworu (por. Saloni i Świdziński 2001, rozdział III).

Zbiór jednostek składniowych GFJP tworzy rozbudowaną hierarchię. Na przykład jednostki zdaniowe obejmują zdanie równorzędne, szeregowe, jednorodne, proste i elementarne.

[1] W jednym z filmików z kolekcji *Wściekłe gacie* pies Gromit czyta w gazecie artykuł pod tytułem *Pies czyta gazetę!*.

Jednostkom składniowym przypisywane są komplety charakteryzujących je cech, nazywanych przez Świdzińskiego parametrami jednostek.

Za podstawowe zadanie opisu gramatycznego Autor uważa zdanie sprawy z ograniczeń łączliwości wewnątrz zdania. Jako źródło tych ograniczeń wyróżnia implikację syntaktyczną i związki składniowe, czyli uzgodnienia (GFJP, s. 23).

Przez implikację syntaktyczną (konotację w sensie pracy Saloniego i Świdzińskiego 2001) należy rozumieć zapowiadanie przez jedną jednostkę składniową pojawienia się innej jednostki. Taka zapowiedź oznacza, że zapowiadana jednostka musi pojawić się w wypowiedzeniu, aby było ono pełne (nieeliptyczne).

Uzgodnienie pewnej cechy składniowej to relacja między dwiema jednostkami składniowymi, spełniona wtedy, gdy wartości tej cechy dla obu jednostek są równe.

Świdziński uwzględnia „wszystkie — w miarę możliwości — istotne syntaktycznie cechy fleksyjne tworzących definiowane konstrukcje jednostek składniowych, a także rozmaite cechy czysto składniowe" (GFJP, s. 21).

1.2. GFJP z lotu ptaka

Opis przedstawiony przez Świdzińskiego ma formę gramatyki metamorficznej (por. rozdział 2). Gramatyka ta składa się z 461 reguł. Przedmiotem opisu jest pojedyncze wypowiedzenie (w dużym uproszczeniu — od wielkiej litery do kropki), które poddawane jest analizie na składniki bezpośrednie.

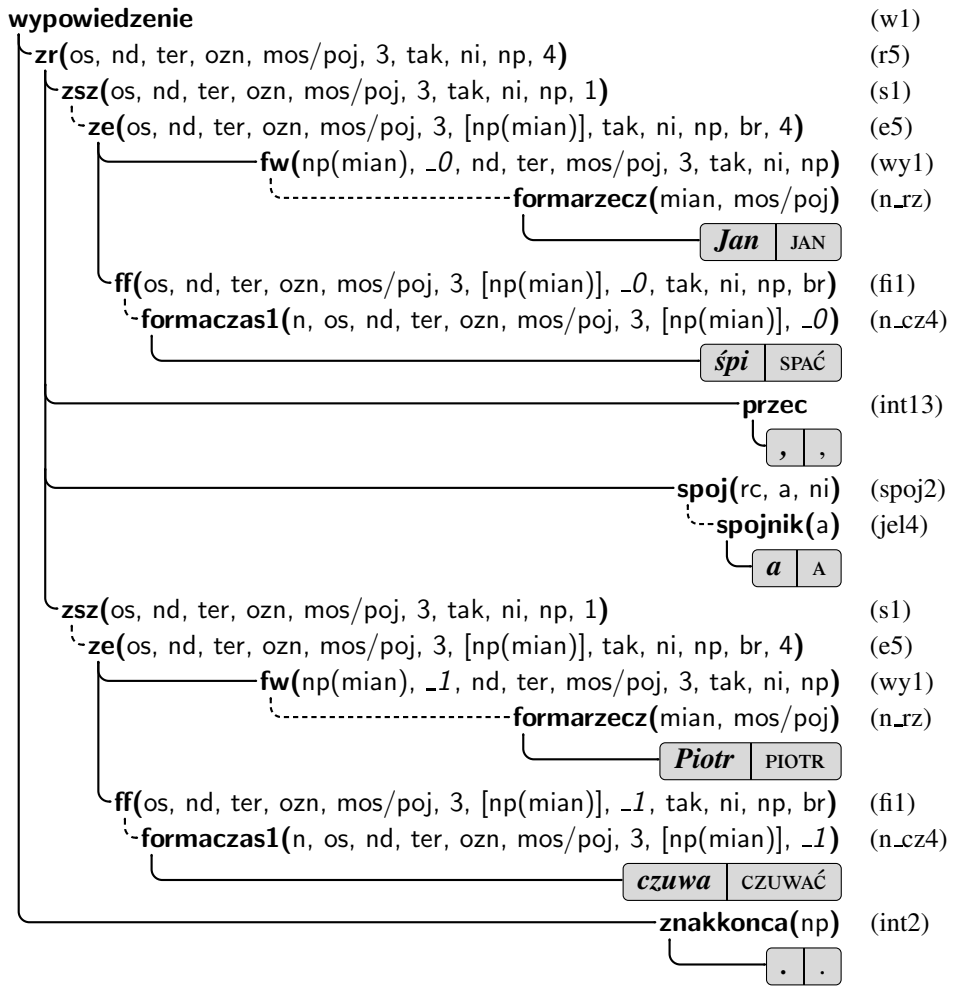
Opisywane są wyłącznie wypowiedzenia mające postać zdań, a więc konstrukcje zawierające frazę finitywną. Fraza finitywna według Świdzińskiego to „człon składniowy, którego centrum stanowi forma osobowa lub bezosobnik czasownika, forma finitywna czasownika niewłaściwego, forma bezokolicznikowa czasownika lub czasownika niewłaściwego, a także równoważnik dystrybucyjny wymienionych form" (GFJP s. 21). Przykłady form czasownikowych poszczególnych typów można znaleźć w punkcie 4.7.5.

Obiektem zainteresowania Świdzińskiego są przede wszystkim „własności składniowe konstrukcji nazywanych przez składnię tradycyjną wypowiedzeniami złożonymi". Autor rozważa bogaty repertuar zdań, uwzględnia przy tym cechy składniowe nie dostrzegane przez składnię tradycyjną (np. uzgodnienie aspektu, ograniczenie czasu lub trybu).

W opisie Świdzińskiego każde **wypowiedzenie** składa się ze zdania równorzędnego — jednostki **zr** i znaku końca — **znakkonca**. Na rysunku 1.1 przedstawiono drzewo analizy na składniki bezpośrednie[2], w którym zdanie równorzędne jest realizowane przez dwa zdania szeregowe **zsz** połączone przecinkiem i spójnikiem a.

Rysunek ten jest zapisem skróconym pokazującym tylko istotne poziomy hierarchii. Przykłady pełnych drzew analizy można znaleźć w dodatku C (pełne drzewo dla tego przykładu — rys. C.1 na s. 130). Znaczenie poszczególnych elementów drzewa objaśniam szczegółowo w dodatku B. Na potrzeby bieżącego rozdziału wystarczy zauważyć, że w wierzchołkach drzewa umieszczono nazwy jednostek składniowych (**wypowiedzenie**, **zr**, **zsz**, itd.) wraz z kompletem przysługujących im cech składniowych (np. os oznacza formę osobową, nd — aspekt niedokonany, ter — czas teraźniejszy, itd.; pełna lista wartości w punkcie B.3). Symbole w nawiasach po prawej stronie rysunku są nazwami reguł gramatycznych według książki Świdzińskiego.

[2] Wszystkie drzewa analizy prezentowane w tej pracy są wynikami programu *Świgra*.



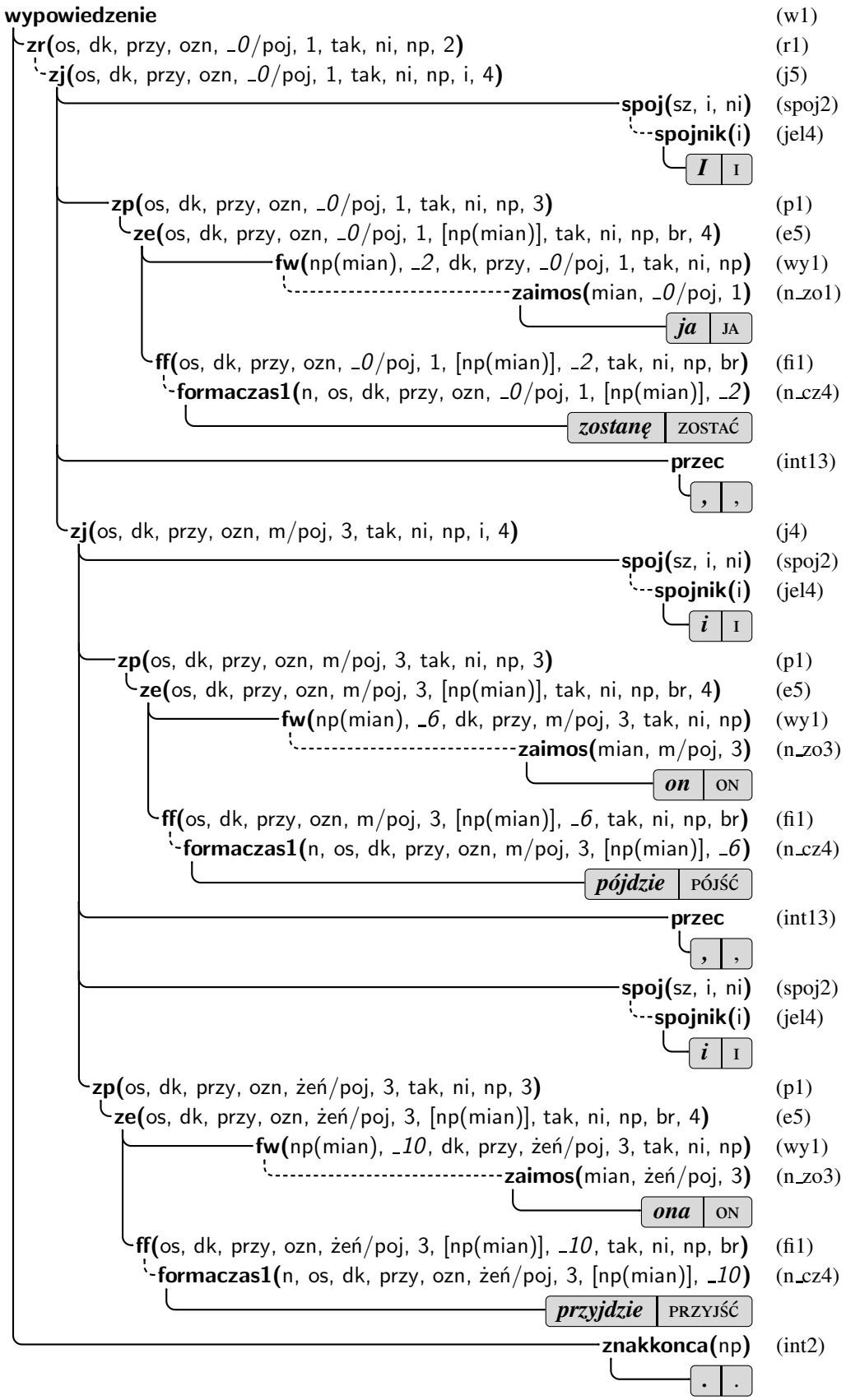
Rys. 1.1. Skrócone drzewo analizy zdania *Jan śpi, a Piotr czuwa.*

Świdziński wyróżnia dwa sposoby zespalania jednostek składniowych: współrzędny i podrzędny (por. GFJP, s. 72). Przy zespoleniu współrzędnym składniki o zbliżonych własnościach gramatycznych są połączone elementem funkcyjnym — spójnikiem równorzędnym lub szeregowym. Zespolenie podrzędne polega na dołączeniu do danego składnika drugiego, ściśle określonego typu, być może o innych własnościach gramatycznych.

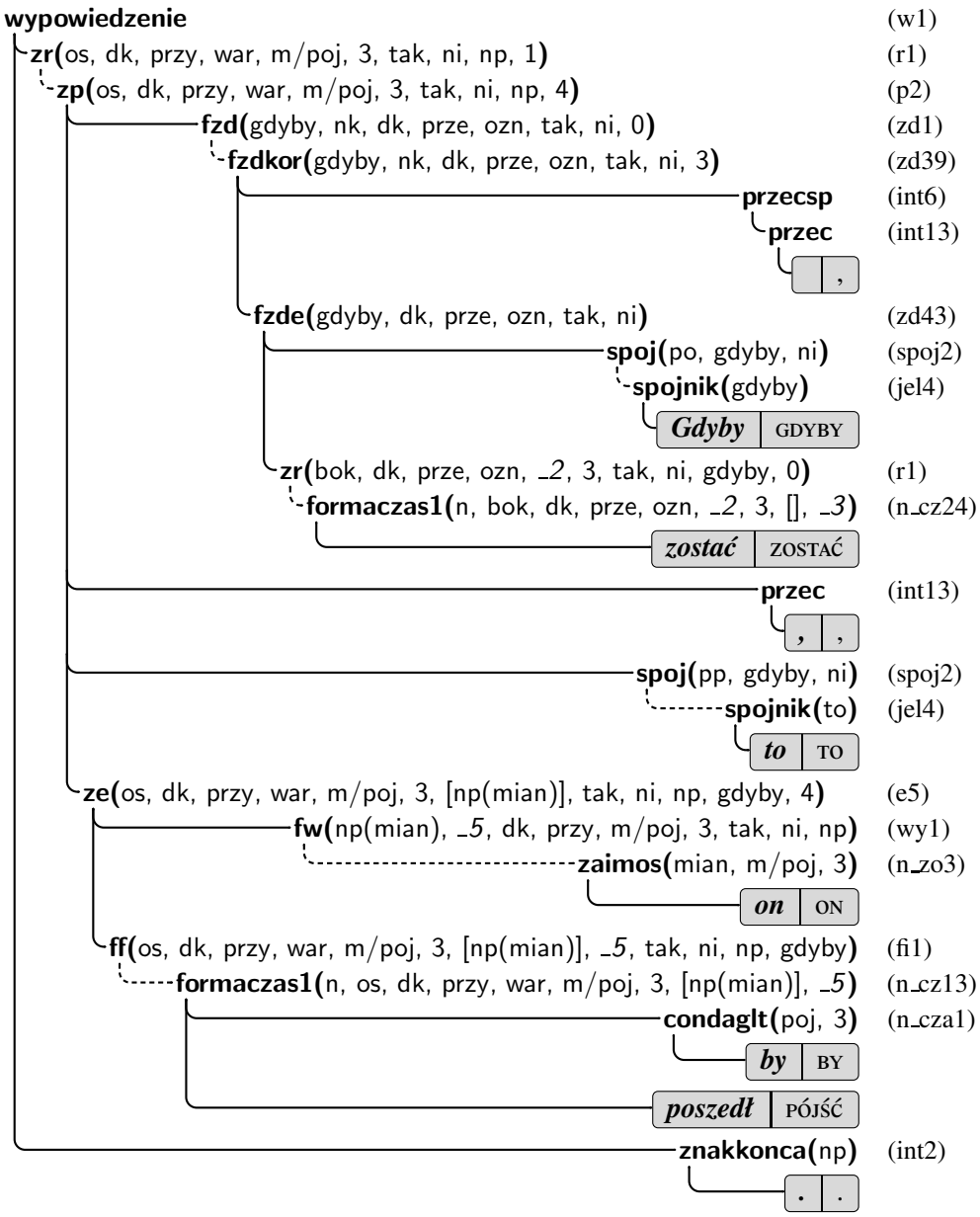
W zdaniu przedstawionym na rys. 1.1 spójnik równorzędny centralny A spaja współrzędnie dwa zdania szeregowe. Innym przykładem spójnika równorzędnego jest spójnik równorzędny nieciągły (np. ZARÓWNO ... JAK TEŻ ...) i spójnik równorzędny inkorporacyjny (np. NATOMIAST). Przykładowe drzewa zawierające te spójniki można znaleźć na rysunkach C.2 i C.3 (s. 131–132).

Przykład konstrukcji zespolonej współrzędnie spójnikami szeregowymi przedstawiono na rysunku 1.2. W drzewie tym główną rolę odgrywa jednostka **zj** — zdanie jednorodne i **zp** — zdanie proste.

Przykład konstrukcji zespolonej podrzędnie można znaleźć na rysunku 1.3. W przykładzie zdanie proste składa się z frazy zdaniowej **fzd**, przecinka, spójnika podrzędnego i zdania elementarnego **ze** (jest to typowy zestaw składników zdania prostego).

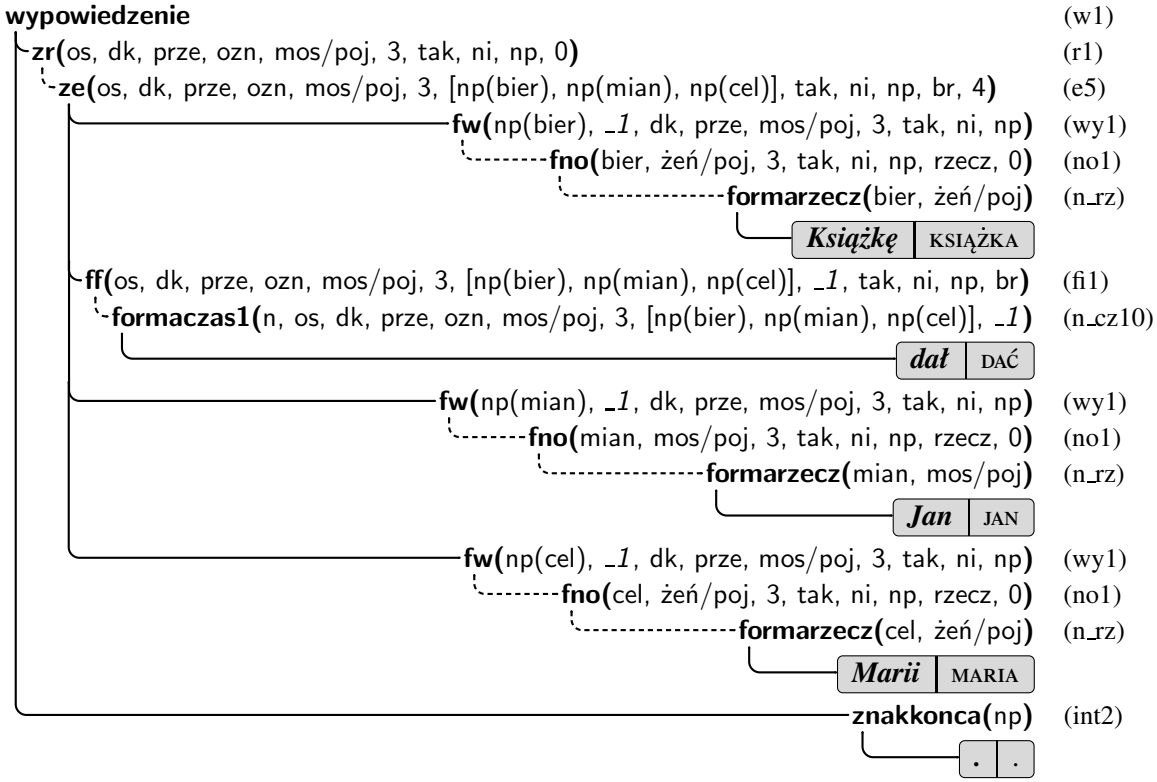


Rys. 1.2. Skrócone drzewo analizy zdania *I ja zostanę, i on pójdzie, i ona przyjdzie.*

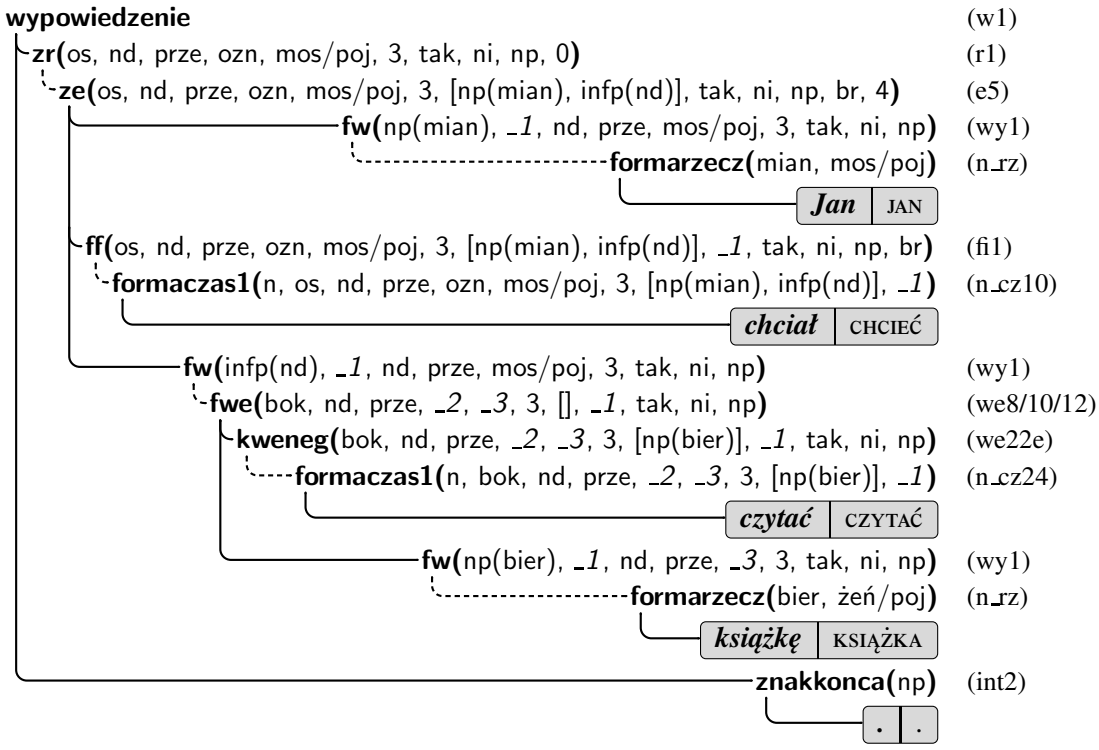


Rys. 1.3. Skrócone drzewo analizy zdania *Gdyby zostać, to on by poszedł.*

Najniższym poziomem hierarchii zdań jest zdanie elementarne **ze**. Typowo zdanie takie składa się z frazy finitywnej **ff** i ciągu fraz wymaganych **fw** w dowolnym porządku. Frazy wymagane mogą być realizowane przez frazy kilku typów, w zależności od wymagań formy czasownikowej stanowiącej tzw. centrum zdania elementarnego. Na przykład na rysunku 1.4 składnikami zdania elementarnego są frazy wymagane realizowane przez frazy nominalne **fno** w bierniku, mianowniku i celowniku. Jak widać, podmiot zdania nie jest wyróżniony jako osobna jednostka. Jest on jedną z fraz wymaganych, znamienną tym, że jest realizowana przez frazę nominalną w mianowniku. W jej wnętrzu zachodzą inne uzgodnienia, niż w pozostałych frazach wymaganych. W przykładzie na rysunku fraza *Jan* uzgadnia rodzaj-liczbę z frazą finitywną (wartość mos/poj w obu frazach), w przeciwieństwie do frazy nominalnej *Marii* (o oznaczeniu rodzaju-liczby żeń/poj).

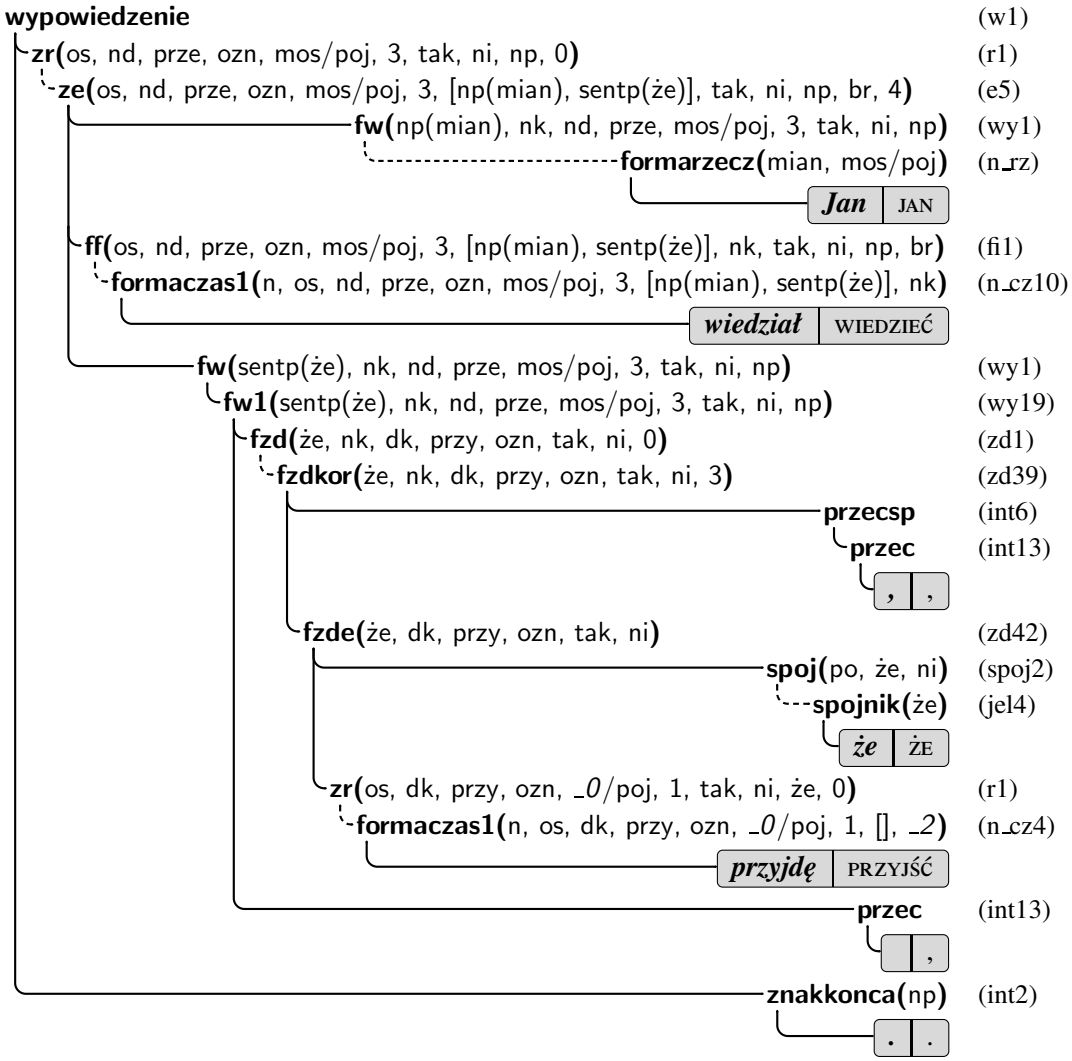


Rys. 1.4. Skrócone drzewo analizy zdania *Książkę dał Jan Marii.*



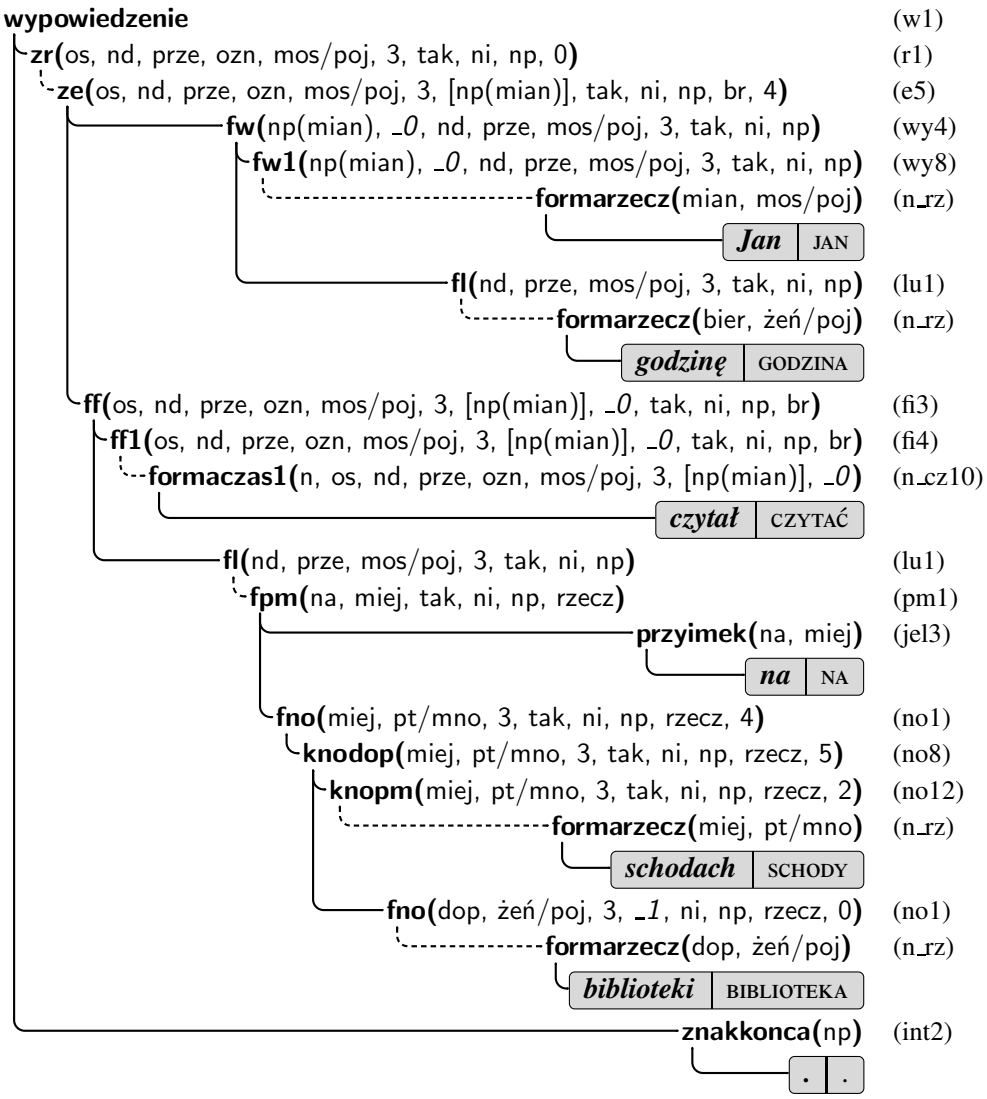
Rys. 1.5. Skrócone drzewo analizy zdania *Jan chciał czytać książkę.*

Na rysunku 1.5 przedstawiono przykład zdania, w którym fraza wymagana jest realizowana przez bezokolicznikową frazę werbalną **fwe**, a na rys. 1.6 — przez frazę zdaniową **fzd** typu że (o pustej realizacji przecinka występującej w tym drzewie piszę w p. 5.6). Pozostałe typy fraz, które mogą stanowić realizację fraz wymaganych, to fraza przyimkowa, fraza przymiotnikowa i fraza przysłówkowa. (Rozwinięcia nazw wszystkich jednostek nieterminalnych można znaleźć w punkcie B.1).



Rys. 1.6. Skrócone drzewo analizy zdania *Jan wiedział, że przyjdę.*

W strukturze zdania elementarnego może również wystąpić fraza luźna **fl**, czyli człon, który nie jest wymagany przez centrum finitywne. Przykłady takich fraz zawiera rysunek 1.7. Frazy luźne (z wyjątkiem inicjalnej) „przyczepiają" się w strukturze zdania elementarnego do poprzedzającej frazy finitywnej lub wymaganej (nie znaczy to jednak, że dana fraza luźna w jakikolwiek sposób przynależy do frazy, do której się przyczepiła — należy raczej traktować te frazy jako składniki zdania elementarnego; Świdziński zastosował taki opis, aby dopuścić obecność fraz luźnych w dowolnym miejscu zdania elementarnego).



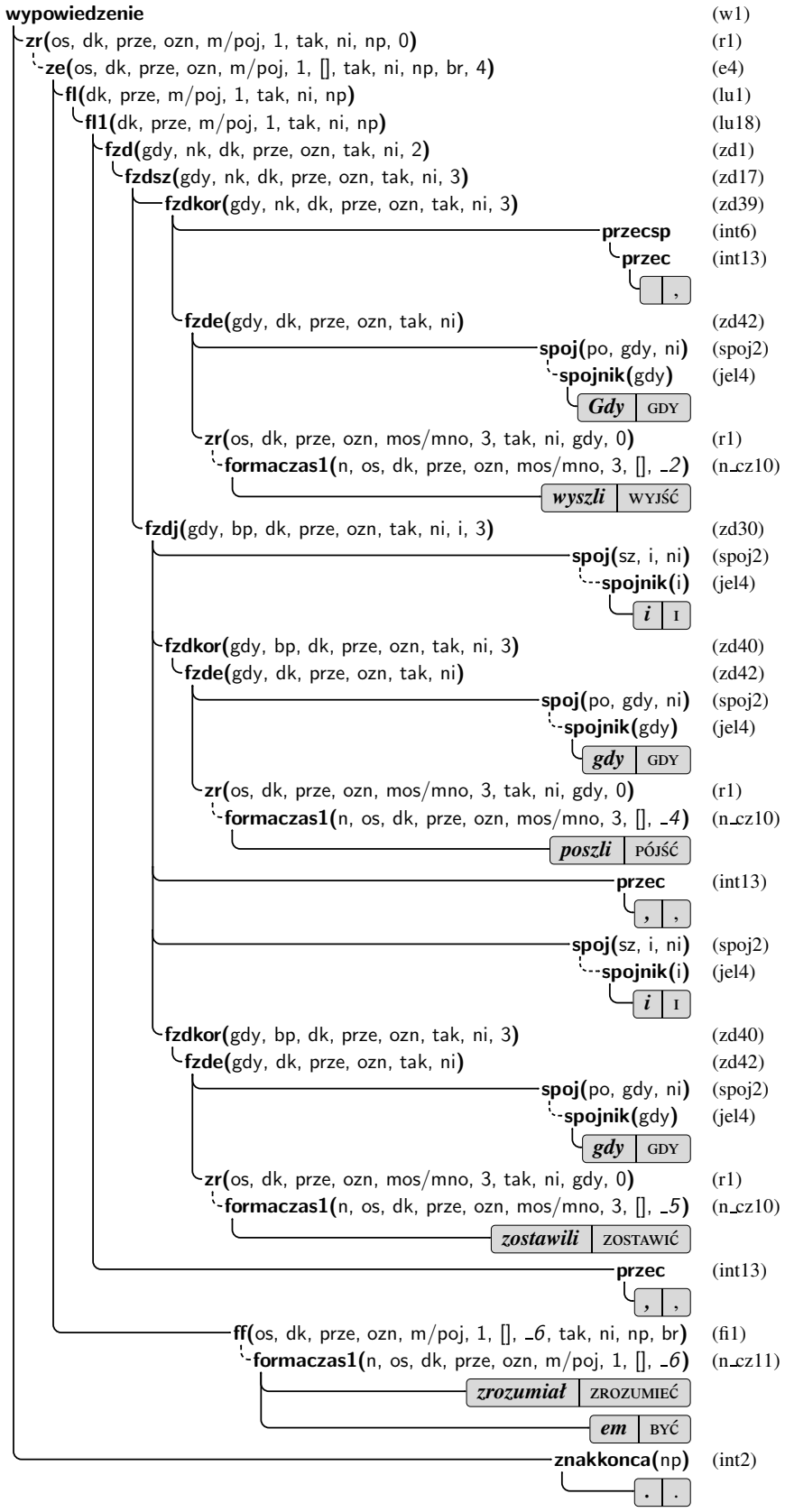
Rys. 1.7. Skrócone drzewo analizy zdania *Jan godzinę czytał na schodach biblioteki.*

Opis fraz jest w GFJP mniej dokładny i mniej rozbudowany niż opis jednostek zdaniowych (GFJP, s. 77):

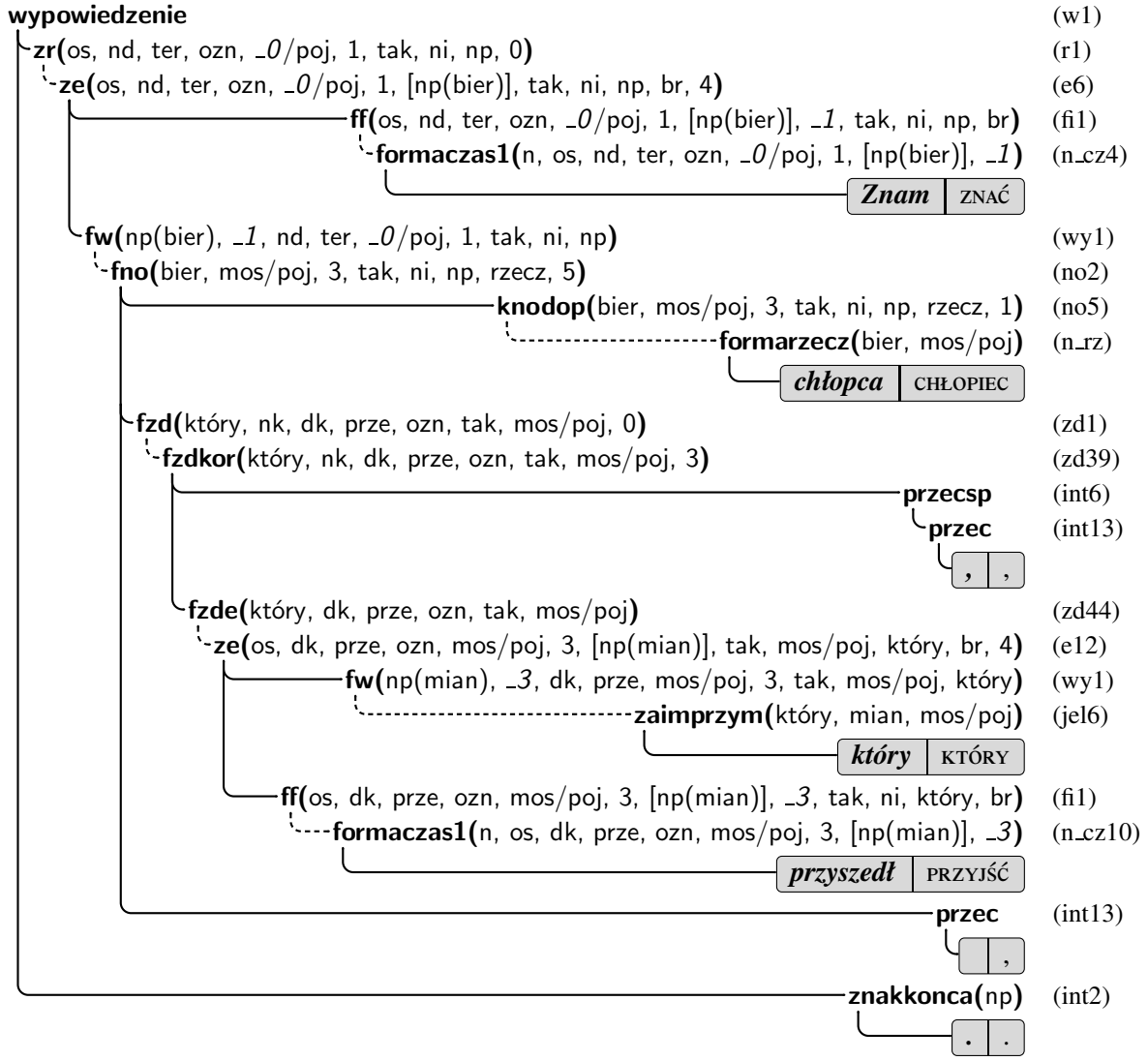
> Każda fraza ma również strukturę hierarchiczną. Lokalne hierarchie fraz nie są jednak tak istotne z punktu widzenia celów tej pracy, jak hierarchie jednostek zdaniowych: tylko jeden typ frazy, mianowicie fraza zdaniowa, zostanie tu opisany w sposób maksymalnie szczegółowy. (...) Dla uproszczenia wszystkie frazy z wyjątkiem frazy zdaniowej będą uważane za frazy elementarne, a więc niewspółrzędne.

Tak więc uwzględnione są współrzędnie złożone frazy zdaniowe (rys. 1.8) oraz obecność fraz zdaniowych np. w strukturze frazy nominalnej (rys. 1.9), ale nie np. współrzędnie złożone frazy nominalne jak *Jan i Piotr*.

Znaki interpunkcyjne są traktowane jako pełnoprawne składniki wypowiedzenia, w związku z czym ich brak powoduje, że wypowiedzenie staje się nieakceptowane przez gra-



Rys. 1.8. Skrócone drzewo analizy zdania *Gdy wyszli i gdy poszli, i gdy zostawili, zrozumiałem.*



Rys. 1.9. Skrócone drzewo analizy zdania *Znam chłopca, który przyszedł.*

matykę. Autor uwzględnia w opisie „kropkę, wielokropek przed wielką literą, wykrzyknik, znak zapytania i przecinek" (GFJP, s. 28).

W GFJP niedospecyfikowany jest słownik stowarzyszony z gramatyką i reguły opisujące najmniej skomplikowane twory — realizację poszczególnych konstrukcji przez pojedyncze słowa. Ta część opisu wymagała uzupełnienia przy realizacji komputerowej (por. p. 4.4 i 4.7).

Nie są objęte opisem zdania z frazami nieciągłymi (por. GFJP, p. 4.2.4, s. 66), mową niezależną, i kilkoma specyficznymi typami konstrukcji zdaniowych (por. GFJP, p. 1.6.5, s. 29).

Przy analizie zjawisk lingwistycznych wielokrotnie zdarza się, że konstrukcji można nadać wiele różnych struktur. Świdziński jako lingwista w takich sytuacjach opisuje wszystkie możliwe warianty interpretacyjne. Dotyczy to w szczególności zdań z przykładów w tym punkcie. Niektóre z nich mają wiele interpretacji, na rysunkach pokazałem drzewa najlepiej ilustrujące dane zjawiska.

1.3. Prace nad komputerową realizacją GFJP

Świdziński zastrzega się, że jego praca nie jest pisana z myślą o realizacji komputerowej. Jeśli jednak wziąć pod uwagę wysoki poziom szczegółowości reguł zawartych w aneksie książki, pomysł realizacji komputerowej nasuwa się w sposób naturalny. Janusz S. Bień podjął pierwszą próbę realizacji GFJP w latach 1994–1996 w ramach projektu KBN nr 8 S503 032 07 *Analizator morfologiczno-syntaktyczny dla obszernego podzbioru języka polskiego*, uzasadniając to w następujący sposób (Bień 1996, s. 2):

> Świdziński pisze wprawdzie (Świdziński 1992, s. 58):
>
> > Opis przedstawiony w niniejszej pracy ukierunkowany jest w większym stopniu lingwistycznie (empirycznie) niż informatycznie. Przyjmuję tutaj tak wysoki stopień szczegółowości empirycznej, że bezpośrednia implementacja nawet fragmentów podanej w tej pracy gramatyki nie wydaje się możliwa. Nie sposób jednak nie zgodzić się z twierdzeniem, że łatwiej zredukować opis bardziej rozbudowany niż rozbudować zbyt ubogi.
>
> ze stanowiskiem tym jednak nigdy się nie zgadzałem. Co więcej, wyznawałem pogląd całkowicie przeciwny. Uważałem mianowicie, że do komputera należy wprowadzić opis najlepiej bez żadnych zmian i dopiero po przeprowadzeniu odpowiednich eksperymentów dokonać w razie potrzeby modyfikacji i rozszerzeń tego opisu. Ze względu na olbrzymią złożoność sformułowanej przez Świdzińskiego formalnej gramatyki języka polskiego, jej analiza „na piechotę" (czyli bez użycia komputera) wymaga tak silnej motywacji i determinacji, że trudno jej oczekiwać od typowego lingwisty, nawet jeśli specjalizuje się on w badaniach składniowych.

Myślę, że pogląd ten można jeszcze rozwinąć: o ile czytelnik zaznajomiony z formalizmem stosowanym przez Świdzińskiego jest w stanie osiągnąć taki poziom zrozumienia, żeby przewidywać, jak „powinny" być interpretowane w zgodzie z GFJP dane konstrukcje, to stwierdzenie, czy faktycznie wszystkie warunki występujące w regułach są spełnione, jest praktycznie niemożliwe. Warunki są bowiem rozsiane po wielu regułach gramatyki i czasami pewne wartości są transmitowane przez wiele poziomów hierarchii zanim w końcu okażą się istotne w jakimś warunku[3]. Można więc powiedzieć, że GFJP, mimo że nie tworzona z myślą o maszynie, może być dogłębnie zrozumiana przez człowieka tylko z pomocą maszyny.

Marek Świdziński udostępnił swoją gramatykę w postaci elektronicznej. Gramatyka w książce zapisana jest w formie zbliżonej do składni marsylskiej Prologu. Pierwszym technicznym krokiem było wydobycie tekstu z pliku w formacie edytora ChiWriter i konwersja reguł na składnię edynburską używaną przez obecnie stosowane interpretery Prologu.

Komputerowa realizacja gramatyki języka polskiego jest na dobrą sprawę niemożliwa bez sprawnego modułu analizy morfologicznej (por. rozdz. 4). Drogę do realizacji GFJP otworzyło pojawienie się analizatora morfologicznego SAM. Program ten opracował w ramach swojej pracy doktorskiej Krzysztof Szafran (1993). Pierwotnie program ten miał służyć do interakcyjnego hasłowania tekstu (Szafran 1997). Na potrzeby realizacji GFJP Szafran przygotował zmodyfikowaną wersję — SAM-95 (Szafran 1996).

[3] Niektóre wartości parametrów okazały się być zaskakujące nawet dla Autora gramatyki. Przykładem jest problem zdania rozważanego na początku punktu 6.4

W ramach projektu KBN Janusz Bień zaprojektował system analizy morfologiczno-syntaktycznej AMOS i częściowo go zaimplementował, w szczególności część związaną z właściwą analizą składniową, interpretacją warunków, strategią analizy itp. Moduły pomocnicze realizowałem ja; był to notabene mój pierwszy kontakt z lingwistyką informatyczną.

Istotnym modułem był program przekształcający wyniki programu SAM, które są zapisywane bardzo skrótowo, na postać zgodną z notacją Świdzińskiego. Janusz Bień zaprojektował sposób reprezentacji w systemie AMOS zinterpretowanego morfologicznie wypowiedzenia, bazujący na koncepcjach swojej pracy (1991), a ja zaprogramowałem odpowiednie przekształcenie w języku Perl i Prolog. Ze względu na stopień komplikacji program ten był uruchamiany osobno, a wyniki analizy morfologicznej były zapisywane do pliku tymczasowego, który dopiero był czytany przez analizator składniowy.

Opracowano także prowizoryczny słownik wymagań czasownikowych w formacie odpowiadającym notacji GFJP. Słownik ten tworzono na podstawie intuicji i w miarę potrzeb tak, aby zapewnić akceptowanie przez program opracowywanych właśnie przykładów.

Ponieważ formalizm stosowany przez Świdzińskiego jest w zasadzie gramatyką metamorficzną, której realizacja jest dostępna w Prologu, naturalnym wydawało się zastosowanie dostępnego w Prologu mechanizmu analizy o strategii zstępującej.

W pierwszych eksperymentach stosowany był opracowany w Instytucie Informatyki Uniwersytetu Warszawskiego interpreter MIM-Prolog. Wkrótce jednak okazał się on niewystarczający, w czym można widzieć potwierdzenie pesymistycznej opinii Świdzińskiego przytoczonej wcześniej. MIM-Prolog był programem działającym w systemie DOS z charakterystycznymi dla tegoż systemu ograniczeniami przestrzeni adresowej. Wkrótce zaczęliśmy więc stosować komercyjną implementację SICStus Prolog, która jest pozbawiona takich ograniczeń. Na tym etapie prac udało się poprawić najbardziej oczywiste błędy techniczne w regułach i doprowadzić do tego, aby przetwarzać proste zdania analizatorem zawierającym odpowiedni podzbiór reguł Świdzińskiego.

Również na tym etapie opracowałem sposób wizualizacji drzew analizy w formie tekstowej i za pomocą systemu TEX. Ten drugi sposób pozwala zapisać drzewa dla serii wypowiedzeń w formie dokumentu PDF z zakładkami pozwalającymi łatwo przejść do obrazów drzew dla danego wypowiedzenia.

Gramatyka Świdzińskiego kryje w sobie wiele pułapek dla programisty. Zawiera reguły lewostronnie rekurencyjne (np. cykl reguł o symbolach (wy1), (wy6), (we3)), dlatego też Bień dodawał w takich wypadkach nowe warunki do reguł, aby zablokować zapętlenia (por. Bień 1996).

Problematyczne było sprawdzanie warunków obecnych w regułach gramatyki. Świdziński pisał warunki jako statyczne ograniczenia, którym muszą podlegać wartości parametrów, nie przejmując się kolejnością obliczania tych wartości. Przy analizie zstępującej prowadzonej od lewej do prawej nie wszystkie wartości są znane w miejscu obliczania warunku. Przykład może stanowić wartość parametru zależności dla zdania oznajmującego, która wyjaśnia się dopiero po przeczytaniu znaku końca wypowiedzenia (kropki), podczas gdy warunki na tę wartość są badane w trakcie analizy zdania. Bień postanowił trudność tę pokonać za pomocą mechanizmu korutyn dostępnego w SICStus Prologu. Mechanizm ten pozwala zawieszać sprawdzenie warunków do momentu ukonkretnienia wskazanych zmiennych (predykat freeze). Niestety sposób ten prowadził do pojawiania się trudnych do wyśledzenia zakleszczeń „zamrożonych" warunków.

W ramach projektu opracowano także podstawowe dane testowe — zdania przykładowe z książki Świdzińskiego. Pierwsza ich część to 660 przykładów pochodzących z aneksu zawierającego reguły. Każde z tych wypowiedzeń jest oznaczone jako poprawne lub niepoprawne i opatrzone symbolem reguły, którą ilustruje. Co więcej w wypowiedzeniach oznaczone są fragmenty, które powinny zostać zanalizowane według wskazanej reguły. Druga część danych testowych to 1377 przykładów z części opisowej książki. Tym zdaniom nie były przypisane numery reguł, część informacji została więc dopisana na podstawie tekstu komentującego przykłady.

Wynikiem projektu jest analizator morfologiczno-syntaktyczny AMOS-95 (zob. Bień 1996, 1997). Ze względu na zbyt małą efektywność analizy, w ramach tego projektu udało się przetworzyć tylko kilkuwyrazowe zdania przykładowe.

Prace nad realizacją GFJP były kontynuowane na marginesie projektu KBN nr 8 T11C 002 13 *Zestaw testów do weryfikacji i oceny analizatorów języka polskiego*, trwającego w latach 1997–1999 również pod kierownictwem Janusza S. Bienia. Główny nurt tego projektu był inny, ale moim w nim zadaniem było opracowanie koncepcji i realizacja nowego analizatora składniowego opartego na GFJP.

Założenia projektowe nowego analizatora składniowego (który został nazwany *AS*) były istotnie inne niż w pierwszej próbie. Przede wszystkim zastosowano wstępujący porządek analizy składniowej (od najprostszych fraz do bardziej skomplikowanych), co okazało się być lepszą strategią dla języka polskiego. Drugim istotnym ulepszeniem analizatora było wprowadzenie mechanizmu zapamiętywania wyników pośrednich (por. Matsumoto *et al.* 1983, 1985). Za tymi zmianami poszła zmiana sposobu reprezentacji parametru zależności. W tej wersji zamiast pojedynczej wartości jednostce składniowej przypisywana była lista dopuszczalnych wartości zależności. Ponieważ jednak za istotny uznajemy walor dydaktyczny uzyskanych drzew, wyniki analizy były poddawane takiemu przekształceniu, aby wyglądały na otrzymane za pomocą oryginalnej gramatyki Świdzińskiego.

Efektywność analizy programu *AS* można określić jako obiecującą — przeciętnie 1 sekunda na wygenerowanie 1 drzewa analizy (mierzone na ówczesnych komputerach, czyli Pentium 200MHz).

Ta wersja programu została zrealizowana za pomocą interpretera SWI-Prolog, który stanowi oprogramowanie swobodne, dostępne na licencji GNU. Dzięki podobieństwu tego dialektu języka do SICStus Prologu program nadal działa z tym ostatnim praktycznie bez zmian.

Na zakończenie projektu (por. Bień 2000 i Bień *et al.* 2000) dysponowaliśmy programem, który dla większości zdań przykładowych dawał wyniki w sensownym czasie. Niestety w wypadku pewnych zdań czas analizy nadal wydawał się nie do oszacowania. Ponadto źródłem niewygody pozostawała analiza morfologiczna, przeprowadzana z użyciem osobnego programu. Czynniki te sprawiały, że interakcyjne eksperymenty z programem były bardzo utrudnione.

Ze względu na niedoskonałość analizatora morfologicznego SAM trzeba było utrzymywać osobny słownik rozmaitych wyjątków i poprawek. Widać było, że w istocie trzeba zacząć w sposób zorganizowany ulepszać słownik morfologiczny. Te czynniki oraz dostępność danych drugiego poprawionego wydania *Indeksu Tokarskiego* (Tokarski 2002) sprawiły, że zdecydowałem się napisać nowy program analizy morfologicznej *Morfeusz*. Słownik programu ma jeszcze widoczne niedoskonałości, podlega jednak kolejnym ulepszeniom. Ponadto program ma postać komponentu dającego się sprawnie wbudować w różne systemy.

I rzeczywiście, już obecna wersja jest wykorzystywana w programie znakującym korpus IPI PAN (gdzie współpracuje z modułem autorstwa Ł. Dębowskiego kontekstowo wybierającym interpretacje morfologiczne), w systemie ekstrakcji informacji SProUT (J. Piskorski), i w gramatyce A. Przepiórkowskiego realizowanej w systemie TRALE.

Tematem niniejszej pracy jest opracowana przeze mnie kolejna wersja analizatora składniowego bazującego na GFJP nazwana *Świgra*. Współdziała ona z analizatorem morfologicznym *Morfeusz*, daje możliwość prowadzenia interakcyjnych eksperymentów z gramatyką (na przykład analizy fragmentu wypowiedzenia jako wskazanej jednostki nieterminalnej i natychmiastowego wyświetlenia wynikowych drzew) oraz jest znacząco poprawiona pod względem wydajności analizy.

2. Gramatyki metamorficzne

Świdziński prezentuje swoją gramatykę w formie zbliżonej do oryginalnej notacji gramatyk metamorficznych zaproponowanej przez Colmerauera (1978). Notacja ta bazuje na tzw. składni marsylskiej Prologu, obecnie praktycznie zupełnie wypartej przez tzw. składnię edynburską. W tym wariancie formalizm nosi nazwę *Definite Clause Grammars* (DCG; por. Pereira i Warren (1980); o ile mi wiadomo, nie utarła się polska nazwa). Ta właśnie notacja jest stosowana w programie *Świgra* i w dalszych rozdziałach niniejszej pracy.

Z praktycznego punktu widzenia gramatyki metamorficzne i DCG można utożsamić (por. końcową część p. 2.1). Tak też postąpię w tej pracy — z wyjątkiem najbliższych dwóch punktów będę posługiwał się określeniem gramatyki metamorficzne na oznaczenie obu tych formalizmów. Aby uniknąć niepotrzebnego zaciemniania reguł różnymi wariantami notacji, reguły Świdzińskiego będę cytował skonwertowane do notacji DCG.

Rozpocznę od nieformalnego przedstawienia notacji gramatycznej Definite Clause Grammars. W punkcie 2.2 pokażę, czym różni się ta notacja od stosowanej przez Świdzińskiego. W następnych punktach opisuję sposób interpretacji gramatyk w Prologu i podaję sposób jej rozszerzenia tak, aby analiza odbywała się na ścieżkach acyklicznego grafu, zamiast na ciągach. W końcu podaję formalną definicję tak rozszerzonych gramatyk metamorficznych.

2.1. Notacja

Jak wspomniałem w punkcie 1.1, techniką stosowaną przez Świdzińskiego jest analiza wypowiedzenia na składniki bezpośrednie. Podstawowym krokiem takiej analizy jest konstatacja, że dany fragment wypowiedzenia, interpretowany jako pewna jednostka składniowa, może być rozłożony na pewien ciąg składników (występujących w ustalonej kolejności). Krok taki można opisać regułą, składającą się z lewej strony, która charakteryzuje jednostkę rozkładaną, i prawej strony, która wymienia ciąg jej składników.

Reguły o takim kształcie pozwalają zapisać gramatykę bezkontekstową. Formalizm DCG zawiera istotne rozszerzenie: jednostkom składniowym mogą być mianowicie przyporządkowywane parametry, opisujące cechy gramatyczne jednostek. Jeżeli każdy z parametrów przyjmuje wartości ze skończonego zbioru, to gramatyka nadal należy do klasy bezkontekstowych (ma jedynie szansę być zapisana w bardziej zwartej formie). Jeżeli jednak, jak w DCG, dopuści się jako parametry dowolne termy, istotnie zwiększa się siła wyrazu formalizmu.

Gramatyka DCG jest zbiorem reguł. Reguła składa się z lewej i prawej strony rozdzielonych strzałką (⟶). Jednostki składniowe są reprezentowane w regułach przez symbole nieterminalne (jednostki nieterminalne, krótko: nieterminale). Symbole nieterminalne mają postać termów, których funktor główny jest nazwą jednostki (na przykład **wypowiedzenie**

lub **zr** — zdanie równorzędne). Argumentami tego funktora są parametry jednostek składniowych.

Oto pierwsza reguła gramatyki Świdzińskiego (reguła (w1)), głosząca, że jednostka **wypowiedzenie** może zostać zdekomponowana na występujące po sobie jednostki **zr** i **znakkonca**:

wypowiedzenie $\longrightarrow$ (w1)
  zr(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Neg*, *I*, *Z***)**,
  znakkonca(*Z***)**.

Jednostka **znakkonca** ma jeden argument, którym jest parametr zależności Z. Jednostka **zr** ma więcej argumentów. Wszystkie jej argumenty, występujące w tej regule, są zmiennymi (nazwy pisane wielką literą), tak więc w tej regule dopuszczalne jest zdanie równorzędne o dowolnych wartościach argumentów. Źródłem jedynego ograniczenia na wartości argumentów jest powtórzenie się zmiennej Z. Oznacza ono, że wartości parametru zależności dwóch jednostek występujących po prawej stronie muszą być równe (mówiąc ściślej, że muszą one być unifikowalne, jako że cały mechanizm gramatyczny zanurzony jest w Prologu). Jest to uzgodnienie parametru zależności Z.

Ponadto w regułach pojawiają się elementy terminalne (terminale), czyli obiekty uznawane za elementarne dla tej gramatyki (nie opisywane w jej ramach). Takie elementy są w regułach ujmowane w nawiasy kwadratowe. Oto przykład reguły zawierającej element terminalny:

znakkonca(p**)** $\longrightarrow$ (int1)
  ['?'**]**.

Wyraża ona fakt, że jednostka **znakkonca** może być realizowana przez znak zapytania, reprezentowany tutaj przez prologowy atom '?'. W regule tej wartością parametru zależności jednostki **znakkonca** jest stała p — wartość pytajna.

Oczywiście elementy terminalne są „podane” na wejście gramatyki przez jakiś mechanizm względem niej zewnętrzny. W wypadku programu *Świgra* elementami terminalnymi są wyniki analizatora morfologicznego, omawiam ten mechanizm w rozdziale 4.

Ostatnim elementem, który może pojawić się po prawej stronie reguły są warunki (akcje) zapisane w formie dowolnych predykatów prologowych ujętych w nawiasy klamrowe. Oto przykład:

zaimrzecz(*F*, *P*, *Rl***)** $\longrightarrow$ (jel5)
  [*F***]**,
  { slow(*F*, zaimrzecz, *P.Rl*) }.

Według tej reguły zaimek rzeczowny **zaimrzecz** o pewnym opisie może być realizowany przez dowolny element terminalny F pod warunkiem, że dla tego elementu spełniony jest predykat slow(F, zaimrzecz, $P.Rl$) (czyli pod warunkiem, że słowo F jest w słowniku opisane jako realizacja zaimka rzeczownego o przypadku P i rodzaju-liczbie Rl).

Reguły opisane dotychczas, mają po lewej stronie jedną jednostkę nieterminalną, po prawej zaś ciąg jednostek nieterminalnych i elementów terminalnych. Wywód prowadzony

za pomocą takich reguł dostarcza pewnego drzewa, nazywanego drzewem analizy. Jest to mianowicie drzewo, którego wierzchołki wewnętrzne odpowiadają zastosowanym regułom, a liście elementom terminalnym i pustym realizacjom jednostek nieterminalnych. Tak otrzymane drzewo analizy jest drzewem składników bezpośrednich danego wypowiedzenia.

Formalizm DCG dopuszcza pewną formę reguł kontekstowych, mianowicie po lewej stronie reguły może stać jednostka nieterminalna i ciąg elementów terminalnych (na prawo od nieterminala). Dla gramatyki kontekstowej w ogólnej postaci zamiast drzewa analizy otrzymuje się pewien acykliczny graf analizy. Jednak w wypadku prostego kontekstu, polegającego na wymienieniu pewnych elementów terminalnych zarówno po lewej jak i prawej stronie reguły, można uznać, że elementy te w ogóle nie podlegają działaniu reguły, a tylko ich obecność stanowi pewien dodatkowy warunek stosowalności reguły, nieistotny dla kształtu tworzonego grafu (drzewa).

W gramatyce Świdzińskiego reguły kontekstowe są użyte w takiej właśnie formie:

przec, **[**'?'**]** ⟶ (int7)
['?'**]**.

Reguła ta głosi, że jednostka **przec** (przecinek składniowy) może mieć realizację pustą w prawostronnym kontekście znaku zapytania.

W gramatyce Świdzińskiego prawie wszystkie wartości argumentów jednostek nieterminalnych są atomami. Jednak według koncepcji Colmerauera argumenty te mogą być dowolnymi termami. Termy w naturalny sposób odpowiadają pewnym strukturom drzewiastym. Nazwa gramatyki metamorficzne wiąże się ze spojrzeniem na gramatykę, według którego reguły gramatyczne opisują pewne przekształcenia (metamorfozy) drzew reprezentowanych w postaci termów. W koncepcji tej argumenty jednostek nie tylko służą do wyrażania ograniczeń stosowalności poszczególnych reguł, ale także do budowy struktur reprezentujących analizowany tekst. Budowanie struktury nie jest uwzględnione w GFJP, dlatego w programie *Świgra* został wprowadzony mechanizm automatycznie konstruujący drzewa analizy.

Według Abramsona i Dahl (1989, s. 78–79) między gramatykami metamorficznymi i Definite Clause Grammars jest taka różnica, że gramatyki metamorficzne dopuszczają po lewej stronie reguł prawostronny kontekst (ciąg nieterminali i terminali), zaś reguły DCG są ściśle bezkontekstowe. W przypisie zaznaczono, że prologowa realizacja gramatyk metamorficznych dopuszcza jedynie kontekst w postaci ciągu elementów terminalnych. Tymczasem w znanych mi realizacjach języka Prolog pod nazwą DCG występuje mechanizm dopuszczający taki właśnie prawostronny kontekst złożony z terminali. Należy więc chyba uznać, że oba formalizmy są równoważne, zwłaszcza że wzmiankowana różnica nie wpływa na teoretyczną siłę wyrazu gramatyk — dzięki argumentom jednostek nieterminalnych i warunkom wszystkie warianty pozwalają zapisać gramatykę dowolnego języka zerowej klasy Chomsky'ego.

Gramatyki metamorficzne są zwykle prezentowane w notacji marsylskiej Prologu, a DCG w notacji edynburskiej, która obecnie jest jedyną używaną. O różnicach notacyjnych piszę w następnym punkcie.

2.2. Różnice między notacją stosowaną w programie *Świgra* i książce Świdzińskiego

Świdziński zapisuje swoją gramatykę w sposób zbliżony do oryginalnej notacji gramatyk metamorficznych. Oto przykłady reguł GFJP w oryginalnym zapisie (marsylskim):

```
ZR (wf, a, c, t, rl, o, neg0, i, z)
     = ZSZ (wf, a, c, t, rl, o, neg, i, z)                (R8)
       PRZEC
       SPÓJ (RC, oz, NI)
       ZSZ (wf, a1, c1, t1, rl1, 3, neg1, NI, z)
       $ RÓWNE (z, BY".CHOĆBY.CZYŻBY.GDYBY.JAKBY.JAKOBY.ŻEBY)
       $ OBLNEG (oz, neg0, neg, neg1)
     = ZSZ (wf, a, c, t, rl, o, neg, i, z)                (R9)
       PRZEC
       ZSZ (wf, a1, c1, t1, rl1, 3, neg1, i1, z)
       $ RÓWNE (z, BY".CHOĆBY.CZYŻBY.GDYBY.JAKBY.JAKOBY.ŻEBY)
       $ RÓWNE (i1, NATOMIAST.ZAŚ)
       $ OBLNEG (oz, neg0, neg, neg1).

ZNAKKOŃCA (P)            = # ? .                          (INT1)

MORFAGL (f, rl, o)       = # f                            (JEL1)
                           $ SŁOW (f, MORFAGL, rl, o).
```

Poniżej te same reguły w notacji stosowanej w programie *Świgra* (edynburskiej):

```
zr(Wf, A, C, T, Rl, O, Neg0, I, Z) ⟶                      (r8)
    zsz(Wf, A, C, T, Rl, O, Neg, I, Z),
    przec,
    spoj(rc, Oz, ni),
    zsz(Wf, A1, C1, T1, Rl1, 3, Neg1, ni, Z),
    { rowne(Z, [byxx, choćby, czyżby, gdyby, jakby, jakoby, żeby]),
    oblneg(Oz, Neg0, Neg, Neg1) }.
zr(Wf, A, C, T, Rl, O, Neg0, I, Z) ⟶                      (r9)
    zsz(Wf, A, C, T, Rl, O, Neg, I, Z),
    przec,
    zsz(Wf, A1, C1, T1, Rl1, 3, Neg1, I1, Z),
    { rowne(Z, [byxx, choćby, czyżby, gdyby, jakby, jakoby, żeby]),
    rowne(I1, [natomiast, zaś]),
    oblneg(Oz, Neg0, Neg, Neg1) }.
znakkonca(p) ⟶                                            (int1)
    ['?'].
```

```
morfagl(F, RI, O) ⟶                                    (jel1)
   [F],
   { slow(F, morfagl, RI, O) }.
```

Najbardziej rzucającą się w oczy różnicą jest zapewne odwrotne stosowanie wielkich i małych liter — w wersji programu *Świgra* nazwa zaczynająca się małą literą oznacza stałą, wielką zaś — zmienną[1]. Inne też są konwencje wprowadzania terminali i warunków. W notacji edynburskiej elementy prawej strony rozdzielane są przecinkami (cała reguła syntaktycznie jest termem z funktorem głównym ⟶).

Świdziński stosuje w symbolach wartości ustalonych niektórych parametrów znak ’ (prim). Ponieważ znak ten trudno wprowadzić w nazwie prologowego atomu, został on zastąpiony literą x (która nie występowała w oryginalnej formie gramatyki).

W nazwach niektórych jednostek nieterminalnych GFJP występowały polskie znaki akcentowe. Ponieważ sprawiało to kłopot w starszej wersji interpretera Prologu, zostały one zamienione na odpowiednie litery łacińskie w jednej z wczesnych wersji programu. Polskie znaki akcentowe występują natomiast w atomach reprezentujących słowa (segmenty) i niektóre wartości parametrów[2].

Aby dopełnić prezentacji notacji, warto zwrócić uwagę, że każda z reguł gramatyki Świdzińskiego jest opatrzona nazwą–numerem, która ją jednoznacznie identyfikuje. Nazwy te są obecne w drzewach analizy generowanych przez program *Świgra*.

Świdziński stosuje w regułach dwa skróty notacyjne. Reguła z fragmentem ujętym w nawiasy kątowe oznacza grupę reguł różniących się jedynie kolejnością elementów w tych nawiasach (wszystkie permutacje). Kwestię takich reguł omawiam szczegółowo w punkcie 5.2.

Drugi skrót polega na tym, że jeżeli w pewnej grupie reguł zapisanych razem lewa strona jest taka sama, Świdziński zapisuje ją tylko raz, na początku grupy. W niniejszej pracy będę zawsze pokazywał kompletne reguły, bez wykorzystania tego skrótu.

W jednej regule, (int5), Świdziński wykracza poza gramatykę metamorficzną. Mianowicie jest to reguła kontekstowa, w której kontekst terminalowy pojawia się po lewej stronie jednostki nieterminalnej. Sposób realizacji tej reguły w programie omawiam w punkcie 5.6.

2.3. Interpretacja

Gramatyka metamorficzna w prosty sposób może być przetłumaczona na program w Prologu rozpoznający język tej gramatyki. Na przykład regułę gramatyki

s ⟶ **p**, **q**, **r**.

można przetłumaczyć na klauzulę programu

s(*S0*, *S3*) :− p(*S0*, *S1*), q(*S1*, *S2*), r(*S2*, *S3*).

[1] W istocie w notacji marsylskiej Prologu stosowane były wyłącznie wielkie litery, a zmienne były oznaczane poprzedzającą gwiazdką.

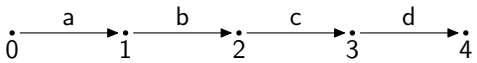
[2] Te elementy były uprzednio reprezentowane w formie napisów. Ponieważ w obecnych interpreterach Prologu jest to możliwe, w programie stosuję wygodną reprezentację w formie atomów.

Zmienne wprowadzone jako argumenty nieterminali mają reprezentować pozycje w analizowanym tekście. Chodzi o to, aby nieterminal **s** odwzorować w predykat s(*S0*, *S3*) o interpretacji: fragment analizowanego tekstu od pozycji *S0* do *S3* da się zanalizować jako wystąpienie jednostki **s**. Całą powyższą klauzulę odczytuje się więc: jeżeli od pozycji *S0* do *S1* rozpoznano jednostkę **p**, od *S1*, do *S2* rozpoznano **q**, zaś od *S2*, do *S3* rozpoznano **r**, to wystarczy to do stwierdzenia obecności jednostki **s** od *S0* do *S3*.

W przykładzie dla prostoty jednostki nieterminalne nie miały argumentów. Argumenty takie niezmienione stają się argumentami predykatów prologowych (dla ustalenia uwagi można przyjąć, że pierwsze dwa argumenty predykatu reprezentują pozycje w tekście, a pozostałe są argumentami symbolu nieterminalnego).

Aby dopełnić przekształcenia gramatyki w program trzeba powiedzieć, jak interpretować elementy terminalne gramatyki. W wielu podręcznikach Prologu wychodzi się w tym celu od reprezentacji napisów w postaci list różnicowych (ta prezentacja pochodzi zresztą od Colmerauera 1978). Można jednak zastosować i inny sposób reprezentacji naturalny dla Prologu — reprezentację klauzulową. Powiedzmy, że zdefiniowany jest predykat input/3 o interpretacji: input(*S0*, *S1*, *F*) znaczy, że między pozycjami *S0* i *S1* wejścia jest element *F*. Jako identyfikatory pozycji mogą posłużyć dowolne różne termy, na przykład liczby naturalne. W takiej reprezentacji ciąg elementów terminalnych [a, b, c, d] zapisujemy jako

```
input(0, 1, a).
input(1, 2, b).
input(2, 3, c).
input(3, 4, d).
```



Każdy element terminalny w regule gramatycznej zastępuje się wywołaniem predykatu input/3 z odpowiednią parą zmiennych. Warunki są przenoszone do klauzul programu w niezmienionej postaci. Na przykład regule

p ⟶ **t(***X***)**, **[***F***]**, { pasuje(*F*, *X*) }.

odpowiada klauzula programu

```
p(S0, S2) :- t(S0, S1, X), input(S1, S2, F), pasuje(F, X).
```

Dzięki takiej konstrukcji pytanie, czy dany ciąg elementów terminalnych należy do języka danej gramatyki i czy da się zinterpretować jako jednostka nieterminalna **s**, jest równoważne kwestii, czy s(*S0*, *Sn*) jest logiczną konsekwencją klauzul stanowiących tłumaczenie gramatyki i klauzul predykatu input/3 (przy założeniu że *S0* i *Sn* reprezentują końce analizowanego napisu).

Większość współczesnych realizacji Prologu zawiera mechanizm tłumaczenia reguł DCG na procedury prologowe realizujące analizę zstępującą (ang. *top-down*). Oczywiście można przetłumaczyć reguły gramatyki metamorficznej na klauzule realizujące inną strategię analizy. Zagadnieniu temu poświęcona jest większość rozdziału 3.

2.4. Graf acykliczny jako reprezentacja wejścia dla gramatyki

W poprzednim punkcie lista elementów terminalnych, stanowiących wejście dla gramatyki metamorficznej, była reprezentowana klauzulowo w postaci grafu o etykietowanych krawędziach. Mechanizm analizy w żaden sposób nie ogranicza postaci grafu do prostej listy. Nasuwa się więc pytanie, czy warto byłoby dopuścić na wejściu analizy graf o bardziej skomplikowanej strukturze.

W takim sformułowaniu zadanie analizy polegałoby na znalezieniu wszystkich dróg w grafie wejściowym, prowadzących od wyróżnionego wierzchołka początkowego do wyróżnionego wierzchołka końcowego, takich że ciąg etykiet na łukach grafu daje się wywieść z symbolu początkowego gramatyki.

W wypadku analizy języka polskiego elementami terminalnymi gramatyki są słowa zinterpretowane przez analizator morfologiczny. Bardzo częstym zjawiskiem są niejednoznaczności interpretacyjne, są one także jednym ze źródeł problemów z efektywnością analizy.

Jeżeli analizowane wypowiedzenie zawiera kilka niejednoznacznych słów, liczba interpretacji fleksyjnych całości jest iloczynem liczby interpretacji dla poszczególnych słów. Byłoby więc korzystnie rozpatrywać te możliwości łącznie, prowadząc analizę składniową na strukturze reprezentującej je wszystkie.

Graf acykliczny, którego krawędzie są etykietowane elementami terminalnymi spełnia ten postulat. W rozdziale 4.5 przedstawię, jaką postać może mieć graf wejściowy stanowiący wynik analizy morfologicznej. Liczba możliwych ścieżek w tym grafie może być duża, ale nie ma potrzeby przetwarzać ich oddzielnie (dotyczy to w szczególności mechanizmu zapamiętywania wyników częściowych). Ponadto dla języka polskiego własności fleksyjne form pozwalają wyeliminować większość interpretacji już przy dosyć lokalnej analizie, tak więc przy wstępującej strategii analizy składniowej praca na grafie jest dobrym sposobem radzenia sobie z niejednoznacznościami wyników analizy morfologicznej.

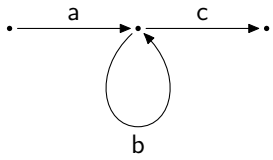
Na przedstawiony mechanizm można też patrzeć w ten sposób, że elementów terminalnych dostarcza gramatyce pewien automat skończony. Z mojego punktu widzenia, wejście dla gramatyki jest bardziej strukturą danych, niż mechanizmem je dostarczającym, tak więc będę raczej mówił o etykietowanym grafie.

Zauważmy, że automat ten niekoniecznie jest deterministyczny. Dopuszczenie wielu łuków z tą samą etykietą wychodzących z danego wierzchołka pozwala wprowadzić do grafu ścieżki reprezentujące jakąś informację zależną od kontekstu (sąsiedztwa łuków).

W pewnych wypadkach mogłoby być interesujące dopuszczenie grafu cyklicznego, to jednak może mieć niekorzystny wpływ na własność stopu zadania (por. van Noord 1995). Rozważmy następującą prostą gramatykę:

z ⟶ **[**a**]**, **b**, **[**c**]**.
b ⟶ **[]**.
b ⟶ **[**b**]**, **b**.

i następujący automat jako wejście dla niej:



Łatwo widzieć, że istnieje nieskończenie wiele drzew dla takiego wejścia — gramatyka jest bowiem w stanie przypisać strukturę każdemu z nieskończenie wielu napisów akceptowanych/emitowanych przez ten automat.

Oczywiście nie jest prawdą, że nieskończenie wiele drzew pojawia się w wyniku zestawienia dowolnego cyklicznego automatu z dowolną gramatyką (ani nawet z dowolną rekurencyjną gramatyką). Niestety rekurencja w GFJP i „pętelki" w grafie wejściowym, które bardzo wygodnie byłoby wprowadzić ze względów opisanych w punkcie 5.6, przypominają powyższy przykład.

W tej pracy przyjmuję więc, że wejście dla gramatyki stanowi acykliczny skierowany graf z etykietowanymi krawędziami (acykliczny niedeterministyczny automat skończony).

2.5. Definicja formalna

Colmerauer w swoim artykule (1978) definiuje gramatykę metamorficzną tak, że elementami reguł mogą być jedynie termy w pełni ukonkretnione. W podawanych przez niego regułach przykładowych pojawiają się zmienne, ale taki zapis jest skrótem notacyjnym oznaczającym wszystkie ukonkretnienia reguł. W takim ujęciu zbiór reguł (czy też relacja przepisywania) może być nieskończony.

W naszym zadaniu chcielibyśmy móc dostawać wyniki zawierające nieukonkretnione zmienne (wyrażające dowolność wartości pewnego parametru gramatycznego) jak również dopuścić zmienne w strukturze elementów terminalnych. Wymaga to nieco innego sformułowania definicji gramatyki. W szczególności w definicji relacji wyprowadzenia musi pojawić się unifikacja termów zamiast ich porównywania.

Gramatykę metamorficzną G definiujemy jako czwórkę uporządkowaną $G = \langle T, N, \mathbf{s}, \longrightarrow \rangle$, gdzie $T, N \subseteq \mathcal{T}$ dla $\mathcal{T}$ będącego pewnym zbiorem termów (ze zmiennymi), T jest zbiorem elementów terminalnych, N jest zbiorem symboli nieterminalnych, przy czym $N \cap T = \emptyset$. Ponadto $\mathbf{s} \in N$ jest wyróżnionym symbolem początkowym, zaś $\longrightarrow \subseteq NT^* \times (N \cup T)^*$ jest zbiorem reguł[3].

Wprowadzimy teraz graf wejścia $W = \langle P, K, p, q, \tau \rangle$, gdzie P jest zbiorem pozycji (wierzchołków), $K \subseteq P \times P$ jest zbiorem krawędzi, $p, q \in P$ są wyróżnioną pozycją początkową i końcową, $\tau : K \to T$ jest funkcją etykietującą krawędzie elementami terminalnymi.

Zbiorem ścieżek $Paths(W)$ nazwiemy zbiór wszystkich ciągów $t_1, \ldots, t_k$, $k \geqslant 1$, dla których istnieje ciąg pozycji $p_0, p_1, \ldots, p_k$ taki, że $p_0 = p$, $p_k = q$, $(p_{i-1}, p_i) \in K$ i $\tau((p_{i-1}, p_i)) = t_i$ dla $1 \leqslant i \leqslant k$.

Relacja bezpośredniego wyprowadzenia $\Rightarrow_G$ jest określona na $(N \cup T)^*$. Zachodzi $\gamma\alpha\delta \Rightarrow_G \gamma\beta\delta$ jeżeli istnieje reguła gramatyki $\alpha' \longrightarrow \beta'$ i najbardziej ogólne podstawienie θ unifikujące α i α' (to znaczy unifikujące odpowiadające elementy obu ciągów) i $\beta = \beta'\theta$.

[3] Zapis X^* oznacza zbiór dowolnej długości ciągów elementów zbioru X. Natomiast XY konkatenację zbiorów (zbiór konkatenacji ich elementów). Tak więc NT^* to zbiór ciągów złożonych z jednego elementu N, po którym następuje zero lub więcej elementów T.

Relacja wyprowadzenia $\Rightarrow_G^*$ jest przechodnim domknięciem relacji $\Rightarrow_G$. Tak więc $\alpha_0 \Rightarrow_G^* \alpha_k$ jeżeli istnieją $\alpha_0, \ldots, \alpha_k$, $k \geqslant 0$ takie, że $\alpha_{i-1} \Rightarrow_G \alpha_i$ dla $1 \leqslant i \leqslant k$.

Językiem $L(G)$ gramatyki G nazwiemy zbiór grafów wejścia W takich, że istnieje $\alpha \in \mathit{Paths}(W)$ takie, że $\mathbf{s} \Rightarrow_G^* \alpha$.

Podobnie jak we wspomnianym artykule Colmerauera, powyższa definicja nie uwzględnia obecności w regułach warunków. Formalne ich opisanie wymagałoby uwikłania semantyki Prologu w relację wyprowadzenia.

3. Technika analizy składniowej

W tym rozdziale omawiam sposób realizacji w Prologu efektywnego analizatora składniowego dla gramatyk metamorficznych. Prolog stanowi naturalny język dla realizacji gramatyki typu unifikacyjnego i wygodne narzędzie do eksperymentów z gramatyką i ze strategią analizy.

Omówię dwie strategie analizy, które zostały wypróbowane dla programu *Świgra*. Pierwszą można określić jako analizę wstępującą (ang. *bottom-up*) z zapamiętywaniem wyników pośrednich. Druga to analiza tablicowa (ang. *chart-parsing*). Złożoność pierwszej strategii jest wielomianowa, ale o wykładniku zależnym od gramatyki ($O(n^{k+1})$ gdzie n jest liczbą wierzchołków grafu wejściowego, a k jest długością najdłuższej prawej strony reguły), złożoność drugiej to $O(n^3)$. Druga strategia powinna zatem dawać efektywniejszy program. Jednak w praktyce okazuje się, że różnica w oszacowaniu asymptotycznym jest nieistotna dla GFJP (por. rozdz. 6.2) i można stosować pierwszy, prostszy algorytm.

Warto może zaznaczyć, że badania nad efektywnością różnych strategii analizy nie były celem tej pracy. Jak się okaże, problemem dla efektywności analizy są głównie konstrukcje, które sprawiają, że gramatyka opisuje bardzo wiele struktur dla danego wypowiedzenia. Z praktycznego punktu widzenia jest to raczej impulsem dla zastanowienia się, jak należy zmienić gramatykę, aby zablokować analizy, które najczęściej są nadmiarowe. Dopiero na drugim miejscu jest efektywność samego algorytmu analizy.

Plan rozdziału jest następujący: zaczynam od wyjaśnienia źródła nieefektywności prostych algorytmów analizy. Następnie prezentuję wariant analizy wstępującej dla gramatyk bezkontekstowych, który radzi sobie z podstawową przyczyną nieefektywności (p. 3.3). Następnie przedstawiam rozszerzenie tego mechanizmu na gramatyki metamorficzne (p. 3.4). Wreszcie omawiam strategię analizy tablicowej (p. 3.5). W ostatnim punkcie opisuję pewną technikę reprezentacji kategorii gramatycznych użyteczną w gramatyce metamorficznej.

3.1. Oszacowanie liczby drzew analizy

Analizator naszkicowany w punkcie 2.3 jest bardzo nieefektywny. Spróbujmy przyjrzeć się przyczynom tego zjawiska.

Przypomnijmy, że interesujące nas zadanie obejmuje nie tylko sprawdzenie, czy dane wypowiedzenie jest akceptowane przez daną gramatykę, ale określenie wszystkich możliwych strukturyzacji — drzew analizy.

Okazuje się, że istnieją gramatyki bezkontekstowe, dla których liczba możliwych struktur rośnie wykładniczo względem długości wypowiedzenia. Na przykład gramatyka

s $\longrightarrow$ **[**a**]**.

s $\longrightarrow$ **s**, **s**.

opisuje język a^+. Drzewami analizy przyporządkowywanymi słowu a^n ($n \geqslant 1$) są wszystkie drzewa binarne o n liściach.

Jeżeli oznaczymy przez C_n liczbę różnych drzew analizy dla ciągu długości n, to oczywiście

$$C_1 = 1$$

Na mocy drugiej reguły gramatyki ciąg długości n można podzielić w dowolnym miejscu na części długości, powiedzmy, i i $n - i$ i jako drzewa analizy dla tego ciągu brać dowolne drzewo dla części pierwszej połączone z dowolnym drzewem dla części drugiej. Zatem ich liczba wyniesie

$$C_n = \sum_{i=1}^{n-1} C_i \cdot C_{n-i}$$

Elementy tego ciągu znane są jako liczby Catalana.

$$C_{n+1} = \frac{1}{n+1}\binom{2n}{n}$$

Z powyższego wynika, że

$$C_{n+1} = \frac{2(2n-1)}{n+1} C_n$$

Więc

$$C_{n+1} \geqslant 2C_n \quad \text{dla } n \geqslant 2$$

a skoro $C_7 = 132 \geqslant 2^7$, więc

$$C_n \geqslant 2^n \quad \text{dla } n \geqslant 7$$

Czyli wzrost liczby drzew jest co najmniej wykładniczy względem długości analizowanego ciągu.

Zatem dowolny algorytm, który generuje kolejno wszystkie drzewa analizy dla gramatyki bezkontekstowej, ma pesymistyczną złożoność czasową wykładniczą.

Taki typ wzrostu liczby drzew pojawia się w sposób naturalny w motywowanej lingwistycznie gramatyce, gdy nie ma powodu by wybrać konkretne ponawiasowanie pewnej konstrukcji — wtedy lingwista dopuszcza je wszystkie. W GFJP są więc oczywiście fragmenty tego typu, na przykład w następujących wypowiedzeniach możliwe jest wiązanie zdań spójnikami równorzędnymi *i* w dowolnej kolejności.

(4) *Jan przyszedł i Piotr wyszedł i Maria czytała.*
(5) *Jan przyszedł i Piotr wyszedł i Maria czytała i Tomek płakał.*
(6) *Jan przyszedł i Piotr wyszedł i Maria czytała i Tomek płakał i Andrzej umarł.*
(7) *Jan przyszedł i Piotr wyszedł i Maria czytała i Tomek płakał i Andrzej umarł i Jerzy łkał.*

Tak więc według GFJP zdanie (4) ma dwa drzewa analizy, zdanie (5) — pięć drzew, zdanie (6) — czternaście, a zdanie (7) — czterdzieści dwa.

W publikacjach dotyczących przetwarzania języka naturalnego często pomijana jest analiza złożoności czasowej algorytmów analizy. Na przykład analizator tablicowy prezentowany w książce Gazdara i Mellisha (1989) ma pesymistyczną złożoność czasową wykładniczą zamiast $O(n^3)$ — ponieważ generuje wszystkie drzewa analizy.

Zauważmy jeszcze, że asymptotyczny pesymistyczny czas wykonania algorytmu analizy nie zależy od tego, czy analiza jest prowadzona zstępująco (jak w punkcie 2.3), czy wstępująco. W obu wypadkach trzeba bowiem dla pewnych gramatyk przejrzeć w całości tę samą przestrzeń rozwiązań. Dzieje się tak na przykład dla gramatyki rozważanej w tym punkcie. Jednak czasy obserwowane wykonania algorytmów zależą od własności stosowanej gramatyki. Eksperymenty wykonane przy wcześniejszych próbach realizacji gramatyki Świdzińskiego wskazują, że w wypadku języka polskiego warto prowadzić analizę w porządku wstępującym. Liczne uzgodnienia fleksyjne nakładają dość silne ograniczenia na łączliwość wyrazów już na poziomie dość lokalnym. Dlatego wstępujący porządek analizy pozwala szybciej odrzucić część możliwości. W związku z tym w tym rozdziale skupiam się na algorytmach pracujących w porządku wstępującym.

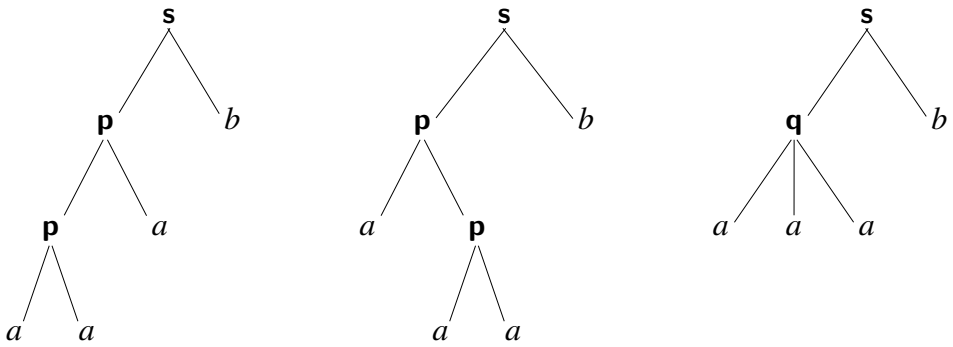
3.2. Drzewa analizy w formie upakowanego lasu

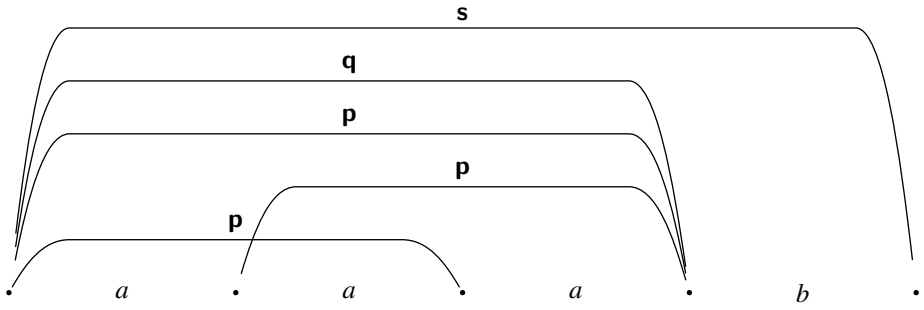
W świetle rozważań z poprzedniego punktu staje się jasne, że aby uniknąć wykładniczej złożoności czasowej trzeba zrezygnować z generowania drzew analizy kolejno. Ponadto, aby uniknąć wykładniczej złożoności pamięciowej, nie można przechowywać każdego z wykładniczo wielu drzew osobno.

Drzewa analizy otrzymywane dla gramatyki z poprzedniego punktu są bardzo do siebie podobne — takie same poddrzewa występują w bardzo wielu drzewach. Jest to prawdą dla wszystkich gramatyk bezkontekstowych: jeżeli pewnemu wystąpieniu jednostki można przypisać kilka drzew, to drzewa dla jednostek zawierających daną będą zawierać je wszystkie jako odpowiednie poddrzewa. Nasuwa się więc pomysł, aby każde poddrzewo przechowywać tylko w jednym egzemplarzu.

Strukturę umożliwiającą taką reprezentację nazywa się upakowanymi lasami analiz (ang. *packed* lub *shared parse forests*, por. np. Billot i Lang 1989). Upakowany las jest grafem, którego wierzchołkami są wierzchołki analizowanego grafu wejściowego. Łuki upakowanego lasu są etykietowane jednostkami nieterminalnymi. Dany łuk łączy wierzchołki reprezentujące końce ciągu elementów terminalnych zanalizowanego jako dane wystąpienie jednostki nieterminalnej.

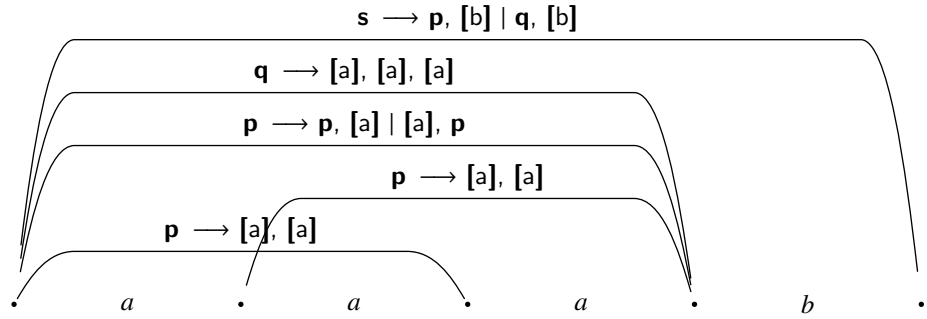
Na przykład następujące trzy drzewa można łącznie przedstawić za pomocą poniższego grafu.





Zauważmy, że graf zawiera tylko jedną krawędź etykietowaną jednostką **s**, która jest wykorzystywana we wszystkich trzech drzewach. Krawędź etykietowana **p** dominująca nad *aaa* jest wykorzystywana w dwóch drzewach.

Informacje zawarte w przedstawionym grafie nie wystarczają do odtworzenia drzew, bez odwoływania się do reguł gramatyki (wiadomo, że jednostka **s** dominuje nad ciągiem terminali *aaab*, nie wiadomo jednak, jakie są jej bezpośrednie składniki). Aby zapewnić wydobywanie konkretnych drzew z upakowanego lasu trzeba uzupełnić etykiety na łukach o prawe strony odpowiednich reguł. Dla łuków (jednostek nieterminalnych), które można uzyskać z wielu reguł, będziemy przechowywać listę prawych stron tych reguł.



W istocie dla uzyskania jednoznaczności trzeba by jeszcze podawać pozycje końców jednostek składowych. W programie *Świgra* uzyskano odtwarzalność drzew inaczej: każdy łuk otrzymał unikatowy identyfikator i prawe strony reguł są reprezentowane jako ciągi identyfikatorów odpowiednich łuków lasu. Jest to wygodne również ze względów przedstawionych w punkcie 3.4.

W programie upakowany las jest przechowywany klauzulowo w formie predykatu forest/5. Klauzula predykatu reprezentuje jeden łuk w grafie. Argumenty predykatu forest to: jednostka nieterminalna, pozycja początku jednostki w grafie wejściowym, pozycja końca, lista rozkładów na składniki bezpośrednie i wreszcie unikatowy identyfikator (numer) wierzchołka. Każdy element listy rozkładów odpowiada jednej regule, za pomocą której można uzyskać daną jednostkę nieterminalną z danymi wartościami atrybutów. Jest on listą zawierającą jednostki nieterminalne (z argumentami) i identyfikatory etykietowanych nimi łuków. Jednostki nieterminalne są potrzebne aby przy wydobywaniu drzewa móc odtworzyć stan ukonkretnienia zmiennych (por. p. 3.4). Ponadto rozkłady opatrzone są nazwami użytych reguł gramatyki.

Dla grafu wejściowego o n wierzchołkach upakowany las dla gramatyki bezkontekstowej może mieć $O(n^2)$ łuków. Widać to łatwo, ponieważ dowolne dwa wierzchołki mogą być połączone co najwyżej tyloma łukami, ile jest różnych jednostek nieterminalnych w gramatyce — jest to więc stała zależna od gramatyki. Natomiast liczba różnych par wierzchołków,

które mogłyby być połączone łukiem, jest ograniczona przez liczbę łuków grafu pełnego o n wierzchołkach, która jest $O(n^2)$.

Zatem przechowywanie wyników w postaci upakowanego lasu analiz pozwala zmieścić wykładniczą liczbę drzew w wielomianowej pamięci. Oczywiście wydobycie wszystkich tych drzew wymaga wykładniczego czasu, jednak do pewnych zadań nie jest to konieczne. W czasie wielomianowym można podać liczbę zawartych w lesie drzew, odpowiedzieć, czy między dwoma wierzchołkami da się rozpiąć daną jednostkę nieterminalną (w szczególności więc, czy dane wejście jest akceptowane przez gramatykę), a także wymienić wszystkie rozpoznane frazy stanowiące części drzew dla całego wypowiedzenia.

3.3. Analiza wstępująca z generowaniem upakowanego lasu

Upakowany las analiz jest nie tylko poręczną reprezentacją wyników, ale może również być podstawą mechanizmu zapewniającego efektywność samego procesu analizy.

Poniżej przedstawię mechanizm analizy, w którym do zastosowania reguły może dojść tylko w następstwie dodania łuku do lasu analiz. Liczba reguł, które mogą zostać użyte w wyniku dodania jednego łuku jest stałą zależną od gramatyki. Jeśli by więc zapewnić, że obliczenie pojedynczej reguły zakończy się w czasie rzędu n, otrzyma się analizator o złożoności $O(n^3)$. Tak jest faktycznie dla gramatyk w postaci normalnej Chomskiego — o czym będzie mowa dalej w tym punkcie.

W tym punkcie przyjmijmy, że mamy do czynienia z gramatyką bezkontekstową. Sposób rozszerzenia algorytmu na gramatyki metamorficzne omówię w następnym punkcie. Ponadto dla uniknięcia nieistotnych szczegółów technicznych przyjmijmy, że na łukach upakowanego lasu będziemy pamiętać jedynie etykietujące je jednostki nieterminalne (czyli algorytm będzie budować uproszczony las odpowiadający pierwszemu grafowi na ilustracji w punkcie 3.2, s. 38). Predykat forest będzie więc trójargumentowy.

Podobnie jak w algorytmie z punktu 2.3 z każdą jednostką nieterminalną **nt** związemy predykat prologowy nt(*W0*, *W1*), którego argumenty będą reprezentować pozycję początkową i końcową danego wystąpienia jednostki. Następującej regule gramatyki:

s ⟶ **p**, **q**, **r**.

będzie odpowiadać klauzula, której nagłówkiem jest pierwsza jednostka prawej strony reguły, zaś lewa strona staje się ostatnim predykatem:

```
p(W0, W1) :- goal(q, W1, W2), goal(r, W2, W3), register(s, W0, W3), s(W0, W3).
```

Predykat goal(*NT*, *W0*, *W1*) jest spełniony wtedy i tylko wtedy, gdy *NT*(*W0*, *W1*). Niestety dla wyjaśnienia algorytmu trzeba odwołać się do opisu proceduralnego, ponieważ istotną rolę pełnią tu pozalogiczne predykaty dynamicznie definiujące klauzule predykatów. Przedstawiona klauzula jest wywoływana w momencie, gdy zostało rozpoznane wystąpienie **p** od *W0* do *W1*. Pomocnicza procedura goal/3 ma za zadanie sprawdzić, czy da się rozpoznać jednostkę będącą jej argumentem. W przykładowej klauzuli trzeba się upewnić, że po **p** da się rozpoznać **q** (od *W1* do pewnego *W2*) i **r** (od *W2* do *W3*). Wtedy można stwierdzić, że zostało rozpoznane wystąpienie jednostki **s**, co jest sygnalizowane przez wywołanie procedury s/2. Jak widać, analiza przebiega w strategii wstępującej.

Przed wywołaniem predykatu odpowiadającego lewej stronie reguły jest wywoływany predykat register/3. Zauważmy, że w tym miejscu wykonania klauzuli wiadomo już, że udało się rozpoznać jednostkę z lewej strony reguły (w przykładzie **s**). Tak więc wywołanie predykatu register/3 powinno spowodować dodanie do lasu łuku od *W0* do *W3* etykietowanego jednostką **s**, jeżeli taki łuk jeszcze nie występował.

W ten sposób analizator zbuduje upakowany las analiz. Nie osiągnęliśmy jednak jeszcze żadnej oszczędności czasowej. Zauważmy, że jeżeli w momencie wywołania predykatu register odpowiedni łuk już jest obecny w lesie, znaczy to że predykat s był już wywoływany z takimi samymi argumentami. Ponowne wywołanie nie dostarczy więc żadnych nowych rozwiązań i można je zablokować. Oto definicja predykatu register/3:

```
register(NT, Od, Do) :- forest(NT, Od, Do), !, fail.
register(NT, Od, Do) :- assert(forest(NT, Od, Do)).
```

Elementy terminalne w regułach są zamieniane odpowiednio na wystąpienia predykatu terminal/3 i funktora terminal/1 w argumencie predykatu goal/3.[1] Na przykład regułom

r ⟶ **[**śpi**]**.
q ⟶ **p**, **[**się**]**.

odpowiadają klauzule

```
terminal(W0, W1, śpi) :- register(r, W0, W1), r(W0, W1).
p(W0, W1) :- goal(terminal(się), W1, W2), register(q, W0, W2), q(W0, W2).
```

Pozostaje powiedzieć, jak zdefiniować procedurę goal/3. Oprócz użyć w klauzulach odpowiadających regułom gramatyki, procedura ta będzie służyć do uruchomienia analizy. Ponieważ analiza odbywa się wstępująco, pierwszym zadaniem procedury goal jest pobranie kolejnego elementu z grafu wejściowego, a następnie wywołanie predykatu terminal/3, jako że właśnie nastąpiło rozpoznanie „jednostki nieterminalnej” terminal.

```
goal(NT, W1, W3) :- input(W1, W2, F), register(terminal(F), W1, W2),
    terminal(W1, W2, F).
goal(NT, W1, W3) :- forest(NT, W1, W3).
```

Z góry wiadomo, że wywołanie predykatu terminal zawiedzie, ponieważ graf wejściowy jest skończony, zaś każda klauzula odpowiadająca regule gramatyki kończy się próbą sprawdzenia, jak da się rozszerzyć rozpoznana jednostka. Tak więc w końcu okaże się, że jednostka nie da się rozszerzyć — z powodu braku pasującej reguły lub braku dalszych elementów terminalnych. Zanim jednak to się stanie, do lasu zostaną dodane łuki odpowiadające wszystkim jednostkom, które da się rozpoznać poczynając od *W1*. Zatem w tym momencie odpowiedź, czy da się rozpoznać nieterminal *NT*, można uzyskać zaglądając do lasu analiz (predykat forest/3).

[1] W gramatyce nie może w związku z tym wystąpić jednoargumentowa jednostka nieterminalna o tej nazwie.

Działanie tego analizatora jest więc dość zabawne: z góry wiadomo, że każde wywołanie predykatu odpowiadającego nieterminalowi gramatyki zawodzi[2], ale to nie przeszkadza zgromadzić kompletnego lasu analiz.

Spróbujmy teraz oszacować pesymistyczną złożoność czasową algorytmu dla grafu wejściowego o n wierzchołkach, liczoną w wywołaniach procedur prologowych. Każde wywołanie predykatu odpowiadającego jednostce nieterminalnej oraz predykatu terminal/3 jest poprzedzone wywołaniem predykatu register/3. Tak więc predykaty te są wywoływane tyle razy, ile razy predykat register/3 skutkuje, ta liczba zaś jest równa liczbie łuków lasu analiz, która jest $O(n^2)$.

Oszacujmy teraz koszt jednego wywołania procedury goal/3. Nie będziemy zaliczać do niego kosztu wykonania predykatu terminal/3 kończącego pierwszą klauzulę procedury goal. Odbywa się ono w następstwie wywołania procedury register, w związku z czym jego koszt uwzględnimy w sumarycznym czasie wykonywania predykatów odpowiadających jednostkom nieterminalnym.

Wywołanie procedury goal/3 jest obarczone kosztem proporcjonalnym do liczby sukcesów, jakie może dać to wywołanie. Koszt pierwszej klauzuli liczony w wywołaniach predykatów jest stały. Koszt jednego sukcesu drugiej klauzuli jest równy kosztowi pobrania łuku z lasu, który jest stałą, jeśli liczymy wywołania predykatów. Druga klauzula może dać $O(n)$ sukcesów, ponieważ z danego wierzchołka może wychodzić co najwyżej liniowa liczba łuków (prowadzą one przecież do różnych wierzchołków grafu). W sumie przypisujemy wykonaniu procedury goal/3 koszt $O(n)$.

Niech k będzie liczbą elementów najdłuższej prawej strony reguły. Wówczas wykonanie klauzuli odpowiadającej jednej regule zawiera co najwyżej $k-1$ wywołań procedury goal i w związku z tym ma koszt $O(n^{k-1})$. Zatem w takim wypadku liczba wywołań procedur w całym algorytmie jest $O(n^2 \cdot n^{k-1})$, czyli $O(n^{k+1})$.

Jeżeli gramatyka jest w postaci normalnej Chomskiego[3], to algorytm ma pesymistyczną złożoność czasową $O(n^3)$.

Algorytm zrealizowany w programie *Świgra* ma faktycznie złożoność $O(n^{k+2})$. Bierze się ona stąd, że liczba alternatywnych analiz na łukach lasu może być rzędu $O(n)$ (np. dla gramatyki rozważanej w p. 3.1). To zaś sprawia, że dodanie łuku do lasu ma w analizatorze *Świgra* czas liniowy, a nie stały, bo alternatywne interpretacje są pamiętane w postaci listy. Czynnik ten można by wyeliminować przechowując alternatywne interpretacje w tablicy.

3.4. Komplikacje związane z nieukonkretnionymi zmiennymi

W poprzednim punkcie omówiłem algorytm działający dla gramatyk bezkontekstowych. Jego rozbudowa do gramatyk metamorficznych wymaga pewnych zmian. Obecność nieustalonych zmiennych wśród argumentów jednostek nieterminalnych komplikuje bowiem zapamiętywanie wyników pośrednich.

[2] Jest tu drobna kwestia techniczna: dla obecnych interpreterów Prologu wywołanie niezdefiniowanego predykatu jest błędem wykonania programu. Dlatego dla wszystkich jednostek nieterminalnych, które nie występują jako pierwszy element prawej strony żadnej reguły, trzeba dodać klauzule, które zawodzą.

[3] Ściślej, wystarczy żeby każda prawa strona reguły składała się z nie więcej niż dwóch elementów. W postaci normalnej Chomskiego prawe strony muszą składać się albo z dwóch jednostek nieterminalnych albo z jednego elementu terminalnego.

Zauważmy najpierw, że dla gramatyk metamorficznych, w których użyto pozalogicznych predykatów w warunkach (na przykład var) wynik istotnie zależy od kolejności wykonania obliczeń. Na przykład dla następującej gramatyki i wejścia [a, b], analizator może rozpoznać lub odrzucić jednostkę **s** w zależności od porządku obliczeń.

s(*X*) ⟶ **p**(*X*), **q**(*X*).
p(2) ⟶ **[**a**]**.
q(*X*) ⟶ **[**b**]**, { var(*X*) }.

Tego rodzaju warunki mają sens jedynie w gramatyce pisanej od razu z myślą o konkretnej strategii analizy. GFJP nie zawiera tego typu komplikacji, w związku z czym można zakładać, że kolejność ukonkretniania zmiennych nie ma wpływu na rozpoznanie wypowiedzenia.

Zajmiemy się teraz problemem wydobywania drzew z lasu analiz. Przyjrzyjmy się następującej gramatyce:

p(_) ⟶ **[**a**]**.
p(2) ⟶ **q**.
q ⟶ **[**a**]**.

Gramatyka ta pozwala zinterpretować *a* na wejściu jako jednostkę **p** na dwa sposoby, przypisując jej różne drzewa rozbioru. Przy tym otrzymane opisy jednostki dają się ze sobą zunifikować, opis uzyskany dzięki zastosowaniu pierwszej reguły jest bardziej ogólny. Zauważmy, że dla tych reguł konieczne jest pamiętanie dwóch łuków dla **p** — w przeciwnym razie nie będzie możliwe wydobycie drzew z właściwymi wartościami argumentu jednostki **p**.

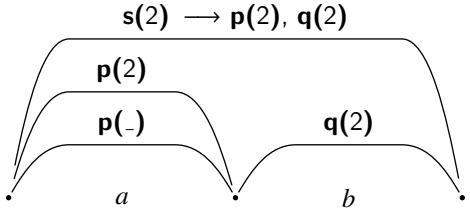
Zatem w wypadku gramatyki metamorficznej jednostka etykietująca łuk powinna mieć taki stan ukonkretnienia zmiennych, jaki wystąpił w momencie jej rozpoznania. W związku z tym przy próbie dodania nowego łuku do lasu nie można się posłużyć unifikacją do sprawdzenia, czy dany łuk jest już obecny. Konieczne jest sprawdzenie strukturalnej równoważności termów. SWI-Prolog ma służącą do tego celu procedurę wbudowaną =@=, która odnosi sukces, gdy jej argumenty są równe z dokładnością do nazw zmiennych.

Przykład ten może się wydawać nierealistyczny, jednak w GFJP mamy do czynienia z taką właśnie sytuacją. Zmienne wolne pojawiają się często w związku z niedookreśleniem wartości parametrów morfologicznych, gdy dane słowo może na przykład realizować formy wyrazowe różnych rodzajów gramatycznych.

Dla następującej gramatyki zastosowanie jej pierwszej reguły daje ten sam łuk, niezależnie od tego, która reguła definiująca jednostkę **p** została użyta:

s(*X*) ⟶ **p**(*X*), **q**(*X*).
p(_) ⟶ **[**a**]**.
p(2) ⟶ **[**a**]**.
q(2) ⟶ **[**b**]**.

Łuk ten występuje więc tylko jeden raz w tablicy wyników:

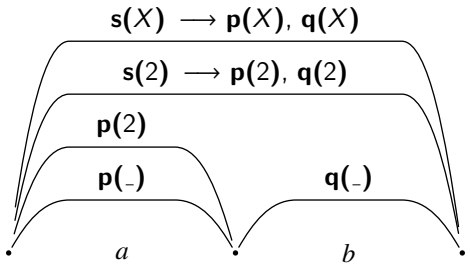


Jednak wypowiedzenie ma dwie możliwe analizy (trzecia reguła przykładu markuje inną strukturę dla **p**).

W związku z tym wydobywając drzewo analizy odpowiadające użyciu drugiej reguły, trzeba dopuścić do zunifikowania się termów **p(2)** w łuku dla pierwszej reguły i **p(_)** w łuku dla drugiej. Zauważmy, że ten drugi term jest mniej ukonkretniony. Tak być musi w poprawnie odtworzonym drzewie dla analizy wstępującej, ponieważ łuk dłuższy zawiera termy z późniejszego etapu obliczeń, więc mogą się one różnić od termów z krótszego łuku jedynie tym, że pewne zmienne zostały ukonkretnione. (Na przykład w regule pierwszej zostaje ukonkretniona zmienna występująca w **p** rozpoznanym z użyciem reguły drugiej.)

Na podstawie powyższego przykładu mogłoby wydawać się rozsądne przyjęcie następującej zasady rządzącej odtwarzaniem drzew: jako pasujące dopuszcza się łuki etykietowane termami, które są unifikowalne z danym i nie bardziej od niego ukonkretnione. Przyjrzyjmy się jednak następnej gramatyce i odpowiadającemu jej lasowi wyników:

s(*X***)** ⟶ **p(***X***)**, **q(***X***)**.
p(_) ⟶ **[**a**]**.
p(2**)** ⟶ **[**a**]**.
q(_) ⟶ **[**b**]**.



Tu również możliwe są dwa drzewa analizy. W pierwszym, uzyskanym za pomocą reguły pierwszej, drugiej i czwartej, zmienna X jest nieukonkretniona. W drugim drzewie, powstałym w wyniku użycia reguły pierwszej, trzeciej i czwartej, argumentem wszystkich jednostek nieterminalnych jest term bez zmiennych: 2.

Zaproponowana zasada poprawnie odrzuca powiązanie łuku z etykietą **s(***X***)** ⟶ **p(***X***)**, **q(***X***)** z łukiem etykietowanym **p(**2**)**.

Niestety zasada ta nie wystarczy aby poprawnie odtworzyć drzewo odpowiadające łukowi **s(**2**)** ⟶ **p(**2**)**, **q(**2**)**. W tym bowiem wypadku term **p(**2**)** jest nie mniej ukonkretniony od termów na obu łukach etykietowanych **p**, w związku z czym zaproponowana zasada dopuściłaby oba i w sumie otrzymalibyśmy trzy drzewa analizy.

W tym przykładzie widać również wyraźnie, dlaczego konieczna jest obecność obu dłuższych łuków: sam łuk **s(**2**)** ⟶ **p(**2**)**, **q(**2**)** nie pozwoli odtworzyć drugiego drzewa, natomiast sam łuk **s(***X***)** ⟶ **p(***X***)**, **q(***X***)** nie może opisywać drzewa pierwszego (bo zawarty w nim term jest bardziej ogólny).

Wydaje się, że rozsądnym sposobem wybrnięcia z sytuacji będzie nadanie każdemu łukowi w lesie unikatowego identyfikatora. Wówczas do łuków reprezentujących składniki bezpośrednie danej jednostki można będzie się odwołać poprzez ich identyfikatory (które oczywiście trzeba pamiętać, jako część reprezentacji prawych stron reguł).

Jeśli chodzi o złożoność obliczeniową, dopuszczenie obecności zmiennych nie spowoduje zmiany własności asymptotycznych, o ile istnieje tylko skończenie wiele ukonkretnień termów stanowiących argumenty jednostek nieterminalnych, czyli gdy gramatyka metamorficzna odpowiada pewnej gramatyce bezkontekstowej. Tak właśnie dzieje się dla GFJP.

3.5. Analiza tablicowa

Obecnie przyjmuje się, ze najlepsza pesymistyczna złożoność czasowa dla problemu analizy dowolnych gramatyk bezkontekstowych to $O(n^3)$. Taką złożoność obliczeniową pozwala osiągnąć np. algorytm Earley'a (1970), oraz analizatory tablicowe (ang. *chart-parsers*, por. np. Gazdar i Mellish 1989)[4].

Jak zobaczyliśmy wcześniej, złożoność $O(n^3)$ daje się uzyskać dla gramatyk w postaci normalnej Chomskiego. Algorytmy osiągające tę złożoność dla dowolnej gramatyki bezkontekstowej dokonują niejawnej konwersji gramatyki na postać normalną Chomskiego.

Ideę tej konwersji zwykle przedstawia się używając pojęcia reguły kropkowanej. Jest to reguła gramatyki z dodaną kropką między pewnymi elementami prawej strony. Kropka sygnalizuje stan zaawansowania w rozpoznawaniu prawej strony reguły — symbole przed kropką zostały już rozpoznane, symbole po kropce pozostają do rozpoznania. Reguł kropkowanych będziemy używać jako dodatkowych jednostek nieterminalnych w przekształconej gramatyce. Regułę oryginalnej gramatyki postaci

s $\longrightarrow$ **p**, **q**, **r**.

zastępujemy regułami:

s $\longrightarrow$ $\boxed{\textbf{s} \longrightarrow \textbf{pq.r}}$, **r.**

$\boxed{\textbf{s} \longrightarrow \textbf{pq.r}} \longrightarrow \boxed{\textbf{s} \longrightarrow \textbf{p.qr}}$, **q.**

$\boxed{\textbf{s} \longrightarrow \textbf{p.qr}} \longrightarrow$ **p.**

Gramatyka jest w oczywisty sposób równoważna oryginalnej w sensie opisywanego języka, ale ma inne drzewa analizy. Analizator tablicowy musi zatem odtworzyć oryginalne drzewa. Liczba reguł gramatyki przekształconej jest równa sumie długości prawych stron gramatyki oryginalnej, jest więc oczywiście nie większa niż liczba reguł oryginalnej gramatyki pomnożona przez stałą — długość najdłuższej prawej strony reguły.

Analizę dla tak zmienionej gramatyki prowadzimy za pomocą algorytmu z punktu 3.3. Powoduje to, że w lesie analiz pojawiają się łuki etykietowane regułami kropkowanymi. Struktura taka jest dokładnie tablicą fraz (ang. *chart*) używaną w analizatorze tablicowym. Krawędzie etykietowane pojedynczymi jednostkami nazywa się nieaktywnymi. Krawędzie etykietowane regułami kropkowanymi to krawędzie aktywne.

[4] Valient (1975) pokazał, że problem analizy dla gramatyki bezkontekstowej można sprowadzić do problemu mnożenia macierzy. Mnożenie macierzy można zrealizować w czasie nieco lepszym niż sześcienny, ale stałe są na tyle duże, że nie ma to sensu w zastosowaniach praktycznych.

Pracę analizatora tablicowego można opisać jako proces tworzenia nowych łuków w wyniku łączenia krawędzi aktywnej z krawędzią nieaktywną. Po każdym dodaniu do tablicy krawędzi nieaktywnej należy dodać krawędzie aktywne dla wszystkich reguł, które zaczynają się od rozpoznanej właśnie jednostki.

Patrząc w drugą stronę, dla gramatyki w postaci normalnej Chomskiego algorytm z punktu 3.3 można widzieć jako analizator tablicowy, w którym zbiór krawędzi aktywnych nie jest przechowywany w tablicy, ale jest ukryty w stosie wywołań rekurencyjnych predykatów prologowych. Taki analizator przechowuje krawędzie aktywne tylko tak długo, jak są potrzebne (są one zapominane w wyniku nawrotów). Można zatem oczekiwać mniejszego zużycia pamięci.

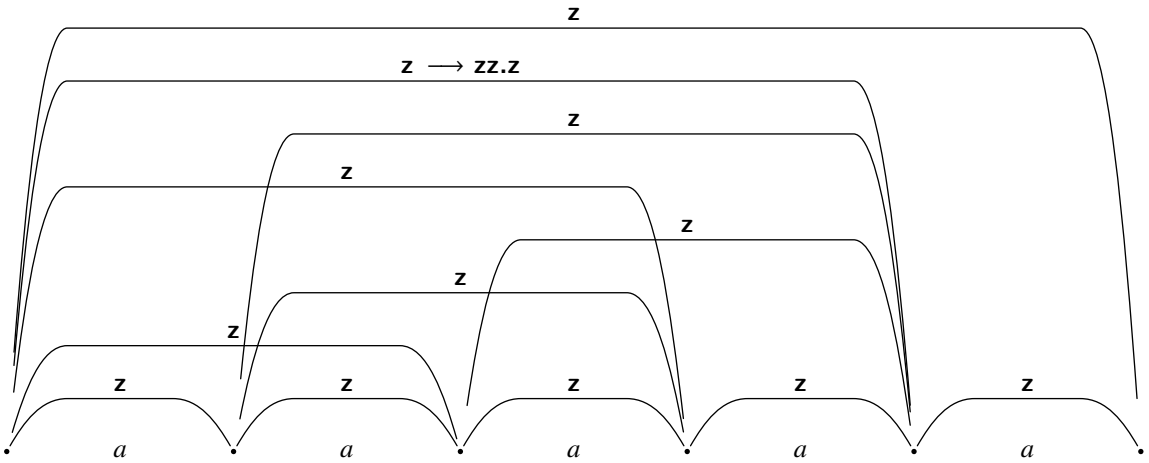
Zwiększenie efektywności analizy polega na tym, że każdy z łuków opatrzonych regułami kropkowanymi może mieć liniową względem swojej długości liczbę rozkładów na dwa elementy poprzedzające kropkę. Dzieje się tak na przykład dla gramatyki

z $\longrightarrow$ **[**a**]**.

z $\longrightarrow$ **z**, **z**.

z $\longrightarrow$ **z**, **z**, **z**.

Oto fragment tablicy fraz dla tej gramatyki:



W tablicy tej łuk etykietowany **z** $\longrightarrow$ **zz.z** zbiera pod sobą liniową ze względu na swoją rozpiętość liczbę rozkładów na dwa wystąpienia jednostki **z**. W kompletnej tablicy najwyższy łuk etykietowany **z** (rozpięty od początku do końca) zbiera pod sobą liniową liczbę łuków **z** $\longrightarrow$ **zz.z**. Gdyby tych łuków nie było, trzeba by zebrać pod najwyższym łukiem kwadratową liczbę trójek wystąpień jednostki **z**.

3.6. Unifikacyjna reprezentacja zbiorów

W gramatyce formalnej często wygodnie byłoby dopuścić, aby wartość pewnego parametru nie była konkretnie ustalona, ale został wskazany pewien zbiór wartości, które mogą przysługiwać parametrowi w danym wypadku. Na przykład w polskiej fleksji rodzaj gramatyczny jest uwikłany w liczne niejednoznaczności. Słowo *został* może być wykładnikiem trzech różnych form odpowiadających trzem rodzajom męskim (osobowemu, zwierzęcemu i rzeczowemu), a zdanie

(8) *On został.*

ma trzy możliwe interpretacje (trzy drzewa rozbioru) różniące się rodzajem przypisanym formie zaimka i czasownika. Oczywiście takie „rozmnażanie się" interpretacji jest zjawiskiem systematycznym i rozsądnie jest w takim wypadku przedstawić jedną analizę. W tym zdaniu mamy bowiem do czynienia z niedookreślonym rodzajem, który nazywamy męskim, a który rozumiemy jako trzyelementowy zbiór.

Oprócz większej elegancji wyników taki zabieg zwiększa efektywność analizy, ponieważ zmniejsza liczbę głębokich nawrotów w programie pojawiających się przy pobieraniu kolejnej alternatywnej interpretacji danego segmentu z wejścia.

Jawne przystosowanie gramatyki do pracy ze zbiorami zamiast pojedynczych wartości wymagałoby zmiany we wszystkich miejscach gramatyki gdzie mamy do czynienia z uzgodnieniem wartości „zbiorowego" parametru dwóch jednostek. Trzeba mianowicie zastąpić sprawdzenie równości parametru specyficznym operatorem uzgodnienia — mianowicie wyznaczeniem przecięcia zbiorów ze sprawdzeniem, czy wynik jest niepusty. Jeżeli wynik jest pusty, parametry są „różne" (niezgodne) i sprawdzenie zawodzi (następuje nawrót). Jeżeli wynik jest niepusty, to stanowi on zbiór wartości dla parametru, który będzie wykorzystany w dalszej analizie.

W gramatyce metamorficznej w naturalny sposób nie mamy do czynienia ze sprawdzaniem równości wartości parametrów, ale ze sprawdzaniem ich unifikowalności. Można więc spróbować reprezentować zbiory wartości w postaci takich termów, aby ich pary dawały się zunifikować jedynie, gdy ich przecięcie jest niepuste i by wynik unifikacji był reprezentacją iloczynu zbiorów. Przy takiej reprezentacji unifikacja spełnia rolę opisanego wyżej operatora uzgodnienia, dzięki czemu — w przypadku realizacji programu w Prologu — nie są konieczne żadne zmiany w regułach gramatyki.

Mellish (1988) i Pulman (1996) piszą, że pomysł takiej reprezentacji zbiorów „jest przypisywany Alainowi Colmerauerowi". Zdaniem Bienia (por. Bień w druku) Colmeraurer prezentował ten pomysł w czasie wizyty w Warszawie (prawdopodobnie w roku 1977) wskazując jako autora jego realizacji swojego współpracownika, Michaela van Caneghema.

Zauważmy, że dopuszczenie zbiorowych wartości parametrów nie stanowi ogólnego rozwiązania problemu niejednoznaczności czy też ograniczeń dopuszczalnych kombinacji wartości parametrów. Na przykład wyrazowi *ładny* można przypisać dowolny z rodzajów męskich, oraz jako wartość przypadka mianownik lub biernik. Jednak interpretacja biernikowa może przysługiwać jedynie rodzajowi męskiemu rzeczowemu. Oznacza to, że nawet dopuszczając wartości zbiorowe parametrów morfologicznych, trzeba rozważać osobno co najmniej dwie interpretacje: rodzaj męski w mianowniku i rodzaj męski rzeczowy w bierniku.[5] Dlatego reprezentacja parametrów w postaci zbiorów została w parserze *Świgra* zastosowana wybiórczo, do niektórych parametrów. W szczególności została zastosowana do parametru rodzaju, którego niejednoznaczność jest szczególnie częsta. Nie została natomiast zastosowana dla przypadka, który zwykle dla danego zbioru dopuszczalnych rodzajów ma jedną lub dwie wartości.

[5] Oczywiście zbiorem reprezentowanym za pomocą przedstawionej tu techniki można by uczynić zbiór par wartości przypadka i rodzaju (por. Pulman 1996). Gdyby jednak chcieć reprezentować tak kombinacje wartości wszystkich parametrów, liczba elementów zrobiłaby się na tyle duża, że rozwiązanie to byłoby zupełnie niepraktyczne. Ponadto wymagałoby to modyfikowania reguł gramatyki, czego chcemy uniknąć.

Przedstawię teraz sposób reprezentacji zbiorów w postaci termów. Rozważmy dla przykładu zbiór trójelementowy $\{a, b, c\}$ i załóżmy, że chcemy być w stanie reprezentować wszystkie jego podzbiory. Przykładem reprezentacji spełniającej warunki zadania są następujące termy:

$\{a\}$	zb(0, 0)
$\{b\}$	zb(0, 1)
$\{c\}$	zb(1, 1)
$\{a, b\}$	zb(0, _)
$\{b, c\}$	zb(_, 1)
$\{a, c\}$	zb(X, X)
$\{a, b, c\}$	zb(_, _)

Poniżej omówię uogólnienie tej reprezentacji na dowolny zbiór skończony (prezentacja według artykułów Mellisha i Pulmana). Weźmy dla przykładu zbiór pięcioelementowy $\{a, b, c, d, e\}$. Każdy term reprezentujący podzbiór tego zbioru będzie miał ten sam funktor główny, np. zb, i arność $n+1$, gdzie n jest mocą reprezentowanego zbioru. Mnemotechnika konstrukcji jest taka: jako szablon wypisujemy term, który jako każdy z argumentów ma inną zmienną, zaś pod przecinkami wypisujemy n elementów zbioru:

```
zb(A,B,C,D,E,F)
    a b c d e
```

Aby uzyskać reprezentację jakiegoś podzbioru, należy w tym termie zunifikować każdą parę zmiennych sąsiadującą z przecinkiem odpowiadającym elementowi, który nie należy do rozpatrywanego podzbioru. Na przykład:

$\{a\}$	zb(A, B, B, B, B, B)
$\{b\}$	zb(A, A, C, C, C, C)
$\{c\}$	zb(A, A, A, D, D, D)
$\{a, b\}$	zb(A, B, C, C, C, C)
$\{b, d\}$	zb(A, A, C, C, E, E)
$\{a, b, c\}$	zb(A, B, C, D, D, D)
$\{a, b, d\}$	zb(A, B, C, C, E, E)
$\{b, d, e\}$	zb(A, A, C, C, E, F)

Następnie należy zunifikować pierwszy i ostatni argument każdego termu odpowiednio z dwiema ustalonymi różnymi stałymi, na przykład 0 i 1:

$\{a\}$	zb(0, 1, 1, 1, 1, 1)
$\{b\}$	zb(0, 0, 1, 1, 1, 1)
$\{c\}$	zb(0, 0, 0, 1, 1, 1)
$\{a, b\}$	zb(0, B, 1, 1, 1, 1)
$\{b, d\}$	zb(0, 0, C, C, 1, 1)
$\{a, b, c\}$	zb(0, B, C, 1, 1, 1)
$\{a, b, d\}$	zb(0, B, C, C, 1, 1)
$\{b, d, e\}$	zb(0, 0, C, C, E, 1)

Łatwo sprawdzić, że wynikiem unifikacji par termów uzyskanych w tym przykładzie są dokładnie termy reprezentujące iloczyn odpowiednich zbiorów (jeśli jest on niepusty).

Idea reprezentacji polega na tym, że dwukrotne wystąpienie tej samej zmiennej wyraża wiedzę o braku w zbiorze elementu „odpowiadającego" przecinkowi między tymi zmiennymi. Informacja o braku elementów pozwala więc „przenosić" ku sobie 0 i 1 z końców termu. Jeżeli wynikiem przecinania ze sobą zbiorów miałby być zbiór pusty, 0 i 1 „spotkają się" (dojdzie do próby zunifikowania ich) i uzyska się porażkę unifikacji.

Zauważmy, że uzyskane w ten sposób termy wszystkie mają 0 jako pierwszy argument i 1 jako ostatni. Wiadomo więc z góry, że te elementy są ze sobą zgodne i można je pominąć:

$\{a\}$	zb(1, 1, 1, 1)
$\{b\}$	zb(0, 1, 1, 1)
$\{c\}$	zb(0, 0, 1, 1)
$\{a, b\}$	zb(B, 1, 1, 1)
$\{b, d\}$	zb(0, C, C, 1)
$\{a, b, c\}$	zb(B, C, 1, 1)
$\{a, b, d\}$	zb(B, C, C, 1)
$\{b, d, e\}$	zb(0, C, C, E)

Do reprezentacji podzbiorów zbioru n-elementowego wystarczy więc term arności $n - 1$ (o dziwo tej prostej obserwacji zabrakło w obu cytowanych artykułach).

Pewną wadą takiej reprezentacji jest jej słaba czytelność utrudniająca na przykład śledzenie programu. Dla pewnych ograniczonych rodzin podzbiorów możliwa jest reprezentacja bardziej czytelna, która co więcej operuje termami o strukturze hierarchicznej, co daje szansę na szybszą unifikację (lub jej porażkę). W programie *Świgra* posługuję się taką właśnie uproszczoną techniką dla reprezentacji zbiorów rodzajów, przedstawiam ją bliżej w punkcie 5.3.2.

4. Analiza morfologiczna

Jest oczywistością, że gramatyka formalna opisująca obszerny podzbiór języka polskiego musi być stowarzyszona z jakimś komponentem słownikowym. Żaden bowiem autor gramatyki nie zechce wypisać 850 000 reguł opisujących z osobna każdą z form rzeczownikowych, setek tysięcy reguł opisujących bogactwo form czasownikowych itd. Jest też aspekt techniczny sprawy: wydzielony komponent słownikowy może być zrealizowany za pomocą prostszych i bardziej efektywnych obliczeniowo środków niż właściwa analiza składniowa.

W GFJP komponent słownikowy objawia się w postaci trójargumentowego predykatu slow, który wiąże słowo tekstowe z klasą gramatyczną („częścią mowy") i zestawem charakteryzujących je parametrów. Można sobie wyobrażać, że predykat ten jest zdefiniowany w Prologu (ma on wówczas jakieś 2 000 000 klauzul[1]).

W analizatorze *Świgra* komponentem słownikowym jest analizator morfologiczny *Morfeusz*, który przyporządkowuje poszczególnym słowom polskim informacje analogiczne do argumentów predykatu slow. *Morfeusz* interpretuje tekst w zasadzie według koncepcji opracowanej na potrzeby znakowania Korpusu IPI PAN (por. Woliński 2003, Przepiórkowski i Woliński 2003a,b,c, Woliński i Przepiórkowski 2001, Przepiórkowski *et al.* 2003). Wykorzystywany w analizatorze *Świgra* system pojęciowy obejmuje tylko niektóre pojęcia występujące w koncepcji Bienia (Bień 1991, 2001, 2004), która była wykorzystywana w analizatorze AMOS (Bień 1996).

W bieżącym rozdziale przedstawiam konwencje opisu morfologicznego przyjęte w programie *Świgra*. Wypada podkreślić, że stanowią one pewną interpretację rzeczywistości językowej i w związku z tym są po części arbitralne.

4.1. Podstawowe pojęcia

Gramatyka formalna opisuje dopuszczalne sposoby zestawiania ze sobą pewnych jednostek niższego poziomu, pozwalające tworzyć z nich jednostki poziomu wyższego. W pewnych kontekstach jednostki poziomu niższego nazywa się „słowami" a wyższego „zdaniami" (wypowiedzeniami). W innych mówi się o „literach" tworzących „słowa". W tym punkcie spróbuję sprecyzować, jakie właściwie elementy stanowią ten niższy, wejściowy poziom dla analizatora składniowego *Świgra*.

Punktem wyjścia dla dalszych rozważań będzie system pojęciowy *Składni współczesnego języka polskiego* (Saloni i Świdziński 2001). Przytoczę podane tam określenia (s. 85):

> Ciąg liter pomiędzy sąsiednimi spacjami będziemy nazywać *słowem*.
> *Forma wyrazowa* to słowo z przypisaną charakterystyką gramatyczną i semantyczną (denotacyjną).

[1] Obszerny słownik języka polskiego można oszacować na 150 000 haseł. W tekstach polskich można mieć do czynienia mniej więcej z 2 000 000 różnych słów realizujących te hasła.

Leksemem będziemy nazywać zbiór form wyrazowych o identycznej lub regularnie zróżnicowanej charakterystyce semantycznej, pozostających względem siebie w regularnych opozycjach.

Tak rozumiane słowa są wyróżnione na podstawie prostego, łatwo stosowalnego kryterium. Pozostałe dwa pojęcia są bardziej abstrakcyjne — ich wyróżnienie istotnie zależy od refleksji badacza.

Definicja słowa, choć przejrzysta, nie odpowiada w pełni intuicji językowej. Wydaje się bowiem, że Autorzy *Składni* nie są skłonni uważać znaków[2] apostrofu i łącznika za litery. Tymczasem wydaje się wygodne uznanie napisu *Lagrange'a* za jedno słowo. Dotyczy to również napisów takich jak *ping-pong* i *PRL-u*. W niniejszej pracy będę traktować je jako pojedyncze słowa. Kolejnym problematycznym znakiem jest kropka, która występuje w tekście w dwóch funkcjach: jako znak interpunkcyjny oraz jako obowiązkowa część skrótu. W tym drugim wypadku kropkę traktuję jako część słowa.

Pojęcie formy wyrazowej wiąże się z trudną do doprecyzowania charakterystyką semantyczną. Będę ją tutaj rozumiał wąsko jako wskazanie leksemu, do którego dana forma należy, poprzez podanie umownego identyfikatora (nazwy) leksemu. Charakterystyka gramatyczna to, na potrzeby tej pracy, komplet cech formy wyrazowej pozwalający z zadowalającą dokładnością opisać jej dystrybucję.

Uznaję, że wypracowanie reguł wyróżniania zbiorów form o „identycznej lub regularnie zróżnicowanej" charakterystyce nie należy do zadań niniejszej pracy i będę zakładał, że leksemy są dane przez słownik morfologiczny. Z punktu widzenia tej pracy są obrane arbitralnie (choć oczywiście od tego, jaki zestaw form zaliczy się do leksemów danego typu, zależy sposób zapisu pewnych reguł gramatyki, por. 4.7; konieczne jest upewnienie się, że przyjęte granice poszczególnych leksemów są w zgodzie z intencją jej autora).

4.2. Reguły segmentacji tekstu

W tradycyjnym opisie gramatycznym można pewne mniej częste zjawiska potraktować pobieżnie. Jeżeli takie zjawiska nie mają zostać pominięte w opisie komputerowym, musi on być prowadzony rygorystycznie do najdrobniejszego szczegółu. W związku z tym pojęcia przedstawione wyżej muszą ulec pewnej modyfikacji. Otóż wydaje się konieczne uznanie, że nie zawsze forma wyrazowa jest realizowana w tekście jako słowo.

Jak zauważają Saloni i Świdziński, niekiedy zdarza się, że słowo jest wykładnikiem więcej niż jednej formy wyrazowej. Dotyczy to na przykład słowa *świniam* rozważanego na s. 96 *Składni*. Otóż

(9) *Świniam.*

jest pełnym zdaniem polskim znaczącym *Jestem świnia*. Jak argumentują Autorzy *Składni*, wypada przyjąć, że wypowiedzenie to składa się z formy rzeczownika *świnia*, skróconej formy *-m* czasownika *być* i kropki. To jednak oznacza, że w tym wypadku wykładnikiem pojedynczej formy wyrazowej nie jest słowo, ale pewna jego część właściwa, którą będę nazywać *segmentem*.

[2] W tej pracy używam słowa 'znak' bez dodatkowych określeń w znaczeniu technicznym takim jak 'znaki ASCII' lub 'znak interpunkcyjny', nie zaś 'znak językowy'.

Postuluję więc opis, w którym budulcem wypowiedzeń są segmenty interpretowane jako wykładniki tekstowe form wyrazowych. Słowa natomiast stają się technicznym pojęciem pomocniczym. W większości wypadków segmenty pokrywają się ze słowami. Zdarza się jednak, że słowo składa się z kilku segmentów.

Na segmenty nie będące słowami natykamy się również w związku z czasem przeszłym czasowników. Możliwe są mianowicie dwa warianty szyku konstrukcji wyrażającej czas przeszły:

(10) *Jak to zrobił**eś**?*
(11) *Jak**eś** to zrobił?*

Segment *zrobił* występujący w przykładzie (11) uznaję za formę czasownika *zrobić* nazwaną *pseudoimiesłowem* (za pracą Saloniego 2000). W tym wariancie szyku czas przeszły czasownika jest wyrażony za pomocą pseudoimiesłowu i osobnej formy, którą sensownie jest uznać za niesamodzielną (aglutynacyjną) formę czasownika *być*. Ta forma, a nie pseudoimiesłów, niesie informację o wartości osoby. Dla konsekwencji uznaję, że w „ciągłym" wariancie szyku czasu przeszłego również mamy do czynienia z pseudoimiesłowem i aglutynantem, czyli że słowo *zrobiłeś* składa się z segmentów *zrobił* i *eś*.

Może się wydawać, że formy aglutynacyjne czasownika *być* pojawiają się jedynie w kontekstach nacechowanych i niezbyt częstych tekstowo i że w związku z tym można je pominąć w analizie automatycznej. Jednak w pewnych kontekstach formy takie występują obligatoryjnie, mianowicie w zdaniach wprowadzanych spójnikiem lub partykułą kończącą się cząstką *by*:

(12) *. . . , gdyby**ś** naprawdę mnie kochał.*
(13) **. . . , gdyby naprawdę mnie kochałeś.*

(Przykład analizy takiej konstrukcji według reguł GFJP można znaleźć na rysunku C.7, por. także p. 5.4).

Innym przykładem słów, które rozbijam na wiele segmentów są „przymiotniki złożone" typu *biało-czerwony*. Takie słowo interpretuję jako konstrukcję złożoną z wykładników leksemów *biały* i *czerwony* połączonych łącznikiem. Dzięki temu nie jest potrzebne wprowadzenie leksemu *biało-czerwony*. Ponieważ znaczenie takiej konstrukcji jest regularne i zdeterminowane przez znaczenia jej składowych, takie rozwiązanie wydaje mi się lepsze niż wprowadzanie do słownika morfologicznego dużej liczby leksemów o niskiej frekwencji (zwłaszcza że możliwe są dłuższe ciągi typu *polsko-niemiecko-afgańsko-francusko-koreański*). Nieostatnie człony takich przymiotników są opisane jako specjalna forma przymiotnika (nazwana fleksemem przyprzymiotnikowym).

Przy segmentacji tekstu polskiego pojawia się też zjawisko odwrotne do omówionego wyżej: niektóre słowa nie wydają się być samodzielnie interpretowanymi segmentami. Przede wszystkim tradycyjnie za wykładnik jednej formy często uznaje się napisy takie jak *bała się* i *będzie czytał*. Rozważanie takich form wymagałoby wprowadzenia segmentów nieciągłych:

(14) *Bardzo **się** tego jutrzejszego egzaminu **boję**.*
(15) *Bardzo **się** tego jakże dla mnie ważnego i trudnego egzaminu **boję**.*
(16) ***Będę** to uważnie **czytał**.*

Dopuszczenie takich segmentów raczej komplikuje opis niż upraszcza. Jeśli bowiem wziąć pod uwagę wypowiedzenie:

(17) *Będzie pisał lub czytał.*

to i tak okaże się konieczne interpretowanie fragmentów takich „segmentów". W tym zdaniu *będzie* odnosi się zarówno do *pisał* jak i do *czytał*. W związku z tym, gdyby całością miałoby być dopiero *będzie pisał* i *będzie czytał*, potrzebny byłby jakiś mechanizm dotwarzający drugie *będzie*. W przeciwnym wypadku wypadałoby przyjąć, że przytoczone wypowiedzenie składa się z jednej formy czasu przyszłego leksemu PISAĆ LUB CZYTAĆ i kropki.

Innym kandydatem na segmenty wielosłowowe są wyrażenia typu *po polsku*, *po prostu*, *gdzie indziej*. Mamy tu do czynienia ze słowami, które mogą wystąpić tylko w ściśle określonym sąsiedztwie. Również takie słowa uznaję za osobne segmenty, odnosi się bowiem do nich podobny argument jak poprzednio — chyba możliwe jest takie wypowiedzenie:

(18) *Mówi po polsku i angielsku.*

Ponadto formy wyrazowe o nieciągłych wykładnikach nie mają naturalnej reprezentacji w formalizmie gramatyki metamorficznej. Dlatego przyjmuję, że wymienione napisy są wykładnikami wielu form wyrazowych, w szczególności że jest segment *się* będący wykładnikiem pełnoprawnej formy wyrazowej. Niestety opis Świdzińskiego nie uwzględnia segmentu *się* odseparowanego od segmentu czasownikowego, w związku z czym wypowiedzenia (14)–(16) nie są akceptowane przez GFJP.

Podsumowując, w tej pracy przyjmuję, że segment zawsze mieści się w słowie. Jest to uproszczenie podyktowane po części względami technicznymi. Temat jednostek leksykalnych wyrażanych więcej niż jednym słowem jest bardzo obszerny. Nie powstał jeszcze zadowalający opis formalny polskich frazeologizmów. Jak sądzę, wykrywanie takich jednostek nie należy do etapu właściwej analizy morfologicznej. W programie *Świgra* ten etap przetwarzania nie został uwzględniony, co oczywiście oznacza, że program nie akceptuje pewnych poprawnych wypowiedzeń polskich. Przetwarzanie wielosegmentowych jednostek leksykalnych jest jednym z elementów, które trzeba będzie uwzględnić w dalszym rozwoju programu.

Segmenty, podobnie jak słowa, są jednostkami unilateralnymi (w sensie Saloniego i Świdzińskiego, 2001), co nie zmienia faktu, że aby je wyróżnić trzeba tekst zananalizować morfologicznie, więc dokonać jego bilateralizacji. To samo dotyczy ustalenia, czy dany łącznik lub kropka są częścią właściwą pewnego segmentu i słowa, czy osobnym segmentem. Jak mi się zdaje, zbiór segmentów dla języka polskiego musi być zadany słownikowo. Decyzję o traktowaniu łącznym lub rozdzielnym danej konstrukcji trzeba podejmować dla każdego leksemu z osobna. Dlatego też nie podaję lepszego określenia segmentu, niż tylko „pewien (spójny) ciąg znaków interpretowany jako wykładnik formy wyrazowej".

Podział słowa na segmenty może być niejednoznaczny. Na przykład słowo *ktoś* może być jednym segmentem, ale może też składać się z segmentów *kto* i *ś*. Tę opozycję ilustrują przykłady:

(19) *Ktoś wszedł do pokoju.*
(20) *Ktoś ty?*

W związku z tym wynik analizy morfologicznej w ogólności nie jest listą segmentów z przyporządkowanymi im interpretacjami, ale grafem acyklicznym (por. p. 4.5).

4.3. Komponent słownikowy GFJP

Świdziński nie doprowadza swojego opisu do poziomu terminalnego gramatyki: podaje reguły tylko dla niektórych jednostek elementarnych. W szczególności brak reguł opisujących formy czasownikowe, rzeczownikowe i przymiotnikowe (por. 4.7.1). W związku z tym na potrzeby automatycznego analizatora opis trzeba było rozszerzyć i uzupełnić. W niniejszej pracy przedstawiam więc pewną propozycję morfologicznego domknięcia opisu Świdzińskiego. Niestety w odniesieniu do niektórych problemów trudno było ustalić intencje Autora gramatyki.

W GFJP czytamy (s. 56):

> Słownik terminalny tej gramatyki pomyślany został docelowo jako zbiór słów, tj. kształtów form wyrazowych, i znaków interpunkcyjnych. W niniejszej pracy słownika takiego z oczywistych powodów nie podaję. Zakładam roboczo, że dysponuję gotowym spisem realizacji leksykalnych tych jednostek składniowych, które występują jawnie w opisie. Spis ten nazwać można leksykonem; rozwiązanie takie zostało przyjęte i uzasadnione w dwu pracach Szpakowicza i Świdzińskiego (1981a, 1981b). Leksykon zawiera wszystkie elementy terminalne, ale również ich dystrybucyjne ekwiwalenty o dowolnym stopniu złożoności. Zadaniem leksykonu jest symulowanie analizy jednostek składniowych, które nie są szczegółowo definiowane.

Odpowiednik tak rozumianego leksykonu w programie *Świgra* praktycznie nie obejmuje konstrukcji złożonych (z wyjątkiem tzw. analitycznych form fleksyjnych). Uwzględnienie złożonych realizacji jednostek elementarnych GFJP w programie komputerowym wymagałoby rozbudowy reguł gramatyki i zapewne wprowadzenia jakiegoś nowego komponentu przetwarzającego frazeologizmy.

W powyższym cytacie kryje się pewna nieścisłość związana z segmentacją. Otóż niektóre reguły gramatyki odwołują się do segmentów będących częściami właściwymi pewnych słów (przykład takiej reguły przedstawiam w punkcie 4.7). Nie widać natomiast mechanizmu, który miałby podzielić owe słowa na segmenty.

Warto zwrócić uwagę, że Świdziński nie odwołuje się w ogóle do pojęcia leksemu. Słowom przypisywane są jedynie pewne opisy gramatyczne. Nie jest natomiast wykorzystywany fakt, że praktycznie oznacza to zebranie form w pewne zbiory. Tymczasem zinterpretowanie słów jako przynależnych do odpowiednich leksemów (i tym samym możliwość wskazania całego leksemu przez podanie jego identyfikatora) pozwala uprościć zapis niektórych reguł gramatycznych (por. 4.7). W programie *Świgra* korzystam z tej możliwości.

Świdziński przyjmuje, że gramatyka operuje na niezinterpretowanych słowach, które są pobierane z wejścia, a następnie ich opisy są pobierane ze słownika. Taka organizacja oznacza, że nie da się zróżnicować interpretacji morfologicznej słowa w zależności od kontekstu, w którym ono występuje. To zaś znaczy, że nie można by podać na wejście analizy wypowiedzenia w formie wstępnie ujednoznacznionej morfologicznie (np. za pomocą tzw. tagera, czyli programu kontekstowo ujednoznaczniającego wyniki analizy morfologicznej).

4.4. Komponent słownikowy programu *Świgra*

Wykorzystanie w programie *Świgra* analizatora morfologicznego *Morfeusz* sprawia, że naturalne jest operowanie w regułach pojęciem leksemu. Ponadto „alfabetem wejściowym”

analizy składniowej nie są niezinterpretowane słowa, ale formy wyrazowe. To zaś oznacza, że w różnych wypowiedzeniach lub różnych fragmentach tego samego wypowiedzenia ten sam segment może mieć różny zestaw interpretacji. Obecna wersja programu nie zawiera komponentu ujednoznaczniającego wyniki analizy morfologicznej, ale zawiera pewien drobny element analizy kontekstowej (por. p. 4.6).

Analizator *Morfeusz* jest programem napisanym przeze mnie na podstawie danych opracowanych przez prof. Z. Saloniego. Podstawowym źródłem wiedzy o fleksji polskiej jest dla programu nowe wydanie pracy Tokarskiego 2002. W programie wykorzystałem też dokładniejsze niż u Tokarskiego dane o odmianie czasowników z książki Saloniego 2001 (por. także Saloni i Woliński 2003). System znaczników stosowany w obecnej wersji programu jest najbliższy opisowi znakowania form zawartemu w moim artykule (2003) (wcześniejsze artykuły różnią się w szczegółach).

Założenia projektowe systemu znakowania form stosowanego w programie *Morfeusz* bazują na obserwacji, że tradycyjnie wyróżniane leksemy są jednostkami niejednorodnymi pod względem fleksyjnym. Na przykład przyjmuje się, że czasowniki to leksemy odmienne przez osobę. Faktycznie wielu formom w paradygmacie czasownika można przypisać wartość osoby, są jednak formy, które są na tę kategorię gramatyczną obojętne, na przykład bezokolicznik i bezosobnik.

W opisie formalnym poręcznie jest zebrać razem mniejsze niż leksem grupy form, które są jednorodne pod względem odmiany. Taką jednostkę będę nazywać fleksemem[3]. W tradycyjnym leksemie czasownikowym można wyróżnić między innymi odmienny przez liczbę i osobę fleksem nieprzeszły obejmujący formy teraźniejsze dla czasowników niedokonanych i przyszłe dla dokonanych, nieodmienny fleksem bezokolicznikowy i nieodmienny fleksem bezosobnikowy.

Morfeusz podaje dla każdego segmentu określenie fleksemu, którego formy dany segment może być wykładnikiem. Leksemy są więc w tym opisie reprezentowane pośrednio przez fakt, że fleksem jest częścią pewnego leksemu, oraz przez „formę podstawową", która jest identyfikatorem całego leksemu, a nie poszczególnych fleksemów. Na przykład dla fleksemów wchodzących w skład czasownika jest to wykładnik bezokolicznika.

W analizatorze *Morfeusz* staram się przypisywać formom wartości wszystkich kategorii gramatycznych potrzebnych do rozróżnienia segmentów różniących się dystrybucją (oczywiście chodzi wyłącznie o cechy fleksyjne i składniowe, a nie semantyczne). W opisie uwzględniam również cechy morfologiczne pomijane przez Świdzińskiego, np. kategorię poprzyimkowości, por. p. 4.6.

Zdarza się, że forma o danym opisie może być realizowana przez więcej niż jeden segment. Mówi się wtedy o wariantach swobodnych. *Morfeusz* przypisuje takim wariantom identyczną charakterystykę. Nie stanowi to problemu w przypadku analizy tekstu. Natomiast przy generacji form trzeba by wskazać programowi, który z wariantów ma wygenerować.

Morfeusz przypisuje każdemu segmentowi wszystkie możliwe interpretacje jako wykładnika formy wyrazowej. Każda interpretacja składa się z identyfikatora leksemu i znacznika wskazującego charakterystykę morfologiczną. Ściślej, należałoby mówić o charakterystyce morfosyntaktycznej, ponieważ niektóre z notowanych cech nie są cechami morfologicznymi

[3] Pojęcie i termin fleksemu zaproponował Bień (1991). Używana tutaj definicja różni się nieznacznie od oryginalnej.

(fleksyjnymi) danej klasy fleksemowej. Są natomiast istotne składniowo. Na przykład dla rzeczowników podawany jest rodzaj, mimo że rzeczowniki nie odmieniają się przez rodzaj.

Znacznik jest ciągiem wartości poszczególnych kategorii gramatycznych rozdzielonych dwukropkami. Na pierwszej pozycji jest umieszczona nazwa fleksemu. Na przykład segmentowi *chłopcem* przypisywany jest znacznik subst:sg:inst:m1. Oznacza on rzeczownik w liczbie pojedynczej w narzędniku, jest to rzeczownik męskoosobowy. Jeżeli danemu segmentowi można przypisać wiele znaczników (czyli zinterpretować go jako wiele form) stosowana jest, o ile to możliwe, notacja skrótowa. Na przykład dla segmentu *stół* zapis subst:sg:nom.acc:m3 oznacza alternatywę dwóch znaczników o wartości przypadka równej mianownikowi i biernikowi.

Zapis został zaprojektowany tak, aby był czytelny dla człowieka. Jednak w Prologu jeśli zdefiniuje się dwukropek jako operator infiksowy, znaczniki staną się poprawnym prologowym zapisem termów. I tak faktycznie są one wykorzystywane w programie *Świgra*. Operator dwukropek wiąże w prawo, więc w szczególności term subst:sg:_ oznacza formę rzeczownika w liczbie pojedynczej o nieustalonych pozostałych elementach charakterystyki, zaś subst:_:_:m1 dowolną formę rzeczownika męskoosobowego. Operator dwukropek wiąże słabiej niż operator kropka, dzięki czemu znaczniki z alternatywą odpowiadają właściwym termom.

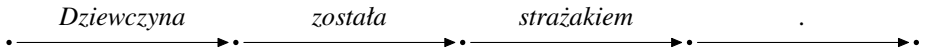
Dla konsekwencji również segmenty złożone ze znaków interpunkcyjnych mają przypisywany „identyfikator leksemu" (który jest równy segmentowi) i znacznik (interp).

4.5. Sposób reprezentacji analizowanego wypowiedzenia

W najprostszym przypadku analizowane wypowiedzenie ma postać listy. Na przykład dla wypowiedzenia

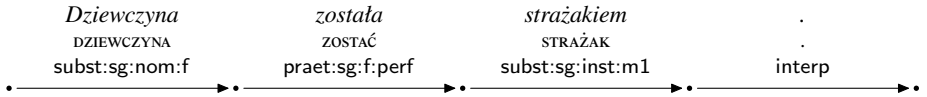
(21) *Dziewczyna została strażakiem.*

jest to lista następująca:



O liście tej będziemy myśleć jako o grafie skierowanym, którego wierzchołki reprezentują punkty pomiędzy interesującymi nas elementami wypowiedzenia, łuki natomiast reprezentują elementy listy.

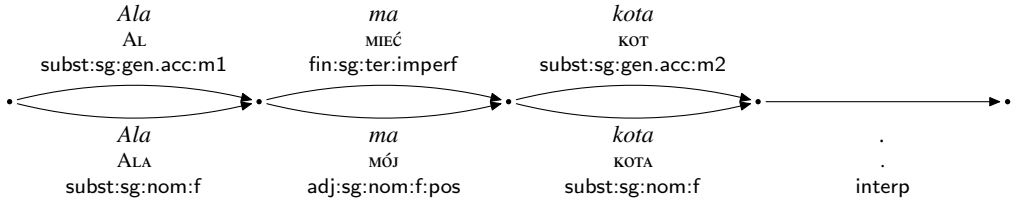
Ściślej rzecz biorąc, elementy te nie są słowami, ale formami wyrazowymi, są więc zinterpretowane jako przynależne do pewnych leksemów i opatrzone charakterystyką morfosyntaktyczną:



Jeżeli dany segment ma więcej niż jedną interpretację, pojawia się więcej łuków łączących dane węzły. Na przykład dla wypowiedzenia

(22) *Ala ma kota.*

graf wejścia ma postać:

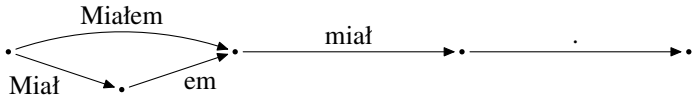


Zauważmy, że graf ten zawiera dwie ścieżki odpowiadające poprawnej składniowo interpretacji całego wypowiedzenia. Pierwsza głosi, że pewna Ala jest w posiadaniu kota. W drugiej pewnej kocie (punktowi wysokościowemu) jest przypisany pewien Al.

Ponadto podział na segmenty może być niejednoznaczny. Dla zdania

(23) *Miałem miał.*

graf przyjmuje postać następującą (dla przejrzystości pominięto interpretację segmentów):



Słowo *miałem* albo stanowi jeden segment o interpretacji rzeczownikowej, albo też rozpada się na dwa segmenty, z których pierwszy, *miał*, albo jest wykładnikiem pseudoimiesłowu czasownika MIEĆ, albo też formą rzeczownika MIAŁ. W tym ostatnim wypadku w wypowiedzeniu występuje nieciągły wariant szyku czasownika *-em miał*.

Graf, na którym pracuje analizator *Świgra*, może być jeszcze nieco bardziej skomplikowany w wyniku zadziałania dwóch mechanizmów: pomijania interpretacji poprzyimkowych (opisanego w punkcie 4.6) i dopisywania przecinków (punkt 5.6).

Jak napisałem w punkcie 2.3 i 2.4, w analizatorze *Świgra* omówiony graf jest reprezentowany klauzulowo, w postaci przedykatu input/3. Wierzchołki są opatrzone numerami (w sposób arbitralny). Każdy łuk grafu jest reprezentowany przez jedną klauzulę predykatu input. Argumentami tego predykatu są: wierzchołek początkowy i końcowy łuku, oraz reprezentacja jednej formy wyrazowej. Ma ona postać termu z funktorem morf/3, którego argumentami są segment, identyfikator leksemu i znacznik morfosyntaktyczny (w postaci termu). Oto przykład:

```
input(0, 1, morf('Miał', miał, subst:sg:nom.acc:m3)).
input(0, 1, morf('Miał', mieć, praet:sg:m1.m2.m3:imperf)).
input(1, 2, morf(em, być, aglt:sg:pri:imperf:wok)).
input(0, 2, morf('Miałem', miał, subst:sg:inst:m3)).
input(2, 3, morf(miał, miał, subst:sg:nom.acc:m3)).
input(2, 3, morf(miał, mieć, praet:sg:m1.m2.m3:imperf)).
input(3, 4, morf('.', '.', interp)).
```

4.6. Poprzyimkowość

Formom *niego* i *jego* zaimka ON można przypisać taką samą charakterystykę liczbowo-rodzajowo-przypadkową: obie opiszemy m.in. jako dopełniacz lub biernik rodzajów męskich.

Jednak formy te różnią się dystrybucją. Forma *niego* może wystąpić w tekście jedynie bezpośrednio poprzedzona przyimkiem.

(24) *Spóźniłem się przez niego.*
(25) *Lubię jego matkę.*

Podobną własność mają inne pary form zaimka ON, na przykład w rodzaju żeńskim: *nie* i *je*.

Cechę różniącą te formy Bień i Saloni (1982) nazywają kategorią poprzyimkowości. Niestety nie została ona uwzględniona w GFJP, w związku z czym następująca para wypowiedzeń, stanowiących w miarę naturalne odpowiedzi na pytanie *Co czytałeś?*, może otrzymać taką samą interpretację:

(26) *Nie czytałem.*
(27) *Je czytałem.*

W interpretacji tej *nie/je* jest biernikową formą zaimka *on* wymaganą przez czasownik *czytać*. Nieodróżnialność tych dwóch form powoduje dodanie błędnych interpretacji w wielu wypowiedzeniach, które zawierają partykułę przeczącą *nie* i czasownik, którego jednym z wymagań jest fraza nominalna w bierniku. Wypowiedzenia takie stanowią znaczący procent wszystkich wypowiedzeń z negacją.

Uwzględnienie poprzyimkowości w gramatyce nie jest trywialne, ponieważ bezpośrednio po przyimku nie zawsze musi stać forma poprzyimkowa zaimka. Porównajmy:

(28) *Spóźniłem się przez niego.*
(29) *Spóźniłem się przez jego matkę.*

Wydaje się, że formy poprzyimkowe są używane tylko, gdy same konstytuują frazę stojącą po przyimku. Być może są jednak możliwe wypowiedzenia typu następującego:

(30) ?*Spóźniłem się przez niego lub nią.*

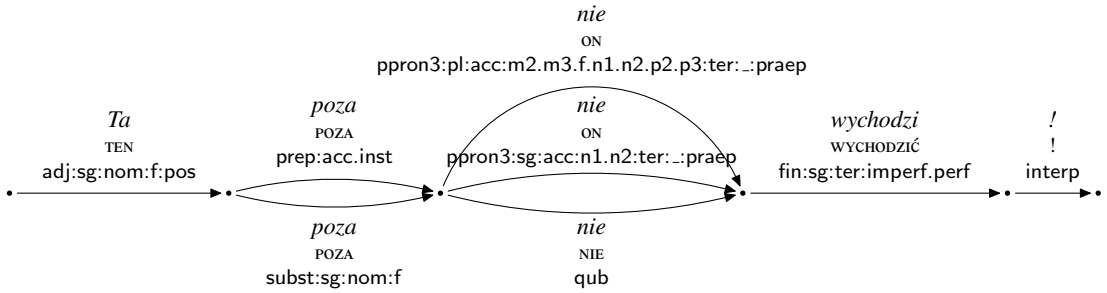
Uznałem, że odpowiednia rozbudowa reguł byłaby zbyt dużym odejściem od oryginalnej gramatyki Świdzińskiego. Żeby jednak zablokować liczne nadmiarowe interpretacje ze słowem *nie*, wprowadziłem mechanizm kontekstowego usuwania odpowiednich interpretacji z wyników analizy morfologicznej.

Mianowicie gdy w wynikach pojawia się forma poprzyimkowa zaimka, sprawdza się, czy poprzedzający segment może być zinterpretowany jako przyimek. Jeżeli nie, z grafu morfologicznego usuwa się łuk odpowiadający formie poprzyimkowej. Jeżeli tak, graf jest modyfikowany tak, aby utworzyć nową ścieżkę, co wymusza skojarzenie formy poprzyimkowej ze stojącym przed nią przyimkiem. Jeżeli jest jakaś inna interpretacja tych form, pozostaje ona w grafie.

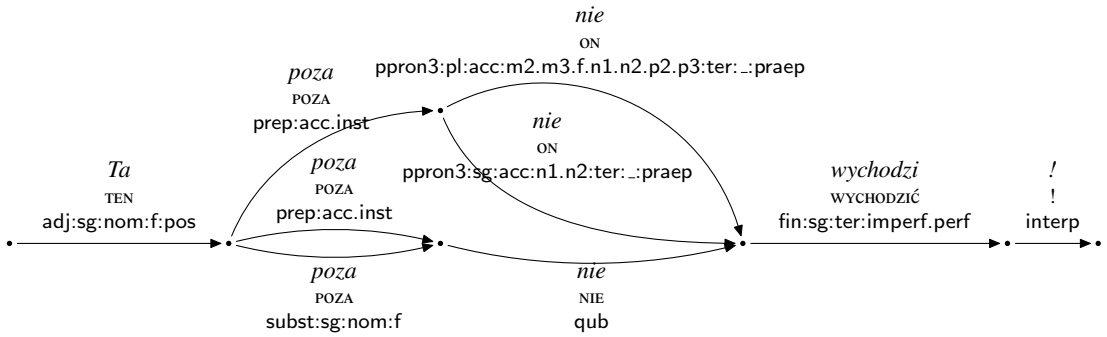
Rozważmy dla przykładu wypowiedzenie:

(31) *Ta poza nie wychodzi!*

Segment *poza* może być interpretowany jako wykładnik rzeczownika lub przyimka. Segment *nie* jako partykuła przecząca lub forma zaimka ON. Graf morfologiczny przed przekształceniem ma postać:



Po przekształceniu otrzymuje się graf następujący:



Warto zwrócić uwagę, że z grafu nie jest usuwana interpretacja *poza* jako przyimka w części grafu prowadzącej do *nie* interpretowanego jako partykuła. Dzieje się tak, ponieważ jest możliwa np. fraza *poza nie jednym, ale dwoma przypadkami*.

Zauważmy także, że uzyskany graf odpowiada automatowi niedeterministycznemu, z jednego z wierzchołków wychodzą dwie krawędzie etykietowane przyimkową interpretacją segmentu *poza*.

W wypadku tego akurat wypowiedzenia zarówno pierwotna jak i dodana część grafu zawiera ścieżki zawierające interpretacje morfologiczne wypowiedzenia możliwe składniowo a nawet semantycznie. Można je zasugerować umieszczając wypowiedzenie w takich kontekstach:

(32) *Pozostałe propozycje ledwie spełniają warunki przetargu. Ta poza nie wychodzi!*

(33) *Ćwiczę układ od 3 godzin i wciąż przewracam się po trzecim kroku. Ta poza nie wychodzi!*

Formy o ustalonej wartości niepoprzyimkowej poprzyimkowości nie podlegają specjalnemu przetwarzaniu. Jest bowiem możliwe ich wystąpienie również po przyimku (np. *przez ich matkę*).

4.7. Jednostki elementarne GFJP

W tym punkcie przedstawię definicje jednostek elementarnych GFJP, a więc zdefiniowanych regułami, których prawa strona zawiera elementy terminalne.

Gramatyka Świdzińskiego jest pisana z założeniem, że wypowiedzenie na wejściu jest ciągiem niezinterpretowanych słów. Oto przykład reguły w której występuje element terminalny:

zaimrzecz(F, P, Rl**)** $\longrightarrow$ (jel5)
  [F**]**,
  { slow(F, zaimrzecz, $P.Rl$) }.

Element F jest słowem pobranym z wejścia. Warunek slow/3 reprezentuje stowarzyszony z gramatyką słownik, przyporządkowujący słowu interpretację.

Przy reprezentacji wypowiedzenia przedstawionej w poprzednim punkcie postać tej reguły nieco się zmienia. Mianowicie elementem terminalnym jest term z funktorem morf. Pierwszym argumentem funktora morf jest segment, drugim — identyfikator leksemu. Oznaczenie klasy gramatycznej (tutaj zaimka osobowego) i komplet parametrów właściwych dla tej klasy stanowi trzeci argument funktora. Znika natomiast warunek slow.

zaimrzecz(H, P, R/L**)** $\longrightarrow$ (jel5)
  [morf(_, H, psubst:L:P:R)**]**.

Dla uproszczenia pomijam tu jedną komplikację. Otóż wartości kategorii gramatycznych stosowane w znacznikach analizatora morfologicznego *Morfeusz* są zapisywane na modłę „łacińsko-międzynarodową”, GFJP natomiast operuje skrótami polskich określeń. Konieczne jest więc tłumaczenie jednych oznaczeń na drugie. Tłumaczenie to odbywa się właśnie w regułach typu powyższej, ponieważ jest to naturalne miejsce, gdzie znaczniki morfosyntaktyczne są „rozkładane” na części. Dla poglądowości pomijam w przytaczanych regułach warunki odpowiedzialne za to tłumaczenie.

W wypadku większości kategorii tłumaczenie polega na wzajemnie jednoznacznej zamianie jednego symbolu atomowego na inny. Wartości kategorii rodzaju są zamieniane na zupełnie inną reprezentację wewnętrzną zapewniającą większą efektywność (czy wręcz poprawność) analizy. Zostało to omówione w punkcie 5.3.2 poświęconym parametrowi rodzaju. Wspomniane warunki są również odpowiedzialne za pobieranie wariantów interpretacji znaczników zapisanych skrótowo (z kropkami). Na przykład znacznik subst:sg:nom.acc:m3 jest w tym miejscu rozbijany na znaczniki subst:sg:nom:m3 i subst:sg:acc:m3.

W powyższej regule występuje wartość psubst jako oznaczenie klasy fleksemowej. Wartości takiej nie ma wśród znaczników analizatora *Morfeusz* — ten sklasyfikowałby zaimki rzeczowne jako rzeczowniki. Pochodzi ona z niewielkiego słownika leksemów, klasyfikującego jednostki funkcyjne GFJP. Dane w tym słowniku służą do zastępowania opisów dostarczanych przez *Morfeusz*a wartościami dostosowanymi specjalnie do wymagań GFJP.

Czytelnik zauważył też zapewne, że pierwszym parametrem jednostki **zaimrzecz** jest segment w oryginale i identyfikator leksemu w programie. Odstępstwo to wyjaśniam niżej.

Co ciekawe, wśród reguł opisujących jednostki elementarne GFJP jest następująca:

morfagl(F, Rl, O**)** $\longrightarrow$ (jel1)
  [F**]**,
  slow(F, morfagl, Rl, O**)**.

Definiuje ona morfem aglutynacyjny, a więc np. segment *śmy* występujący w dwusegmentowych słowach typu *gdybyśmy*. Niestety w książce brak opisu lub choćby wskazania na jakiś mechanizm, który rozbiłby słowo *gdybyśmy* na segmenty, które występują w regułach. W programie *Świgra* takie rozbicie zapewnia analizator morfologiczny (por. 4.5).

Zasada operowania na słowach (czy, jak się okazuje, segmentach) powoduje, że w GFJP są bloki reguł opisujących jednostki funkcyjne, w których wymieniono wszystkie formy fleksyjne pewnych leksemów. Dotyczy to na przykład definicji zaimka **zaimno** nieokreślonego przymiotnego *taki*, która w GFJP składa się z 13 reguł (zaim57)–(zaim69) bazujących na definicji zaimka przymiotnego **zaimprzym**:

zaimprzym(F, P, Rl**)** $\longrightarrow$ (jel6)
  [F**]**,
  { slow(F, zaimprzym, P.Rl) }.

zaimno(przym, mian, Rl, O, tk**)** $\longrightarrow$ (zaim57)
  zaimprzym(taki, mian, Rl**)**,
  { równe(Rl, (mos.poj).(mzw.poj).(mnz.poj)) }.

zaimno(przym, mian, zen.poj, O, tk**)** $\longrightarrow$ (zaim58)
  zaimprzym(taka, mian, żeń.poj**)**.

...

zaimno(przym, narz, Rl, O, tk**)** $\longrightarrow$ (zaim69)
  zaimprzym(takimi, narz, Rl**)**,
  { równe(Rl, (mos.mno).(mzw.mno).(mnz.mno).(żeń.mno).(nij.mno).(pt.mno)) }.

W programie ten blok reguł zastępują następujące dwie reguły:

zaimprzym(H, P, R/L**)** $\longrightarrow$ (jel6)
  [morf(_, H, padj:L:P:R:_)**]**.

zaimno(przym, P, Rl, _O, tk**)** $\longrightarrow$ (zaim57n)
  zaimprzym(taki, P, Rl**)**.

Zauważmy, że w związku z tym pierwszym parametrem jednostki **zaimprzym** musi być postać hasłowa leksemu a nie segment. Jednostka ta występuje wyłącznie w definicjach „funkcyjnych" jednostek zaimkowych — **zaimwzg**, **zaimpyt** i **zaimno**. Zmiana ta nie powoduje więc komplikacji w innych częściach gramatyki.

Podobnym uproszczeniom poddane zostały reguły: (zaim1–4), (zaim5–8) opisujące zaimki pytajne rzeczowne, (zaim9–21) zaimki pytajne przymiotne, (zaim23–26), (zaim27–30), (zaim31–43) zaimki względne rzeczowne, (zaim44–56) zaimki względne przymiotne (formy leksemu *jaki*), (kor3–6) korelaty właściwe.

4.7.1. Jednostki nie definiowane w GFJP

Według punktu 3.6 Aneksu do GFJP następujące jednostki nie są definiowane przez Świdzińskiego: forma czasownikowa **formaczas(**Wf, A, C, T, RL, O, Wa, Wb, Wc, K, Z**)**, forma przyimkowa **formaprzyim(**Pm, P**)**, forma rzeczownikowa **formarzecz(**P, RL**)**, forma przymiotnikowa **formaprzym(**P, RL, St**)**, forma przysłówkowa **formaprzysł(**St**)**. W istocie w regułach występują ponadto niezdefiniowane jednostki: **zaimos(**_, P, RL, O**)** — zaimek osobowy (trudno ustalić znaczenie pierwszego parametru, w jedynym wystąpieniu

tej jednostki ma on ustaloną wartość rzecz) oraz **zaimneg(***Kgz*, *P*, *RL*, *O*, *Kl***)** — zaimek negatywny.

Ponadto reguły opisujące zaimki nieokreślone **zaimno** uwzględniają jedynie zaimki przymiotne i przysłowne (pierwszy parametr o wartości przym i przysł), podczas gdy w regule (no43) opisującej konstrukcję nominalną występuje po prawej stronie jednostka **zaimno** z pierwszym argumentem rzecz (zaimek nieokreślony rzeczowny). Definicje jednostek **zaimneg** i **zaimno** uzupełniłem na podstawie korespondencji z Autorem GFJP. Te jednostki nie należą jednak do elementarnych — są one definiowane za pośrednictwem zaimka rzeczownego, przymiotnego i przysłownego.

4.7.2. Forma rzeczownikowa

Jednostka **formarzecz** może być realizowana przez formy wyrazowe zaklasyfikowane w programie *Morfeusz* jako przynależne do fleksemu rzeczownikowego (oznaczenie subst). Znacznik morfosyntaktyczny dla form tego fleksemu zawiera wartości kategorii liczby, przypadka i rodzaju. Wartości te stają się odpowiednimi argumentami jednostki **formarzecz**:

formarzecz(*P*, *R*/*L***)** ⟶ (n_rz1)
[morf(_, _, subst:*L*:*P*:*R***)]**.

(34) *Jacyś* ***studenci*** *świetnie się bawili.*
(35) *Widzę* ***fretkę****.*

Dodatkowo do paradygmatu rzeczowników rodzaju męskiego-osobowego m1 należą tzw. formy deprecjatywne, zgrupowane w programie *Morfeusz* we fleksem deprecjatywny depr. Mają one przypisany rodzaj męski-zwierzęcy, bo dzięki temu daje się prosto opisać ich zachowanie na poziomie składniowym (por. Woliński 2003).

formarzecz(*P*, *R*/*L***)** ⟶ (n_rz2)
[morf(_, _, depr:*L*:*P*:*R***)]**.

(36) *Jakieś* ***studenty*** *świetnie się bawiły.*

W GFJP gerundia (rzeczowniki odczasownikowe) nie są traktowane w żaden szczególny sposób. Dlatego trzeba dopuścić je jako realizacje formy rzeczownikowej. Gerundiom oprócz cech rzeczownikowych przysługuje wartość aspektu i zanegowania, które są ignorowane w poniższej regule.

formarzecz(*P*, *R*/*L***)** ⟶ (n_rz3)
[morf(_, _, ger:*L*:*P*:*R*:_**)]**.

(37) *Lubię* ***czytanie****, ale* ***nieczytanie*** *też sprawia mi przyjemność.*

4.7.3. Forma przymiotnikowa i przysłówkowa

Formy przymiotnikowe należą do fleksemu o oznaczeniu adj, formy przysłówkowe — do fleksemu adv. Formom tym przysługuje (między innymi) wartość stopnia. Przymiotniki i przysłówki niestopniowalne mają w tej pozycji oznaczenie stopnia równego. Wartość tego parametru nie bierze udziału w żadnych uzgodnieniach (por. GFJP s. 254).

formaprzym(P, R/L, St**)** $\longrightarrow$ (n_pt1)
[morf(_, _, adj:L:P:R:St)**]**.

(38) *Warto być **niewinnym**.*
(39) *On jest **najlepszym** przyjacielem Piotra.*

formaprzysl(St**)** $\longrightarrow$ (n_ps)
[morf(_, _, adv:St)**]**.

(40) *Jest **dobrze**.*

W definicji formy przymiotnikowej trzeba też uwzględnić „przymiotniki złożone" jak *biało-czerwony*. Są one realizowane przez ciąg form fleksemu przymiotnika przyprzymiotnikowego adja i pewną formę przymiotnikową, połączone łącznikiem:

formaprzym(P, R/L, St**)** $\longrightarrow$ (n_pt2)
[morf(_, _, adja)**]**,
[morf('-', '-', interp)**]**,
formaprzym(P, R/L, St**)**.

(41) *Powiewał **biało-czerwony** sztandar.*

W GFJP potraktowano imiesłowy przymiotnikowe jako przymiotniki. W związku z tym do definicji jednostki formaprzym należą jeszcze następujące dwie reguły:

formaprzym(P, R/L, row**)** $\longrightarrow$ (n_pt3)
[morf(_, _, ppas:L:P:R:_)**]**.

(42) *Odłożył **przeczytaną** książkę.*

formaprzym(P, R/L, row**)** $\longrightarrow$ (n_pt4)
[morf(_, _, pact:L:P:R:_)**]**.

(43) *Przyjdą **czytający** chłopcy.*

Oczywiście opis GFJP jest w tym miejscu wyraźnie zbyt mało subtelny, GFJP akceptuje na przykład oba poniższe wypowiedzenia:

(44) *Znam poświęconą Piotrowi książkę.*
(45) *$^{*!}$Znam zieloną Piotrowi książkę.*

Co gorsza, aby zanalizować następujące zdanie ilustrujące regułę (pt9), trzeba dodać możliwość realizacji formy przymiotnikowej przez parę słów *interesujący się*:

(46) *Kim interesujący się chłopcy przyjdą, nie wiadomo.*

Na potrzeby analizy zdań przykładowych Świdzińskiego dodałem następującą regułę, uważam ją jednak za rozwiązanie wyraźnie prowizoryczne i nie uwzględniam wariantów szyku ani analogicznej możliwości dla imiesłowów biernych. Fragment gramatyki dotyczący imiesłowów przymiotnikowych wymaga istotnego ulepszenia (por. Świdziński 2000). Dotyczy to również gerundiów.

formaprzym(P, R/L, row**)** $\longrightarrow$ (n_pt5)
[morf(_, _, pact:L:P:R:_)**]**,
[morf(_, 'się', qub)**]**.

4.7.4. Formy zaimkowe

Zaimki osobowe ze względu na różną charakterystykę fleksyjną zostały w analizatorze *Morfeusz* podzielone na dwa fleksemy: zaimkowy nietrzecioosobowy (ppron12, np. *ja, ciebie, nami, was*), i zaimkowy trzecioosobowy (ppron3, np. *on, nią, one*).

zaimos(P, R/L, O**)** $\longrightarrow$ (n_zo1)
[morf(_, _, ppron12:L:P:R:O)**]**.

zaimos(P, R/L, O**)** $\longrightarrow$ (n_zo2)
[morf(_, _, ppron3:L:P:R:O:_)**]**.

Zaimkom trzecioosobowym oprócz wartości liczby, przypadka, rodzaju i osoby przysługuje nietradycyjna kategoria poprzyimkowości (zob. p. 4.6), nie uwzględniona w GFJP.

Jednostki **zaimrzecz** — zaimek rzeczowny, **zaimprzym** — zaimek przymiotny i **zaimprzys** — zaimek przysłowny są zdefiniowane poprzez odwołanie do odpowiednich klas fleksemów:

zaimrzecz(H, P, R/L**)** $\longrightarrow$ (jel5)
[morf(_, H, psubst:L:P:R)**]**.

zaimprzym(H, P, R/L**)** $\longrightarrow$ (jel6)
[morf(_, H, padj:L:P:R:pos)**]**.

zaimprzys(H**)** $\longrightarrow$ (jel7)
[morf(_, H, padv)**]**.

Wymienione fleksemy nie występują w słowniku programu *Morfeusz*, pochodzą one z małego słowniczka klasyfikującego jednostki funkcyjne GFJP.

4.7.5. Forma czasownikowa

Forma czasownikowa **formaczas** to najbardziej skomplikowana i budzącą najwięcej wątpliwości jednostka elementarna w gramatyce. W oryginale jednostka ta ma 12 parametrów: **formaczas(**Wf, A, C, T, Rl, O, Wa, Wb, Wc, K, I, Z**)**. Są to kolejno: wyróżnik fleksyjny, aspekt, czas, tryb, rodzaj-liczba, osoba, trzy wymagania, korelatywność, inkorporacyjność i zależność. W programie Świgra liczba parametrów została zmniejszona do ośmiu: **formaczas(**Wf, A, C, T, Rl, O, Wym, K**)**.

Parametry Wa, Wb, Wc opisujące pewną kombinację wymagań zostały zastąpione jednym parametrem reprezentującym wszystkie kombinacje wymagań dopuszczalne dla danej formy. Przedstawiam to szczegółowo w punkcie 5.2.3. Parametr inkorporacyjności I został chyba wprowadzony przez pomyłkę w regułach (we30–35), w których przysługuje on definiowanej tam jednostce **kwer1** (konstrukcji werbalnej właściwej). Tymczasem we wszystkich użyciach jednostki **kwer1** (reguły (we27–29)) nie ma ona tego parametru. Nie powinno zatem go być i w jednostce **formaczas**. Zależność Z nie jest cechą słownikową czasownika, nie powinna zatem pojawić się wśród argumentów jednostki elementarnej **formaczas**. Wartości

zależności są nadawane/sprawdzane w regułach wyższego poziomu hierarchii (we30–35), opisujących konstrukcję werbalną właściwą.

Oto przykłady realizacji tekstowych jednostki **formaczas**, podane na stronie 233 GFJP: *będę czytać, byłybyście się dowiedziały, niech zostanie, powiedziano by, powiedzieć by* (ściślej mówiąc, w książce są one podane jako przykłady konstrukcji werbalnych właściwych, które są realizowane przez jedno wystąpienie jednostki **formaczas**). Zgodnie z wyjaśnieniami tam podanymi jednostka **formaczas** może być realizowana przez kilka słów tekstowych (czy też segmentów). Natomiast możliwości realizacji „formy czasownikowej", której segmenty nie sąsiadują ze sobą, Świdziński świadomie nie uwzględnia.

Reguły opisujące formę czasownikową muszą więc oprócz syntetycznych form czasownikowych uwzględniać konstrukcje analityczne z *się*, *niech*, *by*, nieciągły szyk czasu przeszłego i przyszłego złożonego. Ponadto na stronie 76 czytamy:

> Frazy werbalne (finitywne i niefinitywne) to frazy redukowalne do formy czasownika. Nie rozróżniam tu czasowników i czasowników niewłaściwych, jak w pracy Saloniego i Świdzińskiego (1987: r. IV).

Trzeba zatem uwzględnić również leksemy klasyfikowane przez *Morfeusz*a jako predykatywy (w zdaniach przykładowych występuje np. predykatyw WARTO, s. 380). Natomiast zanegowanie konstrukcji czasownikowej należy do innego poziomu, por. reguły (we20–24).

Jak wyjaśniam w punkcie 5.2.3, dla danego czasownika realizacje z *się* i bez *się* są opisane w osobnych pozycjach słownika wymagań czasownikowych. Można też na to patrzeć tak, że istnieją osobne leksemy np. *dać* i *dać się*. Dostęp do opisu wymagań danej formy czasownikowej daje predykat (warunek) slowczas/3. Jego argumentami są: identyfikator leksemu, określenie „sięności" (s — konstrukcja z *się*, n — bez *się*) i opis wymagań. Strukturę trzeciego argumentu wyjaśniam w punkcie 5.2.3.

Jednostkę **formaczas** zdefiniuję poprzez pomocniczą jednostkę **formaczas1**, którą można by nazwać formą czasownikową właściwą. Jednostka ta służy zmniejszeniu liczby reguł koniecznych do opisania różnych kombinacji szyku elementów czasownikowych z partykułą SIĘ. Jednostka ta ma o jeden parametr więcej niż **formaczas**. Parametr ten ma dwie możliwe wartości wyrażające fakt, czy dana jednostka musi wystąpić w kontekście partykuły *się*.

Wartość ustalona s pierwszego argumentu jednostki **formaczas1** oznacza, że jednostka ta ma niezrealizowane „wymaganie *się*", czyli że aby mogła zostać uznana za pełnoprawną formę czasownikową musi wystąpić w kontekście partykuły *się*. Brak takiego „wymagania *się*" sygnalizuje wartość ustalona n. Oznacza ona, że dana jednostka **formaczas1** opisuje formę czasownika bez *się*, lub że wymaganie *się* zostało już wypełnione w obrębie tej jednostki.

Jednostka **formaczas** może być reprezentowana albo przez pojedyncze wystąpienie jednostki **formaczas1**, albo przez takie wystąpienie w kontekście partykuły *się*:

formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)** ⟶ (n_cz1)
  formaczas1(n, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)**.

formaczas1(n, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)** ⟶ (n_cz2)
  formaczas1(s, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)**,
  [morf('się', 'się', _)**]**.

formaczas1(n, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)** ⟶ (n_cz3)
[morf('się', 'się', _)**]**,
formaczas1(s, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)**.

(47) ***Czytałybyście*** *książkę.*
(48) *Jan* ***bałby się.***
(49) *Dobrze* ***się czytało.***

Reguły opisujące doczepianie *się* przed lub po formie czasownikowej stanowią rozwiązanie tymczasowe, wobec braku precyzyjnego opisu „wymagania *się*" w GFJP (por. s. 233, p. 7.2.6).

Oczywiście tworzone w ten sposób poddrzewa nie odzwierciedlają żadnej struktury lingwistycznej. Wprowadzenie jednostki **formaczas1** traktuję jako zabieg czysto techniczny. Dotyczy to również pozostałych jednostek używanych w obrębie realizacji formy czasownikowej.

Tryb oznajmujący

Poniższa reguła opisuje formy fleksemu nieprzeszłego fin. Formy te, o ile należą do czasownika niedokonanego, charakteryzujemy jako teraźniejsze; dla czasownika dokonanego — jako przyszłe. Odpowiednią zależność wartości parametrów czasu *C* i aspektu *A* wyraża pomocniczy warunek czas/2.

W tej regule po raz pierwszy mamy do czynienia z predykatem slowczas/3. Jak widać, pierwszy argument jednostki **formaczas1** jest drugim argumentem predykatu slowczas. Oznacza to, że jeżeli ze słownika wymagań zostanie pobrany opis dla realizacji z *się*, czyli z wartością s tego argumentu, jednostce **formaczas1** zostanie przypisane „wymaganie *się*". Jeżeli zaś pobrany zostanie opis z wartością n, wynikowa jednostka nie będzie dopuszczała połączenia z *się*.

czas(nd, ter).

czas(dk, przy).

formaczas1(*S*, os, *A*, *C*, ozn, _*R*/*L*, *O*, *Wym*, _*K***)** ⟶ (n_cz4)
[morf(_, *H*, fin:*L*:*O*:*A*)**]**,
{ czas(*A*, *C*),
slowczas(*H*, *S*, *Wym*) }.

(50) ***Czytacie*** *książkę.*
(51) ***Przeczytacie*** *książkę.*
(52) ***Jestem*** *człowiekiem.*

Osobnej reguły wymagają formy czasu przyszłego czasownika BYĆ. Jako jedyny ma on zarówno formy czasu teraźniejszego jak i przyszłego (prostego). Te pierwsze opisuje reguła (n_cz4) (bo jest to czasownik niedokonany), te drugie — reguła następująca:

formaczas1(*S*, os, nd, przy, ozn, _*R*/*L*, *O*, *Wym*, _*K***)** ⟶ (n_cz5)
[morf(_, być, bedzie:*L*:*O*:nd)**]**,
{ slowczas(być, *S*, *Wym*) }.

(53) ***Będę*** *przeklęty.*

Do opisania „czasu przyszłego złożonego” posłużymy się pomocniczą jednostką **przyzlo**. Może ona być realizowana przez formę bezokolicznika (i wtedy ma nieustalony rodzaj-liczbę) lub przez pseudoimiesłów (nieaglutynacyjny, por. reguły dla czasu przeszłego):

przyzlo(S, _RL, Wym, _K**)** $\longrightarrow$ (n_czp1)
[morf(_, H, inf:nd)**]**,
{ slowczas(H, S, Wym) }.

przyzlo(S, R/L, Wym, _K**)** $\longrightarrow$ (n_czp2)
[morf(_, H, praet:L:R:nd:nagl)**]**,
{ slowczas(H, S, Wym) }.

„Czas przyszły złożony” jest wyrażany przez jednostkę **przyzlo** i formę fleksemu bedzie w dowolnej kolejności, między którymi może wystąpić partykuła *się*. Tę możliwość trzeba uwzględnić osobno, ponieważ reguły (n_cz2) i (n_cz3) opisują tylko doczepienie *się* przed całą konstrukcją lub po niej.

formaczas1(S, os, nd, przy, ozn, R/L, O, Wym, _K**)** $\longrightarrow$ (n_cz6)
[morf(_, być, bedzie:L:O:nd)**]**,
przyzlo(S, R/L, Wym, K**)**.

(54) *Całą zimę* ***będę jeździć/jeździł*** *na rowerze.*
(55) *Często się* ***będziesz zastanawiać/zastanawiała****.*

formaczas1(S, os, nd, przy, ozn, R/L, O, Wym, _K**)** $\longrightarrow$ (n_cz7)
przyzlo(S, R/L, Wym, K**)**,
[morf(_, być, bedzie:L:O:nd)**]**.

(56) ***Jeździć będę*** *całą zimę na rowerze.*

formaczas1(n, os, nd, przy, ozn, R/L, O, Wym, _K**)** $\longrightarrow$ (n_cz8)
[morf(_, być, bedzie:L:O:nd)**]**,
[morf('się', 'się', _)**]**,
przyzlo(s, R/L, Wym, K**)**.

(57) ***Będziecie się bały*** *przyjść same.*

formaczas1(n, os, nd, przy, ozn, R/L, O, Wym, _K**)** $\longrightarrow$ (n_cz9)
przyzlo(s, R/L, Wym, K**)**,
[morf('się', 'się', _)**]**,
[morf(_, być, bedzie:L:O:nd)**]**.

(58) ***Bać się będę*** *niewyobrażalnie.*

W regułach (n_cz8) i (n_cz9) pierwszy parametr ma wartość ustaloną n, ponieważ „wymaganie *się*” jest już wypełnione wewnątrz konstrukcji. Natomiast pierwszy argument jednostki **przyzlo** ma wartość s aby taka konstrukcja była dopuszczalna tylko dla realizacji dopuszczonych w słowniku wymagań.

Formy czasu przeszłego są realizowane przez pseudoimiesłów praet dla trzeciej osoby i przez pseudoimiesłów z aglutynacyjną formą być dla osoby pierwszej i drugiej. Dla nie-

których czasowników pseudoimiesłów ma różne formy stosowane w tych dwóch sytuacjach. Tę pierwszą nazywamy nieaglutynacyjną, drugą — aglutynacyjną, por. przykłady.

formaczas1(*S*, os, *A*, prze, ozn, *R*/*L*, 3, *Wym*, _*K***)** ⟶ (n_cz10)
[morf(_, *H*, praet:*L*:*R*:*A*:nagl)**]**,
{ slowczas(*H*, *S*, *Wym*) }.

(59) *Piotr* ***czytał****.*
(60) *Jan* ***zagniótł*** *ciasto.*

formaczas1(*S*, os, *A*, prze, ozn, *R*/*L*, *O*, *Wym*, _*K***)** ⟶ (n_cz11)
[morf(_, *H*, praet:*L*:*R*:*A*:agl)**]**,
{ slowczas(*H*, *S*, *Wym*) },
[morf(_*F*, 'być', aglt:*L*:*O*:_:_)**]**.

(61) ***Czytał·eś****.*
(62) ***Zagniotła·m*** *ciasto.*

(Kropką oznaczono wewnątrzsłowowe granice segmentów).

Zauważmy, że w tej ostatniej regule trzeba rozbić wartość rodzaju-liczby, ponieważ następuje tu uzgodnienie liczby, natomiast aglutynant nie niesie w sobie specyfikacji rodzaju — dopuszcza dowolny rodzaj.

Tryb warunkowy

Tryb warunkowy opisuję za pomocą jednostki **condaglt** (aglutynant warunkowy). Jej wprowadzenie pozwala zmniejszyć liczbę dalszych reguł do dwóch. Jednostka ta opisuje twory takie jak *by*, *bym*, *byście* a więc połączenia partykuły (kublika) BY z aglutynantem.

condaglt(*L*, 3**)** ⟶ (n_cza1)
[morf(by, by, qub)**]**.

condaglt(*L*, *O***)** ⟶ (n_cza2)
[morf(by, by, qub)**]**,
[morf(_*F*, 'być', aglt:*L*:*O*:_:_)**]**.

Konstrukcja warunkowa składa się z pseudoimiesłowu i aglutynantu warunkowego w dowolnej kolejności:

formaczas1(*S*, os, *A*, _*C*, war, *R*/*L*, *O*, *Wym*, _*K***)** ⟶ (n_cz12)
[morf(_, *H*, praet:*L*:*R*:*A*:nagl)**]**,
{ slowczas(*H*, *S*, *Wym*) },
condaglt(*L*, *O***)**.

(63) ***Czytałbym*** *książkę.*

formaczas1(*S*, os, *A*, _*C*, war, *R*/*L*, *O*, *Wym*, _*K***)** ⟶ (n_cz13)
condaglt(*L*, *O***)**,
[morf(_, *H*, praet:*L*:*R*:*A*:nagl)**]**,
{ slowczas(*H*, *S*, *Wym*) }.

(64) *Chętnie* ***byśmy czytali*** *książkę.*

W pierwszym wariancie szyku między tymi elementami nie ma odstępu, różnica ta jednak nie uwidacznia się na tym poziomie opisu, ponieważ tutaj mamy już do czynienia z tekstem podzielonym na segmenty.

Może się też zdarzyć, że pomiędzy tymi elementami wystąpi *się*.

formaczas1(n, os, *A*, _*C*, war, *R*/*L*, *O*, *Wym*, _*K*) ⟶ (n_cz14)
[morf(_, *H*, praet:*L*:*R*:*A*:nagl)**]**,
[morf('się', 'się', _)**]**,
{ slowczas(*H*, s, *Wym*) },
condaglt(*L*, *O***)**.

formaczas1(n, os, *A*, _*C*, war, *R*/*L*, *O*, *Wym*, _*K*) ⟶ (n_cz15)
condaglt(*L*, *O***)**,
[morf('się', 'się', _)**]**,
[morf(_, *H*, praet:*L*:*R*:*A*:nagl)**]**,
{ slowczas(*H*, s, *Wym*) }.

Tryb rozkazujący

Formy syntetyczne trybu rozkazującego są realizowane przez formy fleksemu rozkazującego impt.

formaczas1(*S*, os, *A*, przy, roz, _*R*/*L*, *O*, *Wym*, _*K*) ⟶ (n_cz16)
[morf(_, *H*, impt:*L*:*O*:*A*)**]**,
{ slowczas(*H*, *S*, *Wym*) }.

(65) ***Czytaj*** *książkę.*
(66) ***Czytajmy*** *książkę.*
(67) ***Czytajcie*** *książkę.*

Ponadto możliwe jest wyrażenie tej konstrukcji z pomocą słowa NIECH i formy fleksemu nieprzeszłego, ale tylko w niektórych kombinacjach osoby i liczby:

formaczas1(*S*, os, *A*, przy, roz, _*R*/*L*, *O*, *Wym*, _*K*) ⟶ (n_cz17)
[morf(_, niech, qub)**]**,
[morf(_, *H*, fin:*L*:*O*:*A*)**]**,
{ (*L* = poj, *O* \= 2; *L* = mno, *O* = 3),
slowczas(*H*, *S*, *Wym*) }.

(68) ***Niech skonam****!*
(69) ***Niech czyta*** *książkę.*
(70) ***Niech czytają*** *książkę.*

formaczas1(n, os, *A*, przy, roz, _*R*/*L*, *O*, *Wym*, _*K*) ⟶ (n_cz18)
[morf(_, niech, qub)**]**,
[morf('się', 'się', _)**]**,
[morf(_, *H*, fin:*L*:*O*:*A*)**]**,
{ (*L* = poj, *O* \= 2; *L* = mno, *O* = 3),
slowczas(*H*, s, *Wym*) }.

(71) ***Niech się boi****!*

Formy nieosobowe

W regułach opisujących formy nieosobowe czasowników stosowany jest dodatkowy warunek wykluczpodmiot/2, który służy do wyeliminowania z listy wymagań wymagania podmiotu (frazy nominalnej w mianowniku, np(mian)), której takie formy systemowo nie dopuszczają. Argumentami predykatu wykluczpodmiot jest lista wymagań i odpowiadająca jej lista wymagań z wyeliminowanym wymaganiem podmiotu.

Oto realizacje bezosobnikowe form czasownikowych:

formaczas1(*S*, bos, *A*, prze, ozn, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz19)
[morf(_, *H*, imps:*A*)**]**,
{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(*S*, bos, *A*, _*C*, war, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz20)
[morf(_, *H*, imps:*A*)**]**,
[morf(by, by, qub)**]**,
{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(*S*, bos, *A*, _*C*, war, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz21)
[morf(by, by, qub)**]**,
[morf(_, *H*, imps:*A*)**]**,
{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(n, bos, *A*, _*C*, war, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz22)
[morf(_, *H*, imps:*A*)**]**,
[morf('się', 'się', _)**]**,
[morf(by, by, qub)**]**,
{ slowczas(*H*, s, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(n, bos, *A*, _*C*, war, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz23)
[morf(by, by, qub)**]**,
[morf('się', 'się', _)**]**,
[morf(_, *H*, imps:*A*)**]**,
{ slowczas(*H*, s, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

(72) ***Czytano*** *książkę.*
(73) ***Czytano by*** *książkę.*
(74) *Chętnie* ***by czytano*** *książkę.*
(75) ***Bano się by*** *go jak ognia.*
(76) *Pewnie* ***by się bano*** *go jak ognia.*

A to realizacja bezokolicznikowa i imiesłowowe:

formaczas1(*S*, bok, *A*, _*C*, _*T*, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz24)
[morf(_, *H*, inf:*A*)**]**,

{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(*S*, psu, *A*, _*C*, _*T*, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz25)
[morf(_, *H*, pant:*A***)]**,
{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

formaczas1(*S*, psw, *A*, _*C*, _*T*, _*RL*, _*O*, *NWym*, _*K***)** ⟶ (n_cz26)
[morf(_, *H*, pcon:*A***)]**,
{ slowczas(*H*, *S*, *Wym*),
wykluczpodmiot(*Wym*, *NWym*) }.

(77) *Lubię* ***czytać****.*
(78) ***Przeczytawszy*** *książkę, przyszła.*
(79) *Przyszła* ***czytając*** *książkę.*

Czasowniki niewłaściwe (predykatywy)

Skąpe wzmianki w GFJP o czasownikach niewłaściwych nie klarują, jakie wartości parametrów powinna mieć forma czasownikowa realizowana przez czasownik niewłaściwy. Przyjmuję wyróżnik fleksyjny równy os aby oddać finitywność tych form.

W regułach opisujących czasowniki niewłaściwe nie ma potrzeby użycia predykatu wykluczpodmiot, ponieważ żadne formy tych czasowników nie dopuszczają podmiotu i fakt ten jest odnotowany w słowniku wymagań.

W związku z niedopuszczaniem podmiotu nie są istotne wartości parametru rodzaju-liczby i osoby. Wartości te pozostawiam nieustalone.

Jedyna forma predykatywu, występująca samodzielnie, odpowiada czasowi teraźniejszemu trybu oznajmującego:

formaczas1(*S*, os, nd, ter, ozn, _*RL*, _*O*, *Wym*, _*K***)** ⟶ (n_cz27)
[morf(_, *H*, pred**)]**,
{ slowczas(*H*, *S*, *Wym*) }.

(80) ***Warto*** *czytać książki.*

Analityczny czas przyszły powstaje przez zestawienie z formą *będzie* fleksemu bedzie:

formaczas1(*S*, os, nd, przy, ozn, _*RL*, _*O*, *Wym*, _*K***)** ⟶ (n_cz28)
[morf(będzie, być, bedzie:sg:ter:imperf**)]**,
[morf(_, *H*, pred**)]**,
{ slowczas(*H*, *S*, *Wym*) }.

formaczas1(*S*, os, nd, przy, ozn, _*RL*, _*O*, *Wym*, _*K***)** ⟶ (n_cz29)
[morf(_, *H*, pred**)]**,
[morf(będzie, być, bedzie:sg:ter:imperf**)]**,
{ slowczas(*H*, *S*, *Wym*) }.

(81) ***Będzie warto*** *przeczytać tę książkę.*
(82) ***Można będzie*** *przeczytać tę książkę.*

Analityczny czas przeszły zawiera formę *było* pseudoimiesłowu czasownika być:

formaczas1(S, os, nd, prze, ozn, _RL, _O, Wym, _K**)** $\longrightarrow$ (n_cz30)
[morf(było, być, praet:sg:_:imperf)**]**,
[morf(_, H, pred)**]**,
{ slowczas(H, S, Wym) }.

formaczas1(S, os, nd, prze, ozn, _RL, _O, Wym, _K**)** $\longrightarrow$ (n_cz31)
[morf(_, H, pred)**]**,
[morf(było, być, praet:sg:_:imperf)**]**,
{ slowczas(H, S, Wym) }.

(83) ***Było warto*** *to zobaczyć.*

Analityczny tryb warunkowy powstaje przez zestawienie z formą *by* (pisaną rozdzielnie w obu wariantach szyku):

formaczas1(S, os, nd, _C, war, _RL, _O, Wym, _K**)** $\longrightarrow$ (n_cz32)
[morf(_, H, pred)**]**,
[morf(by, by, qub)**]**,
{ slowczas(H, S, Wym) }.

formaczas1(S, os, nd, _C, war, _RL, _O, Wym, _K**)** $\longrightarrow$ (n_cz33)
[morf(by, by, qub)**]**,
[morf(_, H, pred)**]**,
{ slowczas(H, S, Wym) }.

(84) ***Warto by*** *posłuchać.*

Wreszcie jedyna forma niefinitywna predykatywów to analityczny bezokolicznik, pojawiający się na przykład w zdaniach:

(85) *Piotra musi* ***być stać*** *na samochód!*
(86) *Może* ***być warto*** *czytać książki.*

Uwzględniam tylko szyk z *być* z przodu — odwrotny szyk wydaje się tak nienaturalny, że chyba nie jest rozpoznawany przez użytkowników języka jako ta konstrukcja.

formaczas1(S, bok, nd, _C, _T, _RL, _O, Wym, _K**)** $\longrightarrow$ (n_cz34)
[morf(być, być, inf:imperf)**]**,
[morf(_, H, pred)**]**,
{ slowczas(H, S, Wym) }.

5. Komputerowa realizacja GFJP

W tym rozdziale opisuję te mechanizmy GFJP, których przy realizacji komputerowej nie można było pozostawić w postaci niezmienionej. Ponieważ naczelnym celem tej pracy jest realizacja GFJP w formie możliwie bliskiej oryginału, zmiany wprowadzałem tylko tam, gdzie to konieczne. Takich konieczności było kilka: pewne mechanizmy nie mieszczą się w formalizmie gramatyk metamorficznych (lewostronny kontekst, przy przyjętej strategii — reguły o pustej prawej stronie). Inne mechanizmy gramatyki mają tak duży negatywny wpływ na efektywność analizy, że konieczne było ich przeformułowanie (chodzi tu na przykład o analizę składników zdania elementarnego). W pewnych wreszcie miejscach zmiany były konieczne, aby uzyskać poprawność analizy — niektóre warunki są w oryginale umieszczone w takim miejscu, że program nie może znać wartości ich argumentów. W takich sytuacjach konieczne było przeniesienie warunków i odpowiednie przetransmitowanie ich argumentów.

Nie opisuję tutaj zjawisk które „liczą się tak, jak w książce". Na przykład nie przedstawiam hierarchii jednostek składniowych i nie piszę o pionowych i poziomych uzgodnieniach parametrów, bo w programie są one zrealizowane dokładnie tak, jak w GFJP. Nie przedstawiam też wszystkich mechanizmów sterujących analizą w GFJP — można o nich wyczytać w książce Świdzińskiego.

Rozdział ten ma z konieczności charakter dosyć techniczny i szczegółowy, jego lektura wymaga pewnego zaznajomienia z regułami GFJP. Podpunkty tego rozdziału nie mają jakiegoś wyraźnego naturalnego porządku i są od siebie w miarę niezależne. Przy realizacji programu dopiero po uwzględnieniu wszystkich omawianych tu zjawisk program zaczął produkować wyniki zgodne z oczekiwaniami.

Zmiany i uzupełnienia w regułach opisujących jednostki elementarne gramatyki, jako mające trochę inny charakter „domykania morfologicznego" opisu Świdzińskiego, omówiłem w punkcie 4.7.

Niektóre z omawianych tu zmian mogłyby być wprowadzone bardziej elegancko, gdyby ich wprowadzeniu towarzyszyło rozszerzenie stosowanego formalizmu gramatycznego. Jednak zastosowane tu pomysły mają często charakter tymczasowych poprawek i powinny raczej być zastąpione odpowiednimi zmianami w regułach gramatyki (sygnalizuję to w tekście). Uznałem, że dalej idące zmiany w mechanizmach GFJP nie należą do zakresu tej pracy, że lepiej będzie rozważyć je po bliższym przyjrzeniu się działającej gramatyce w jej obecnej postaci.

Pewne zmiany sprawiają, że drzewa uzyskane z programu różnią się od tych opisywanych przez reguły Świdzińskiego. Jednak wyniki prezentowane przez program powinny być możliwie bliskie GFJP. Dlatego są one poddawane kilku przekształceniom. Omawiam je odpowiednio w punktach opisujących zmiany. Podstawowym mechanizmem odtwarzania właściwej formy wyników jest prologowy predykat `portray/1` pozwalający przechwycić wypisywanie danego rodzaju termów.

Nie wszystkie różnice są ukrywane przed czytelnikiem wynikowych drzew. Dotyczy to takich sytuacji, gdy drzewa produkowane przez program są bardziej czytelne niż oryginalne, lub gdy dany element oryginału wydawał się być wprowadzony z powodów czysto technicznych, nie odpowiadających zjawiskom lingwistycznym (na przykład puste frazy wymagane symbolizujące brak wymagań). Pewne zmiany pozostały także ze względów technicznych. Na przykład aby drzewa wynikowe dały się reprezentować naturalnie w postaci termów prologowych, znak prim w wartościach parametrów został zastąpiony literą x (która nie występowała w regułach gramatyki). Pełną listę różnic między drzewami, jakie uzyskałoby się stosując oryginalne reguły GFJP, a tymi produkowanymi przez program można znaleźć w dodatku B.

5.1. Problem rekurencji

Sporo kłopotu przy komputerowej realizacji GFJP sprawia sposób potraktowania przez Autora rekurencji. Jak pisze Świdziński (GFJP, s. 59):

> Aby gramatyka formalna, której zbiór produkcji jest skończony, zdawać mogła sprawę z nieograniczonej liczby wyrażeń, musi zawierać przynajmniej jedną regułę rekurencyjną. Jest to zrozumiałe. Każda konstrukcja składniowa danego poziomu (wyjątki są nieliczne) może zawierać jako swój składnik bezpośredni inną konstrukcję tego samego poziomu, a nawet poziomów wyższych. Ponieważ nie do pomyślenia jest hierarchia o nieograniczonej liczbie poziomów, dla zapewnienia rekurencji dopuścić trzeba realizację jednostki składniowej poziomu najniższego (najprostszej) przez jednostkę poziomu najwyższego.

Na mocy tego rozumowania do GFJP zostało wprowadzonych pięć rozłącznych cykli złożonych z reguł o lewej i prawej stronie składającej się z pojedynczego nieterminala. Oto jednostki biorące udział w tych cyklach:

$$\mathbf{zr} \longrightarrow \mathbf{zsz} \longrightarrow \mathbf{zj} \longrightarrow \mathbf{zp} \longrightarrow \mathbf{ze} \longrightarrow \mathbf{zr}$$

$$\mathbf{fno} \longrightarrow \mathbf{knodop} \longrightarrow \mathbf{knopm} \longrightarrow \mathbf{knoatr} \longrightarrow \mathbf{knoink} \longrightarrow \mathbf{knom} \longrightarrow \mathbf{fno}$$

$$\mathbf{fps} \longrightarrow \mathbf{kpspm} \longrightarrow \mathbf{kpsps} \longrightarrow \mathbf{kpsink} \longrightarrow \mathbf{kprzysl} \longrightarrow \mathbf{fps}$$

$$\mathbf{fpt} \longrightarrow \mathbf{kptno} \longrightarrow \mathbf{kptpm} \longrightarrow \mathbf{kptps} \longrightarrow \mathbf{kptink} \longrightarrow \mathbf{kprzym} \longrightarrow \mathbf{fpt}$$

$$\mathbf{fzd} \longrightarrow \mathbf{fzdsz} \longrightarrow \mathbf{fzdj} \longrightarrow \mathbf{fzdkor} \longrightarrow \mathbf{fzd}$$

Koncepcja ta została przejęta z pracy Szpakowicza[1]. Tymczasem takie jałowe cykle prowadzą do nieskończonej rekursji niezależnie od przyjętej strategii analizy. Jeśli bowiem jakieś zdanie może być zinterpretowane jako wystąpienie nieterminala **zr**, to może także być zinterpretowane jako **ze**, które po kolejnych krokach jest interpretowane jako **zr** itd. Można dobudowywać dowolnie długie gałęzi w drzewie rozbioru wypowiedzenia. Szpakowicz zdaje sobie sprawę z problemu i pisze o konieczności zablokowania cykli.

[1] Szpakowicz jednak zakładał, że ze względu na słabą moc obliczeniową ówczesnych komputerów, przy realizacji komputerowej nie uwzględni reguł zamykających cykle. Ponadto w ówczesnych interpreterach języka Prolog standardowo był dostępny predykat ANCETRE, za pomocą którego można było badać stos wywołań procedur i tym samym łatwo blokować pewne formy rekursji. Predykat ten zniknął z obecnych wersji języka.

Dopuszczenie cyklu jednostek nieterminalnych, które można przepisywać jedna na drugą sprawia, że dana gramatyka przestaje mieć sens jako gramatyka bezkontekstowa czy metamorficzna. Zwykły sposób interpretacji prowadzi bowiem do wygenerowania dla każdego zdania, które ma jakąś interpretację, nieskończenie wielu innych trywialnych interpretacji.

Dyskusja problemu jałowej rekurencji

O ile z pierwszymi dwoma zdaniami przytoczonego fragmentu nie sposób się nie zgodzić, to dalsze rozumowanie budzi wątpliwości. Rozważmy następującą prostą gramatykę, sformułowaną analogicznie do hierarchii jednostek zdaniowych GFJP[2]:

zr ⟶ **zp**.

zr ⟶ **zp**, **[**i**]**, **zp**.

zp ⟶ **ze**.

zp ⟶ **ze**, **[**gdy**]**, **ze**.

ze ⟶ **[**'Jan', umarł**]**.

ze ⟶ **[**'Maria', spała**]**.

ze ⟶ **[**'Piotr', czytał**]**.

ze ⟶ **zr**.

Występuje w niej cykl złożony z jednostek **zr**, **zp** i **ze**. Dla porównania rozważmy następującą gramatykę nie zawierającą cyklu:

zr ⟶ **zp**.

zr ⟶ **zr**, **[**i**]**, **zr**.

zp ⟶ **ze**.

zp ⟶ **zr**, **[**gdy**]**, **zr**.

ze ⟶ **[**'Jan', umarł**]**.

ze ⟶ **[**'Maria', spała**]**.

ze ⟶ **[**'Piotr', czytał**]**.

Gramatyka ta opisuje nieskończenie wiele wypowiedzeń (np. wypowiedzenia typu *Jan umarł gdy Maria spała gdy Piotr czytał gdy Maria spała...*). Hierarchia jednostek składniowych jest skończona — ma trzy poziomy. Dopuszcza jednak dowolny stopień zagnieżdżenia. Akceptowane jest zdanie *Jan umarł i Maria spała gdy Piotr czytał* z dwiema interpretacjami — albo *i* góruje nad *gdy*, albo na odwrót. Wreszcie wypowiedzenia interpretowane według tej gramatyki otrzymują takie same struktury jak w przypadku gramatyki poprzedniej, w tym sensie że otrzymane drzewa mają takie same rozgałęzienia. A jednak w gramatyce tej jednostka poziomu najniższego (**ze**) nie może być realizowana przez jednostkę poziomu najwyższego (**zr**).

Wydaje się zatem, że nie jest prawdziwy pogląd wyrażony w zacytowanym wcześniej fragmencie, a następujące jego przetworzenie (s. 76) jest warunkiem wystarczającym ale nie koniecznym:

[2] Dla uproszczenia całkowicie zaniedbujemy znaki interpunkcyjne.

Zdanie elementarne może być w szczególności zdaniem równorzędnym. Wprowadzenie w taki sposób rekurencji zapewnia możliwość zanalizowania konstrukcji o większej liczbie poziomów, niż ich liczy hierarchia.

Powyższa gramatyka nie zdaje sprawy z tego, jakiego typu jednostki występują jako składniki w prawych stronach reguł — we wszystkich regułach „złożonych" występuje jednostka **zr**. Czy jest w związku z tym mniej szczegółowa od poprzedniej? Przyjrzymy się bliżej jednostkom występującym w pierwszym z cykli w GFJP:

zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, *Z***)**
zsz(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, *Z***)**
zj(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, *Z*, przec**)**
zp(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, *Z***)**
ze(*Wf*, *A*, *C*, *T*, *RI*, *O*, _, _, _, *Neg*, *I*, *Z*, _**)**

Wymienione jednostki można swobodnie przepisywać jedna na drugą z zachowaniem wymienionych wartości parametrów (w szczególności **zp** stanowi **zj** o oznaczeniu spójnika przec). To zaś oznacza, że wszystkie te jednostki są sobie dystrybucyjnie równoważne. Trudno twierdzić, że w GFJP zdanie równorzędne składa się z dwóch zdań szeregowych połączonych spójnikiem, skoro każde z tych zdań szeregowych ze względu na swoją budowę może być zdaniem równorzędnym lub elementarnym. Tak więc można by w uniformizacji pójść jeszcze dalej i zastąpić wszystkie jednostki występujące w cyklu jedną (na przykład o nazwie **zdanie**):

zdanie ⟶ **zdanie**, **[**i**]**, **zdanie**.

zdanie ⟶ **zdanie**, **[**gdy**]**, **zdanie**.

zdanie ⟶ **[**'Jan', umarł**]**.

zdanie ⟶ **[**'Maria', spała**]**.

zdanie ⟶ **[**'Piotr', czytał**]**.

Otrzymana w ten sposób gramatyka opisuje ten sam zbiór zdań i przypisuje im drzewa o tych samych kształtach (z dokładnością do długości nierozgałęziających się gałęzi). Jeżeli chcemy oddać jakoś typ budowy poszczególnych realizacji zdania, można wprowadzić parametr *Typ_budowy*, ze świadomością wszakże, że nie bierze on udziału w uzgodnieniach i spełnia funkcję czysto dekoracyjną.

Jak sądzę, z powyższych rozważań wynika następujący wniosek: granicą szczegółowości dla jednostek nieterminalnych gramatyki metamorficznej są klasy abstrakcji równoważności dystrybucyjnej. Stosowanie jednostek bardziej szczegółowych prowadzi do anomalii w gramatyce i powoduje konieczność wyjścia przy interpretacji poza mechanizm gramatyk metamorficznych.

Jałowa rekurencja w programie *Świgra*

Implementacja GFJP ma z założenia być możliwie wierna oryginałowi. W związku z tym postanowiłem nie wprowadzać zbyt daleko idących zmian w regułach gramatyki i (podobnie jak Szpakowicz) zablokować cykle zamiast je usuwać. Poniżej przedstawiam sposób zablokowania jałowej rekursji, przy założeniu, że analiza odbywa się z dołu do góry. Rozważmy następujący fragment gramatyki:

s ⟶ **a**.

a ⟶ **b**.

a ⟶ **d**, **e**.

a ⟶ **a**, **c**.

b ⟶ **c**.

b ⟶ **c**, **f**.

c ⟶ **a**.

c ⟶ **g**.

Zawiera on cykl **a** ⟶ **b** ⟶ **c** ⟶ **a**. Zakładamy, że nie ma on wspólnych krawędzi z innymi jałowymi cyklami w gramatyce. Dla uproszczenia pomijamy argumenty jednostek nieterminalnych.

Sposób blokowania cykli polega na liczeniu, ile razy pod rząd użyto jałowych reguł. Dopuszczalne jest o jedno mniej takich przejść niż wynosi długość cyklu.

Aby zrealizować liczenie, do każdego nieterminala w jałowym cyklu dodano jeden argument:

```
:- op(100, fy, '@').
```

s ⟶ **a**(_).

a(*X*) ⟶ **b**(@ *X*).

a(@ @ 0) ⟶ **d**, **e**.

a(@ @ 0) ⟶ **a**(_), **c**(_).

b(*X*) ⟶ **c**(@ *X*).

b(@ @ 0) ⟶ **c**(_), **f**.

c(*X*) ⟶ **a**(@ *X*).

c(@ @ 0) ⟶ **g**.

Term @ @ 0 stanowi reprezentację liczby 2, czyli długości jałowego cyklu pomniejszonej o jeden. Stała ta stanowi „kredyt zaufania" który przyznaje się każdemu nowo rozpoznanemu nieterminalowi, poprzez który można wejść w jałowy cykl („wejść" analizując z dołu do góry). Każde przejście przez jałową regułę cyklu powoduje zmniejszenie kredytu zaufania o 1. Przy tym przez jałowe reguły nie da się przejść z kredytem mniejszym od jedności. Pełny kredyt zaufania przyznawany jest lewej stronie reguł o więcej niż jednym nieterminalu po prawej stronie. Jest to bezpieczne przy założeniu braku reguł o pustej prawej stronie.

Poprzez analizę wyjść z cyklu można by zmniejszyć kredyt zaufania przyznawany w „bocznych wejściach" do cyklu. Na przykład w powyższej gramatyce jedynymi wyjściami są **a** i **c**, więc kredyt w trzeciej regule dla **a** można zmniejszyć do jednego (aby wystarczał do przejścia do **c**):

a(@ 0) ⟶ **a**(_), **c**(_).

Podobnie dla drugiej reguły dla **a**; w drugiej regule dla jednostki **b** kredyt musi pozostać równy 2, aby dawało się ją przekształcić w **c**. Taka analiza nie została wykonana dla gramatyki Świdzińskiego.

Term reprezentujący wartość licznika kredytu dla cyklu jest przy wypisywaniu zamieniany na odpowiednią liczbę naturalną. Jego obecność nie jest całkowicie ukrywana — w wydrukach drzew jednostki należące do cykli mają o jeden parametr więcej niż w GFJP.

5.2. Konstrukcja zdania elementarnego i kwestia wymagań czasownikowych

Sposób realizacji wymagań czasownikowych stanowi w GFJP centralny problem dla budowy niezłożonych konstrukcji zdaniowych. Niestety te akurat reguły są sformułowane niekonsekwentnie i wymagają znaczącego przekształcenia, aby otrzymać działający program.

W tym punkcie omówię repertuar fraz, które w GFJP są opisane jako wymagane (rozważane są jedynie wymagania czasowników), następnie przedstawię opis zdań elementarnych w GFJP i w końcu wariant tego opisu, który zapewnia efektywną analizę automatyczną (w szczególności omówię konstrukcję słownika wymagań czasownikowych).

5.2.1. Typy fraz wymaganych

W GFJP jednostką zdaniową najniższego poziomu jest zdanie elementarne (**ze**). W wariancie, nazywanym przez Świdzińskiego standardowym, może ona być zrealizowana jako ciąg złożony z frazy finitywnej (**ff**) i od zera do trzech fraz wymaganych (**fw**). Składnikiem zdania elementarnego może być dodatkowo fraza luźna (**fl**). Występują także realizacje niestandardowe zawierające zdanie równorzędne.

Wśród parametrów frazy finitywnej reprezentowane są w szczególności typy fraz wymaganych dopuszczalne dla czasownika stanowiącego jej centrum. Typy te są określone dla danego czasownika przez słownik wymagań.

Frazy wymagane reprezentują składniki implikowane syntaktycznie (w sensie pracy Saloniego i Świdzińskiego 2001, r. X) przez frazę finitywną (por. GFJP, s. 191). Mogą one być realizowane (pośrednio, poprzez poziom frazy wymaganej właściwej **fw1**) przez sześć rodzajów fraz wymienionych niżej, a także przez frazę pustą. Potrzeba dopuszczenia tej ostatniej realizacji wynika stąd, że w GFJP każda forma czasownikowa ma trzy wymagania. Schematy o mniejszej liczbie fraz wymaganych są reprezentowane za pomocą wartości nic na pozycji niektórych wymagań. W analizatorze składniowym ta wartość została wyeliminowana (por. niżej, p. 5.2.4).

Pierwszym parametrem frazy wymaganej jest *typ frazy wymaganej Tfw*. Wartości tego parametru zależą od tego, jaka fraza realizuje frazę wymaganą. Dla frazy werbalnej bezokolicznikowej *Tfw* przyjmuje wartość aspektu czasownika stanowiącego centrum frazy. Dla frazy przyimkowej jest to wartość parametru przyimek tej frazy. Dla frazy nominalnej oznaczeniem typu frazy wymaganej staje się wartość przypadka właściwa tej frazie. Dla frazy przymiotnikowej oznaczeniem tym jest wartość przypadka uzupełniona symbolem prim. Dla frazy przysłówkowej jest to symbol przys. Dla frazy zdaniowej — wartość parametru *typ frazy zdaniowej*.

Zastosowanie jako typów fraz wymaganych wartości pewnych parametrów charakteryzujących frazy realizujące wymagania ma pewne niekorzystne konsekwencje. Aby zapewnić faktyczne rozróżnienie typów fraz wymaganych, konieczne było sztuczne zmodyfikowanie

oznaczeń przypadka dla fraz przymiotnikowych. Wprowadzone zostały arbitralne oznaczenia przyimków. Ponadto taki zestaw oznaczeń powoduje trudności przy rozbudowie gramatyki — trzeba zwrócić baczną uwagę, czy np. nowowprowadzona wartość typu frazy zdaniowej nie jest jednocześnie nazwą przyimka (co gorsza ta ostatnia lista jest zadana w słowniku a nie w samej gramatyce).

W programie *Świgra* parametr *Tfw* ma postać jedno- lub dwuargumentowego termu. Funktor tego termu określa typ frazy, argumenty zaś przekazują informację o charakterystycznych cechach fraz. Pozwala to uniknąć wymienionych wyżej problemów, jak również, moim zdaniem, zwiększa czytelność zapisów w słowniku wymagań czasownikowych.

Oto lista możliwych postaci parametru *Tfw*:

infp(*A*)	fraza bezokolicznikowa o aspekcie *A*
prepnp(*F*, *P*)	fraza przyimkowa z przyimkiem o wykładniku *F* w przypadku *P*
np(*P*)	fraza nominalna w przypadku *P*
adjp(*P*)	fraza przymiotnikowa w przypadku *P*
advp	fraza przysłówkowa
sentp(*T*)	fraza zdaniowa typu *T*

5.2.2. Reguły opisujące zdanie elementarne w GFJP

Oto pierwsza z reguł opisujących realizację standardową zdania elementarnego **ze**:

ze(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *Neg*, *I*, *Z*, *Ow***)** ⟶ (e1)
fl(*A*, *C*, *RI*, *O*, *Neg*, *I*, *Z1***)**,
⟪**ff(***Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Neg*, ni, *Z*, *Ow***)**,
fw(*Wa*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z2***)**,
fw(*Wb*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z3***)**,
fw(*Wc*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z4***)**⟫,
{ rowne(*Z*, [p, px, pxx]),
rowne(*Z1*, [p, np]),
rowne(*Z2*, [p, np]),
rowne(*Z3*, [p, np]),
rowne(*Z4*, [p, np]) }.

Symbole ⟪ i ⟫ oznaczają permutację objętych nimi jednostek nieterminalnych (por. GFJP, s. 319).

Pozostałe reguły opisujące realizacje standardowe różnią się od przytoczonej (nie)obecnością inicjalnej frazy luźnej **fl**, oraz zestawem warunków. Świdziński zawsze określa w regule co jest składnikiem inicjalnym danej realizacji (ten składnik uzgadnia parametr inkorporacji). Składnik ten zatem zawsze stoi poza znakami permutacji.

W regułach tych warto zwrócić uwagę na permutowanie takich samych elementów. Mianowicie trzy jednostki **fw** podlegają dokładnie tym samym ograniczeniom. Jeśli jednak rozumieć permutację dosłownie, jako zamienianie kolejności również tych elementów, to w zestawieniu z permutowaniem wartości parametrów wymagań (zob. niżej) uzyska się wiele identycznych drzew. W notacji „permutacyjnej" chodzi więc raczej o wstawienie frazy finitywnej **ff** w któreś miejsce ciągu fraz wymaganych **fw**.

Frazy wymagane

Na mocy reguły (wy5) fraza wymagana może mieć realizację pustą:

fw(*Tfw*, *K*, *A*, *C*, *Rl*, *O*, *Neg*, ni, np**)** ⟶ (wy5)

Warto zauważyć, że wartość parametru *Tfw* jest nieustalona, co oznacza, że fraza wymagana dowolnego typu może mieć realizację pustą, a więc że możliwe są realizacje eliptyczne wypowiedzeń.

Niepuste frazy wymagane składają się z frazy wymaganej właściwej i opcjonalnej frazy luźnej. Frazy wymagane właściwe są realizowane odpowiednio do wartości typu frazy wymaganej przez frazę werbalną bezokolicznikową, przyimkową, nominalną, przymiotnikową, przysłówkową lub zdaniową. Tylko fraza pusta może mieć typ nic.

Warto w tym miejscu przypomnieć, że fraza podmiotowa jest w GFJP zwykłą frazą wymaganą. Jest ona zdefiniowana jako fraza nominalna w mianowniku. Jej cechą szczególną jest to, że uzgadnia parametry rodzaju-liczby i osoby, w odróżnieniu od pozostałych fraz nominalnych. W GFJP jest zresztą w tym miejscu prosty błąd techniczny sprawiający, że wszystkie frazy nominalne uzgadniają te parametry. Poprawione reguły prezentują się następująco:

fw1(np(mian), *K*, *A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z***)** ⟶ (wy8)
  fno(mian, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*, _**)**.

fw1(np(bier), *K*, *A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z***)** ⟶ (wy9)
  fno(dop, *Rl1*, *O1*, *Neg*, *I*, *Z*, *Kl*, _**)**,
  { rowne(*Neg*, [ani, nie]) }.

fw1(np(bier), *K*, *A*, *C*, *Rl*, *O*, tak, *I*, *Z***)** ⟶ (wy10)
  fno(bier, *Rl1*, *O1*, tak, *I*, *Z*, *Kl*, _**)**.

fw1(np(*P*), *K*, *A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z***)** ⟶ (wy11)
  fno(*P*, *Rl1*, *O1*, *Neg*, *I*, *Z*, *Kl*, _**)**,
  { rozne(*P*, ['mian', 'bier', 'wol']) }.

Reguła (wy9) realizuje tzw. dopełniacz negacji.

Fraza finitywna

Parametry określające wymagania *Wa*, *Wb*, *Wc* zdania elementarnego są przekazywane w postaci niezmienionej do frazy finitywnej **ff**, frazy finitywnej właściwej (reguła (fi1)–(fi3)) i wreszcie do frazy werbalnej (fi4). W regułach (we1)–(we19) fraza werbalna jest opisywana jako fraza werbalna z negacją przepleciona z frazami wymaganymi, które realizują część wymagań czasownika. W takim wypadku na odpowiedniej pozycji pojawia się wartość wymagania nic. Na przykład:

fwe(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wa*, nic, nic, *K*, *Neg*, *I*, *Z***)** ⟶ (we4)
  kweneg(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Neg*, *I*, *Z***)**,
  fw(*Wb*, *K*, *A*, *C*, *Rl*, *O*, *Neg*, ni, *Z1***)**,
  fw(*Wc*, *K*, *A*, *C*, *Rl*, *O*, *Neg*, ni, *Z2***)**,
  { rowne(*Z*, [p, px, pxx]),

rowne(*Z1*, [np, p]),
rowne(*Z2*, [np, p]) }.

W regułach wymienione są wszystkie kombinacje szyku jednej konstrukcji werbalnej z negacją **kweneg** i od zera do trzech fraz wymaganych **fw**.

Nie jest dla mnie jasny cel realizacji wymagań wewnątrz frazy czasownikowej występującej jako fraza finitywna. Nie udało mi się też uzyskać przekonującego wyjaśnienia od Autora. Jak sądzę, możliwość realizacji wymagań została wprowadzona w tych regułach ze względu na użycie jednostki **fwe** w regułach opisujących frazę wymaganą bezokolicznikową i frazę luźną imiesłowową (a więc niefinitywne użycia frazy werbalnej). W takich frazach konieczna jest realizacja ich wymagań wewnątrz frazy werbalnej (np. *Marii* i *książkę* we frazie *czytając Marii książkę*). Jednak gdy jednostka **kweneg** konstytuuje frazę finitywną, pojawiają się możliwości interpretacji, które wydają się nadmiarowe. Wszystkie lub wybrane wymagania mogą bowiem być zrealizowane wewnątrz jednostki **fwe** lub dopiero na poziomie zdania elementarnego. Daje to różne struktury drzew, którym chyba nie odpowiada żadne faktyczne zróżnicowanie lingwistyczne.

Prof. Świdziński przyznał w dyskusji, że dobrym rozwiązaniem tego problemu jest użycie w regule opisującej frazę finitywną właściwą jednostki **kweneg** zamiast **fwe**:

ff1(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Neg*, *I*, *Z*, *Ow***)** ⟶ (fi4)
kweneg(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Neg*, *I*, *Z1***)**,
{ rowne(*Wf*, [os, bos, bok]),
oblzal(*Z1*, *Z*, *Ow*) }.

Takie właśnie rozwiązanie zostało zastosowane w programie *Świgra*.

Następnie wymagania przechodzą niezmienione przez poziom frazy werbalnej z inkorporacją by wreszcie trafić do następujących reguł opisujących konstrukcję werbalną:

kwer(bos, *A*, prze, *T*, *RI*, 3, *Wa*, *Wb*, *Wc*, *K*, *Z***)** ⟶ (we27)
kwer1(bos, *A*, prze, *T*, *RI*, 3, *W1*, *W2*, *W3*, *K*, *Z***)**,
{ rowne(*W1*, [*Wa*, *Wb*, *Wc*]),
rowne(*W2*, [*Wa*, *Wb*, *Wc*]),
rowne(*W3*, [*Wa*, *Wb*, *Wc*]) }.

kwer(*Wf*, *A*, *C*, *T*, *RI*, 3, *Wa*, *Wb*, *Wc*, *K*, *Z***)** ⟶ (we28)
kwer1(*Wf*, *A*, *C*, *T*, *RI*, 3, *W1*, *W2*, *W3*, *K*, *Z***)**,
{ rowne(*Wf*, [bok, psu, psw]),
rowne(*W1*, [*Wa*, *Wb*, *Wc*]),
rowne(*W2*, [*Wa*, *Wb*, *Wc*]),
rowne(*W3*, [*Wa*, *Wb*, *Wc*]) }.

kwer(os, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Z***)** ⟶ (we29)
kwer1(os, *A*, *C*, *T*, *RI*, *O*, *W1*, *W2*, *W3*, *K*, *Z***)**,
{ rowne(*W1*, [*Wa*, *Wb*, *Wc*]),
rowne(*W2*, [*Wa*, *Wb*, *Wc*]),
rowne(*W3*, [*Wa*, *Wb*, *Wc*]) }.

W regułach tych nieco zaskakujące są warunki. Oznaczają one bowiem, że *W1*, *W2* i *W3* mogą przyjąć wartość tego samego wymagania np. *Wa*. Jeśli jednostka **kwer1** ma na przykład *Wa* = mian to możliwa jest realizacja zdania elementarnego zawierającego trzy frazy wymagane rzeczownikowe w mianowniku. Chyba nie o to chodziło Autorowi gramatyki. Bardziej prawdopodobna wydaje się hipoteza, że według tych reguł *W1*, *W2*, *W3* miało być pewną permutacją *Wa*, *Wb* i *Wc*.

Jednak nawet przyjęcie takiej interpretacji prowadzi do wyraźnie nadmiarowych analiz. Wyobraźmy sobie bowiem, że pewien czasownik ma wymagania: *Wa* = mian, *Wb* = nic, *Wc* = nic (na przykład forma *spał*). Najpierw zauważmy, że permutowanie w (we27–29) nie może dotyczyć zmiennych *Wa*, *Wb* i *Wc*, tylko ich wartości. W przeciwnym razie dostaniemy po dwie identyczne interpretacje, w których na przykład *W1* = *Wb* i *W2* = *Wc* albo *W1* = *Wc* i *W2* = *Wb*. (Ta uwaga dotyczy oczywiście wszelkich powtarzających się wartości wymagań, nie tylko wartości nic).

Jednak nawet jeśli zauważymy, że te interpretacje są identyczne, pozostaje swoboda wyboru, który z parametrów ma mieć wartość różną od nic. W zależności od tego wyboru dwie puste frazy wymagane będą umieszczone w zdaniu elementarnym razem lub osobno, przed, po lub pomiędzy frazą finitywną i frazą wymaganą mianownikową. To bogactwo różnych strukturyzacji związane ze swobodą szyku pustych fraz wymaganych chyba nie jest warte tak rozbudowanego opisu. W każdym razie w wypadku realizacji komputerowej tę różnorodność drzew, które nie wnoszą nic do wiedzy o analizowanym zdaniu, odbiera się wyraźnie jako nadmiar.

Zauważmy jeszcze, że przy takiej organizacji reguł, przy analizie prowadzonej w porządku wstępującym, najpierw konstruowane są frazy finitywne z wszystkimi możliwymi permutacjami wartości wymagań i dopiero wtedy spośród nich jest wybierana ta, która odpowiada szykowi analizowanego wypowiedzenia. Oczywiście odbija się to negatywnie na efektywności analizy.

W programie *Świgra* konstrukcja zdania elementarnego została zmieniona tak, aby uzyskiwać drzewa bardziej zgodnie z duchem GFJP niż z jej literą. Zanim jednak zapoznamy się z tymi zmianami, przyjrzyjmy się organizacji słownika wymagań.

5.2.3. Słownik wymagań czasownikowych

W GFJP konstrukcja werbalna właściwa **kwer1** jest realizowana przez jednostkę **formaczas**. Na przykład:

kwer1(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*) ⟶ (we30)
  formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*),
  { rowne(*Z*, [np, npx, npxx]) }.

Świdziński nie podaje definicji tej ostatniej jednostki. Jeśli jednak **formaczas** miałaby być zdefiniowana podobnie do innych jednostek elementarnych GFJP, można by się spodziewać reguły takiego kształtu, por. (jel1–7):

formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*) ⟶ [*F*],
  { słow(*F*, czasownik, *Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*) }.

Słownik form programu musiałby więc zawierać wszystkie dopuszczalne kombinacje wymagań dla każdej formy czasownikowej. (Ignorujemy tu fakt, że czasem „forma czasownikowa" jest realizowana przez więcej niż jedno słowo tekstowe, por. 4.7.5).

Z takim rozwiązaniem wiążą się co najmniej dwa problemy. Po pierwsze, realizacja komputerowa byłaby nieefektywna, ponieważ dla każdej dopuszczalnej kombinacji wartości parametrów *Wa*, *Wb*, *Wc* konstruowana byłaby pełna fraza finitywna (różniąca się od innych tylko wartościami tych parametrów) i dopiero wtedy program mógłby podjąć sprawdzenie, czy te wartości wymagań dają się zrealizować w konkretnym zdaniu. Mielibyśmy zatem do czynienia z głębokimi nawrotami do słownika w celu pobrania kolejnego zestawu wartości wymagań.

Po drugie, dla pewnych zdań byłyby produkowane drzewa rozbioru różniące się tylko wartościami wymagań zrealizowanych jako puste frazy wymagane. Weźmy dla przykładu takie zdanie:

(87) *Jan dał Piotrowi.*

Z formą *dać* może być związany zestaw wymagań *Wa* = mian, *Wb* = cel, *Wc* = bier (a więc wymaganie trzech fraz rzeczownikowych w mianowniku, celowniku i bierniku, *Jan dał Piotrowi książkę*). Ale jest też możliwy taki na przykład zestaw: *Wa* = mian, *Wb* = cel, *Wc* = nd (frazy rzeczownikowe w mianowniku i celowniku i fraza werbalna w bezokoliczniku, na przykład niedokonana, *Jan dał Piotrowi odpocząć*). Zdanie przykładowe jest niepełne (eliptyczne) i według reguł GFJP może być zanalizowane z oboma zestawami wartości wymagań. Dla pierwszego zestawu będziemy mieć pustą frazę wymaganą o *Tfw* = bier a dla drugiego *Tfw* = nd. (Widać tu zresztą kolejne źródło nieefektywności: zapisane w słowniku wymaganie frazy bezokolicznikowej musi mieć postać nd lub dk, a skoro większość czasowników, które dopuszczają frazę bezokolicznikową, dopuszcza frazy obu aspektów, większość ta musi mieć po dwa zestawy wymagań, wybierane poprzez głęboki nawrót do słownika).

W analizatorze składniowym parametry *Wa*, *Wb*, *Wc* zostały zastąpione jednym parametrem (będziemy go nazywać *Wym*), który reprezentuje pełny opis wymagań danego czasownika. Parametr ten reprezentuje łącznie wszystkie warianty wartości wymagań dopuszczalne dla danej formy. Jest on przekazywany w niezmienionej postaci od słownika aż do frazy finitywnej. Na poziomie zdania elementarnego jego wartości sterują rozpoznaniem odpowiedniej sekwencji fraz wymaganych. Szczegółowo opisuję ten proces w następnym punkcie. Podobnie zrealizowane jest wypełnianie wymagań w obrębie frazy werbalnej.

Jak wyjaśniłem w punkcie 4.7, *Świgra* korzysta ze słownika leksemowego. Zgodnie z regułą przedstawioną w punkcie 4.7.5 wymagania dla danej formy są pobierane na podstawie identyfikatora leksemu, do którego forma została zakwalifikowana.

Słownik wymagań analizatora *Świgra* jest, jak pisałem, prowizoryczny, ma obecnie kilkaset haseł. Dlatego w programie można sobie pozwolić na reprezentowanie go w postaci prologowego predykatu slowczas, którego klauzule opisują poszczególne leksemy. (We współczesnym interpreterze Prologu dzięki indeksowaniu klauzul taka prosta reprezentacja może okazać się wystarczająca nawet dla słownika zawierającego tysiące pozycji).

Oto fragment słownika wymagań analizatora *Świgra*:

```
slowczas(bać, s, [[np(mian), np(dop)], [np(mian), infp(_)], [np(mian), sentp(że)], [np(mian),
        sentp(czy)], [np(mian), prepnp(o, bier)]]).
slowczas(bawić, n, [[np(mian), np(bier), np(narz)]]).
```

slowczas(bawić, s, [[np(mian), prepnp(z, narz), prepnp(w, bier)], [np(mian), prepnp(z, narz), np(narz)]]).

slowczas(znać, n, [[np(mian), np(bier)], [prepnp(po, miej), np(bier)]]).

slowczas(znać, s, [[np(mian), prepnp(z, narz)], [np(mian), prepnp(na, miej)]]).

Predykat slowczas jest trójargumentowy. Pierwszy argument jest identyfikatorem leksemu. Drugi argument określa, czy opisywana jest konstrukcja z *się*, czy bez *się* — mają one inny zestaw wymagań. Należy to rozumieć w ten sposób, że uznajemy istnienie osobnych leksemów np. BAWIĆ i BAWIĆ SIĘ (ściślej rzecz biorąc, leksemu BAWIĆ, którego formy nie dopuszczają połączenia z *się* i leksemu BAWIĆ, którego formy obligatoryjnie łączą się z *się*). Ponieważ jednak leksemy te mają identyczne formy fleksyjne, opisujemy je łącznie na poziomie analizy morfologicznej. Natomiast w słowniku wymagań mają one dwa osobne opisy. Dla czasownika BAĆ w słowniku nie będzie klauzuli z wartością n na drugiej pozycji, w związku z czym reguły z p. 4.7.5 nie dopuszczą konstrukcji, w których formom tego czasownika, którym nie towarzyszy *się*.

Trzeci argument opisuje wymagania dla danego leksemu. Jest on listą maksymalnych zestawów wymagań, a więc takich, które nie są podzbiorami innych zestawów wymagań dopuszczalnych dla danego czasownika. Każdy zestaw jest listą wartości parametru *Tfw* wymienionych w punkcie 5.2.1. W przykładzie wartość infp(_) oznacza frazę bezokolicznikową o dowolnym aspekcie.

Na przykład dla czasownika *znać* występującego bez *się* mamy możliwe m.in. takie dwa wypowiedzenia:

(88) *Jan zna zniszczenia wojenne.*
(89) *Po domu znać zniszczenia wojenne.*

W pierwszym mamy zrealizowane wymagania np(mian) i np(bier). W drugim natomiast wymagania prepnp(po, miej) i np(bier). Zauważmy, że te dwa zestawy są maksymalne. W szczególności, mimo że w obu występuje fraza nominalna w bierniku, nie można tych schematów połączyć:

(90) **Jan po domu zna zniszczenia wojenne.*

5.2.4. Realizacja wymagań czasownikowych w analizatorze *Świgra*

Reguły opisujące zdanie elementarne **ze** w GFJP można (na poziomie intencji, a nie faktycznego zapisu) streścić następująco:

— Frazie finitywnej można przypisać od zera do trzech wymagań.
— Realizacja wymagania polega na tym, że składnikiem bezpośrednim zdania elementarnego jest fraza wymagana o wartości parametru *Tfw* równej jednemu z wymagań frazy finitywnej.
— Porządek, w jakim występuje fraza finitywna i frazy wymagane, jest dowolny.
— Dowolne z wymagań mogą pozostać niewypełnione (w oryginalnej gramatyce — zrealizowane jako pusta fraza wymagana).

Z dyskusji reguł GFJP przedstawionej wyżej wynika, że konieczne są pewne modyfikacje aby faktycznie uzyskać takie zachowanie programu.

Jak już powiedziano wcześniej trzy parametry *Wa*, *Wb*, *Wc* opisujące wymagania zostały zastąpione jednym parametrem *Wym*. Parametr ten przekazuje wartość pobraną ze słownika wymagań w regule opisującej formę czasownikową **formaczas** bez żadnych zmian ani uzgodnień aż do frazy finitywnej **ff** (z wyjątkiem realizacji wymagań wewnątrz frazy werbalnej, które odbywa się analogicznie, jak na poziomie zdania elementarnego). W szczególności wymagania nie są permutowane w regułach (we27–29).

W regułach opisujących realizacje zdania elementarnego zamiast permutacji wprowadzone zostały dwa operatory. Noszą one nazwy **wymagania** i **wymagane** i z punktu widzenia składniowego wyglądają jak jednostki nieterminalne.

Przedstawiony niżej zapis jest nieco skomplikowany, ale nie widzę zgrabniejszego sposobu wyrażenia potrzebnej tu funkcji. Można patrzeć na sprawę w ten sposób, że realizacja wymagań czasownikowych została zapisana bezpośrednio w Prologu. A można widzieć tu jakieś ogólnie użyteczne rozszerzenie gramatyk metamorficznych. Jest ono wyrażone w sposób na tyle ogólny, że powinno dać się zastosować w dowolnym wypadku, gdy wymagania jakiejś frazy mają być wypełniane przez inne frazy określonego typu.

Operator **wymagania** jest pięcioargumentowy. Jego wystąpienie ma postać:

wymagania(*WymWyp*, *Wym*, *WymReszta*, *FF*, [*W1*/*FW1*, *W2*/*FW2*, *W3*/*FW3*]**)**.

Interpretacja operatora **wymagania** jest następująca: analizowany fragment tekstu musi się dać zinterpretować jako ciąg złożony z wystąpienia jednostki *FF* i jednostek *FW1*, *FW2* i *FW3*. Jednostka *FF* może przy tym wystąpić przed, po lub pomiędzy którymiś *FWn*, ich porządek natomiast jest ustalony. Co więcej jednostki *FWn* nie muszą wystąpić wszystkie, jeśli jednak któraś jest realizowana to i wszystkie poprzedzające. (Implementacja pozwala wymienić tu dowolnie wiele jednostek *FWn*, w regułach GFJP występuje nie więcej niż trzy).

Argument *Wym* jest opisem wymagań czasownikowych przysługujących jednostce *FF*. Zmienna *W1* jest jednym z argumentów jednostki *FW1* określającym jej typ jako frazy realizującej wymaganie (w GFJP jest to parametr *Tfw* frazy wymaganej **fw**). Podobnie *W2* i *W3*.

Argument *WymWyp* jest listą wymagań, które zostały już zrealizowane przed przystąpieneim do obliczania operatora **wymagania**. W GFJP będzie on miał wartość niepustą, jeżeli któraś z fraz wymaganych występuje przed „permutacją" (jako element inicjalny reguły, por. reguła (e2) na następnej stronie). Argument *WymReszta* reprezentuje wymagania, które pozostały niezrealizowane. W GFJP argument ten pozostaje zmienną wolną co oznacza, że dopuszczalne są realizacje eliptyczne. Nakładając warunki na ten argument można wykluczyć wszystkie lub niektóre takie realizacje.

Opiszę teraz zależności między wymienionymi wyżej parametrami, które muszą zachodzić aby wystąpienie operatora **wymagania** mogło zostać pomyślnie obliczone. Dla poglądowości przyjmijmy na chwilę, że każdy czasownik ma dany jeden zestaw wymagań, który reprezentujemy w postaci listy *Wym*. Wówczas, aby zrealizować wymagania, wszystkie wartości z listy *WymWyp* muszą być elementami listy *Wym*. Jeżeli nie są realizowane żadne jednostki *FWn*, wartością *WymReszta* jest różnica list *Wym* i *WymWyp*. Jeżeli jest realizowana jednostka *FW1*, to elementem wspomnianej różnicy musi być *W1*, które się z niej usuwa. Jeżeli jest realizowana *FW2*, podobną operację wykonuje się na *W2* i podobnie przetwarzane są kolejne elementy listy. Wartością *WymReszta* jest lista *Wym*, z której wyjęto wszystkie zrealizowane wymagania.

Rzeczywisty obraz jest bardziej skomplikowany ze względu na możliwość alternatywnych zestawów wymagań dla czasowników. Jak to zilustrowano w punkcie 5.2.3, często dla czasownika możliwych jest kilka wykluczających się zestawów wymagań. Co gorsza nie muszą one być rozłączne, więc jeśli dopuszcza się realizacje eliptyczne, mogą zdarzyć się zdania (lub frazy), które realizują część wspólną kilku wykluczających się zestawów wymagań. Na przykład we fragmencie wypowiedzenia *znać zniszczenia* zrealizowane zostało wymaganie np(bier) wspólne dwóm zestawom wymagań dla *znać*.

Z tego względu parametr *Wym* jest listą list wymagań. Na przykład dla *znać* może on mieć postać: [[np(mian), np(bier)], [prepnp(po, miej), np(bier)]]. W uproszczonym przypadku omówionym wyżej przy rozpoznawaniu kolejnych fraz *FWn* z listy wymagań wyjmowało się odpowiadające frazie wymaganie *Wn*. Obecnie trzeba wykonać bardziej skomplikowaną operację. Chodzi w niej o obliczenie listy możliwych jeszcze nie wypełnionych wymagań dla analizowanej jednostki. Jeśli dana fraza *FWn* ma realizować wymaganie, to trzeba z listy wybrać tylko te listy wymagań, które zawierają *Wn* i w dalszym przetwarzaniu uwzględniać listę tych list z wyjętym elementem *Wn*. Na przykład rozpoznanie frazy o charakterystyce np(bier) powoduje przekształcenie listy wymagań do następującej postaci: [[np(mian)], [prepnp(po, miej)]]. Lista ta pozwala w dalszym przetwarzaniu rozpoznać frazę typu np(mian) albo prepnp(po, miej), ale nie obie. Pojawienie się w jakimś momencie przetwarzania listy pustej sygnalizuje porażkę (nie istnieją zestawy wymagań, z których dało się wyjąć dane wymaganie). Natomiast w wyniku pełnej realizacji wymagań jakiegoś schematu otrzymuje się listę wymagań, której jedynym elementem jest lista pusta (wymagań, które pozostały do zrealizowania).

Oto reguła (e1) zapisana za pomocą operatora **wymagania**:

```
ze(Wf, A, C, T, Rl, O, Wym, Neg, I, Z, Ow) ⟶                          (e1)
   fl(A, C, Rl, O, Neg, I, Z1),
   wymagania([], Wym, Reszta Wym,
      ff(Wf, A, C, T, Rl, O, Wym, K, Neg, ni, Z, Ow),
      [W1/fw(W1, K, A, C, Rl, O, Neg, ni, Z2),
       W2/fw(W2, K, A, C, Rl, O, Neg, ni, Z3),
       W3/fw(W3, K, A, C, Rl, O, Neg, ni, Z4)]),
   { rowne(Z, [p, px, pxx]),
   rowne(Z1, [p, np]),
   rowne(Z2, [p, np]),
   rowne(Z3, [p, np]),
   rowne(Z4, [p, np]) }.
```

Argument *WymWyp* jest niepusty w regułach z inicjalną frazą wymaganą, na przykład:

```
ze(Wf, A, C, T, Rl, O, Wym, Neg, I, Z, Ow) ⟶                          (e2)
   fw(W1, K, A, C, Rl, O, Neg, I, Z1),
   wymagania([W1], Wym, Reszta Wym,
      ff(Wf, A, C, T, Rl, O, Wym, K, Neg, ni, Z, Ow),
      [W2/fw(W2, K, A, C, Rl, O, Neg, ni, Z2),
       W3/fw(W3, K, A, C, Rl, O, Neg, ni, Z3)]),
   { rowne(Z, [p, px, pxx]),
   rowne(Z1, [p, np]),
```

rowne(*Z2*, [p, np]),
rowne(*Z3*, [p, np]) }.

Operator **wymagane** jest uproszczoną wersją poprzedniego używaną w regułach, w których fraza finitywna jest elementem inicjalnym, a więc realizowanym poza „permutacją". Operator ten ma trzy argumenty: **wymagane(***Wym*, *WymReszta*, [*W1*/*FW1*, *W2*/*FW2*, *W3*/*FW3*]**)**. Oto przykład jego użycia:

ze(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *Neg*, *I*, *Z*, *Ow***)** ⟶ (e3)
ff(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, *Neg*, *I*, *Z*, *Ow***)**,
wymagane(*Wym*, *Reszta*Wym,
[*W1*/**fw(***W1*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z1***)**,
W2/**fw(***W2*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z2***)**,
W3/**fw(***W3*, *K*, *A*, *C*, *RI*, *O*, *Neg*, ni, *Z3***)**]**)**,
{ rowne(*Z*, [p, px, pxx]),
rowne(*Z1*, [p, np]),
rowne(*Z2*, [p, np]),
rowne(*Z3*, [p, np]) }.

Oczywiście omawiane operatory są traktowane specjalnie przy tłumaczeniu gramatyki na program prologowy.

5.3. Parametry jednostek nieterminalnych i nałożone na nie warunki

W tym punkcie omawiam parametry jednostek nieterminalnych, które wzbudziły wątpliwości interpretacyjne przy realizacji gramatyki.

5.3.1. Parametr zależności

Oto uwagi o parametrze zależności ze strony 89 GFJP:

> Zależność to jeden z najważniejszych parametrów w tym opisie. Jest to przede wszystkim parametr jednostek zdaniowych. Uzależnia on własności gramatyczne takiej jednostki, a także jej składników bezpośrednich, od rozmaitych czynników zewnętrznych. Zdania równorzędne, szeregowe, jednorodne, proste i elementarne mogą być pytajne, niepytajne, pospójnikowe (dla różnych typów spójników podrzędnych), pytajnozależne lub względne (dla różnych typów zaimków względnych). Wartościami tego parametru są albo oznaczenia specjalne (np. np — wartość niepytajna, p — wartość pytajna, p' — wartość pytajna z ograniczeniem czasu), albo oznaczenia spójnika podrzędnego (np. bo, gdy, żeby), albo oznaczenia typu zaimka względnego (np. kto, co, który). (...)
>
> Parametr zależności przysługuje również frazom składnikowym, czyli frazie finitywnej, frazie wymaganej i frazie luźnej, a także wszystkim szczegółowym typom fraz — z wyjątkiem frazy zdaniowej. Inwentarz wartości ustalonych, jakie przyjmuje fraza finitywna oraz fraza werbalna, jest identyczny ze zbiorem wartości ustalonych dla jednostek zdaniowych. Jest to zrozumiałe, ponieważ można uważać własności składniowe takiej jednostki za wyznaczone przez własności składniowe centrum

> finitywnego. Repertuar wartości ustalonych parametru zależności dla pozostałych fraz jest uboższy. Frazy takie nie przyjmują wartości równych oznaczeniu spójnika podrzędnego, mogą być natomiast niepytajne, pytajne, pytajnozależne lub względne (dla określonego typu zaimka względnego).

Wartości ustalone parametru zależności wymienia Świdziński w punkcie 5.1.4 (s. 103–106). Obejmują one wartości pytajne, niepytajne, odspójnikowe (odpowiadające poszczególnym spójnikom podrzędnym), pytajnozależne i względne (odpowiadające typom fraz względnych).

Dyskusja parametru zależności w GFJP

Niestety interpretacja parametru zależności jest niespójna. W niektórych partiach gramatyki wydaje się on oznaczać: „ta fraza może wystąpić w kontekście typu *Z*" (bo nie zawiera niczego, co by się kłóciło z wystąpieniem w takim kontekście). W innych miejscach raczej: „ta fraza musi wystąpić w kontekście typu *Z*" (bo zawiera jakiś składnik, który wymaga takiego kontekstu, który sprawia że fraza „jest typu *Z*").

W pewnych miejscach gramatyki ta różnica między „może" i „musi" jest istotna. Na przykład na stronie 104 czytamy:

> Jednostka pytajna zawiera przynajmniej jeden składnik bezpośredni pytajny (wyjątkowa jednostka zdaniowa pytajna nie spełniająca tego warunku będzie omówiona w 6.2.3).

Wyjątkiem owym jest pytanie bez jednostki pytajnej, a więc wystąpienie zdania o wartości niepytajnej zależności w kontekście pytajnym (np. wymuszonym przez pytajnik na końcu). Na przykład (s. 181 GFJP):

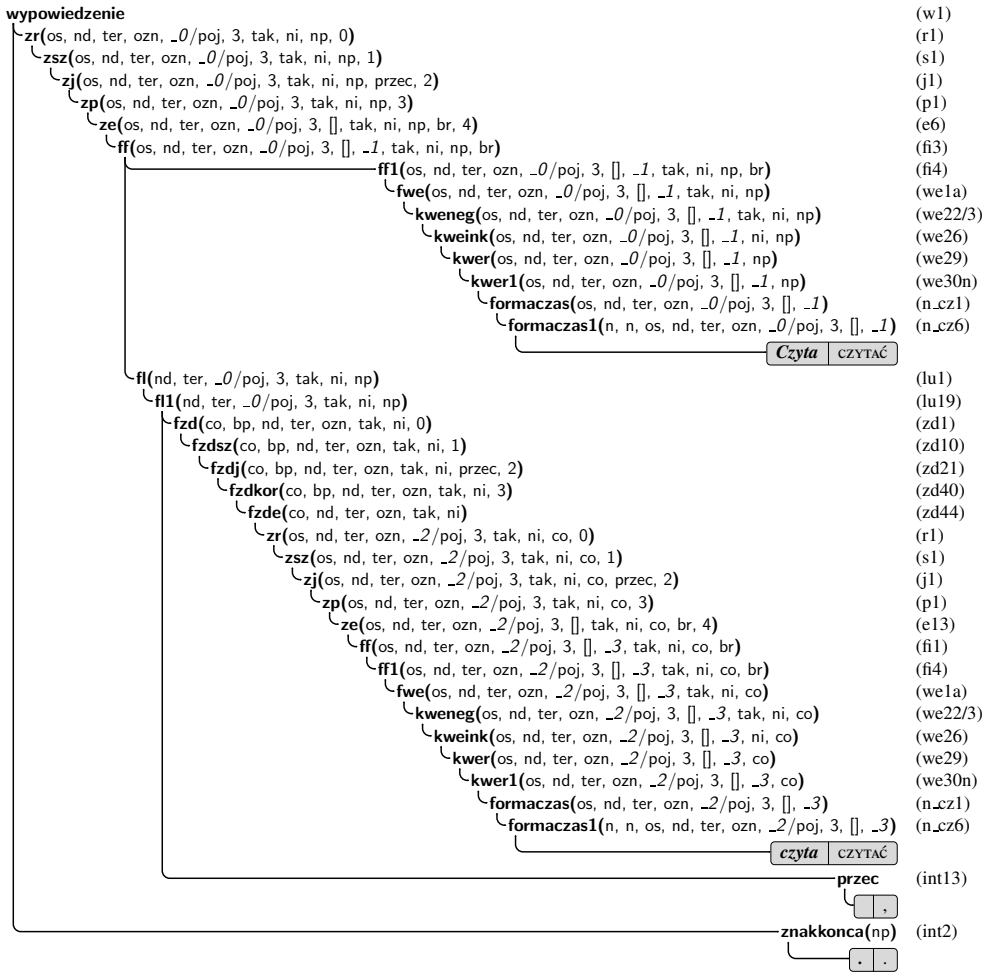
(91) *Przyjdziecie?*
(92) *Kiedy post się skończy, przyjdziecie?*

Taką realizację dopuszcza reguła (e15), według której zdanie równorzędne o wartości npt zależności może zostać zinterpretowane jako zdanie elementarne pytajne (o wartości p, p' lub p" zależności). Wydaje się zatem, że realizacje zdań elementarnych pytajnych według innych reguł powinny spełniać warunek, że co najmniej jeden składnik bezpośredni takiego zdania jest pytajny. Tymczasem na mocy reguły (we31) forma *przyjdziecie* może konstytuować konstrukcję werbalną właściwą o wartości pytajności p, która może być pytajną frazą finitywną i tym samym może być pytajnym zdaniem elementarnym na mocy reguły (e3), która opisuje typowe zdania pytajne, w których to fraza finitywna zawiera element pytajny (jako frazę luźną).

W wyniku otrzymuje się podwójną interpretację zdań pytajnych nie zawierających elementu pytajnego. Problemem jest tu to, że w regule (e3) intencją Autora było zapewne stwierdzenie „fraza finitywna musi być pytajna (zawierać element pytajny)", podczas gdy w (we31) chodzi raczej o „fraza może znaleźć się w kontekście pytajnym".

A oto inny przykład: GFJP akceptuje następujące zdanie, z interpretacją przedstawioną na rysunku 5.1, wedle której drugie wystąpienie słowa *czyta* stanowi frazę luźną realizowaną przez frazę zdaniową typu co.

(93) *!*Czyta czyta.*



Rys. 5.1. Drzewo analizy zdania *Czyta czyta.*

Fraza zdaniowa elementarna typu co może być realizowana przez zdanie równorzędne o wartości zależności co (reguła (zd44)). Takie zdanie elementarne według reguły (e13) może składać się z frazy finitywnej o wartości zależności co i pewnej liczby fraz wymaganych. W tym miejscu objawia się problem. Mianowicie fraza finitywna złożona z formy *czyta* może mieć wartość co zależności na mocy reguły (we32), bo taka fraza ma prawo znaleźć się w kontekście „typu co". Jednak w regule (e13) chodzi raczej o frazę finitywną, która zawiera w sobie element wymuszający „coiczność", czyli formę zaimka co.

Na potrzeby programu *Świgra* rozwiązałem problem luźnych względnych fraz zdaniowych poprzez usunięcie reguły (e13). W obecnej postaci gramatyki reguła ta wprowadza tylko niepoprawne analizy, ponieważ inicjalna fraza finitywna występująca w tej regule nie ma szansy zawierać faktycznego elementu względnego. A jak pisze Świdziński na s. 187 GFJP „cechą charakterystyczną takiego [względnego] zdania elementarnego jest to, że składnik inicjalny jest względny" (co w tym kontekście chyba oznacza „zawiera zaimek względny"). Dzieje się tak, ponieważ fraza finitywna może zawierać frazę luźną (która mogłaby zawierać element względny) tylko w pozycji nieinicjalnej. Zatem inicjalnym składnikiem frazy finitywnej jest fraza werbalna. Ta mogłaby zawierać inicjalną względną frazę wymaganą na mocy reguł (we17–19), jak jednak argumentuję w p. 5.2.2 te realizacje

odrzucam jako nadmiarowe. W związku z tym inicjalnym składnikiem frazy werbalnej jest jednostka **kweneg**, która nie może zawierać inicjalnego składnika względnego.

Wątpliwości co do funkcji parametru zależności

Powstaje pytanie, czy reguły (we30–35) są właściwym miejscem do nadawania wartości parametru zależności. Według tych reguł istnieje 40 różnych konstrukcji werbalnych właściwych realizowanych przez formę *czyta*, różniących się jedynie wartością parametru zależności. Traktując rzecz serio, student uczący się gramatyki powinien nauczyć się rozpoznawać i odróżniać od siebie tych 40 fraz. Jednak odróżnić ich nie sposób, ponieważ budowa wewnętrzna wszystkich czterdziestu jest identyczna, a parametr zależności ma wartości nadane na zasadzie „może". Trudno mi się zgodzić z cytowanym na stronie 87 stwierdzeniem, że własności składniowe (tutaj kontekstowe) jednostki zdaniowej są wyznaczone własnościami składniowymi jej centrum finitywnego. Własności kontekstowe są właśnie cechą kontekstu a nie centrum. A parametr zależności nie opisuje własności frazy, tylko kontekstu, w którym się ona znalazła, por. GFJP s. 102:

> Pozwala on [parametr zależności] zdać sprawę z rozmaitych uwarunkowań zewnętrznych poszczególnych jednostek zdaniowych. Wartości tego parametru odpowiadają kontekstom składniowym, które w różny sposób ograniczają charakterystykę gramatyczną jednostek składniowych w nich się pojawiających.

Przypisanie formie *czyta* jako parametru potencji wystąpienia w kontekście typu co wydaje mi się pomysłem karkołomnym.

Zbyt daleko posunięte wnioski z poglądu o determinacji własności frazy przez jej centrum można łatwo sprowadzić do niedorzeczności. W pewnych formalizmach gramatycznych nazwa jednostki nieterminalnej jest jednym z parametrów tej jednostki (por. Gazdar i Mellish (1989)). Patrząc z tej perspektywy, skoro wszystkie cechy składniowe konstrukcji (i w związku z tym wszystkie parametry danej jednostki) są wyznaczone własnościami jej centrum, to równe powinny być również nazwy jednostek nieterminalnych.

Jak sądzę, byłoby poręczniej przyjąć dla parametru zależności konsekwentną interpretację typu „musi". Oznaczałoby to, że konstrukcje werbalne właściwe mają nienacechowaną (nieograniczoną) wartość np zależności, zaś inne wartości zależności pojawiają się dla bardziej złożonych konstrukcji, gdy wśród składników konstrukcji pojawi się element wymuszający taką właśnie wartość zależności. Oznaczałoby to nadawanie (obliczanie) nowej wartości zależności w większej liczbie reguł niż obecnie, ale ponieważ byłyby to miejsca w których frazy składowe jakoś oddziałują na siebie tworząc całość o innych (bardziej ograniczonych czy konkretnych) właściwościach, w sumie gramatyka zyskała by chyba na czytelności. W obecnej formie trudno jest znaleźć w gramatyce miejsce, w którym nadana w regule (we32) wartość co faktycznie zainterweniuje w budowę jakiejś frazy.

Dobrym pomysłem mogłoby też być ustrukturyzowanie wartości parametru zależności. Można by na przykład wstępnie scharakteryzować pewną grupę konstrukcji jako względne, nadając im wartość zależności wzgl(_), która mogłaby być uszczegółowiona do wzgl(co), wzgl(jaki), wzgl(kto) lub wzgl(który). Dzięki temu uprościłyby się warunki w regułach (e11–13) i podobnych miejscach, gdzie następuje porównanie parametru zależności z listą co.jaki.kto.który. Być może zresztą warto byłoby wartości zależności podzielić na niepytajne (niezależne) i inne (zależne) i dopiero w obrębie tych ostatnich prowadzić dalszy podział. Można tu wykorzystać technikę stosowaną w programie *Świgra* do reprezentacji hierarchii

rodzajów (zob. p. 5.3.2). Oczywiście taka zmiana wymagałaby analizy wykraczającej poza ramy tej pracy, warto ją chyba wykonać przy uspójnianiu interpretacji parametru zależności.

Parametr zależności w programie *Świgra*

W analizie komputerowej prowadzonej w porządku wstępującym dosłowne potraktowanie GFJP oznaczałoby, że dla każdej formy czasownikowej trzeba skonstruować 40 konstrukcji werbalnych właściwych i dalej prowadzić wariantywną ich rozbudowę do bardziej skomplikowanych fraz werbalnych i finitywnych, by na poziomie zdania elementarnego (w najlepszym razie) wybrać tylko te, które pasują w danym kontekście. Co więcej parametr zależności jest w zasadzie w GFJP przekazywany z góry na dół. Oznacza to, że odsiewanie konstrukcji z niepasującą wartością zależności odbywałoby się późno — w najbardziej niesprzyjającym wypadku, gdy kompletne zdanie równorzędne zostaje skonfrontowane z towarzyszącym mu znakiem końca (znakiem interpunkcyjnym, któremu też przysługuje wartość zależności, pytajna lub niepytajna). Ze względu na oczywistą nieefektywność takiego postępowania w programie *Świgra* reguły gramatyki zostały nieco przekształcone.

Parametr zależności zamiast przechowywać pojedynczą wartość jest zbiorem wszystkich wartości dopuszczalnych dla danej frazy. Zbiory te są odpowiednio przecinane zgodnie z warunkami gramatyki. W kompletnej analizie zwykle daje się ustalić jedną wartość zależności przysługującą danej konstrukcji. Jeżeli pozostanie kilka dopuszczalnych wartości, należy taki wynik rozumieć jako skrótowy zapis kilku możliwych drzew różniących się tylko zależnością. Ostatecznie obliczona wartość jest „zsyłana" wzdłuż całej gałęzi drzewa, aby służyła za etykietę dla wszystkich konstrukcji, zgodnie z literą GFJP. Parametr zależności jest więc w programie Świgra złożoną strukturą, która dopiero na potrzeby prezentacji wyniku analizy jest zamieniana na pojedynczą wartość.

Konkretnie na odpowiedniej pozycji wśród parametrów jednostki nieterminalnej stoi term o postaci z(*SwZ*, *Z*), gdzie *Z* jest listą bez powtórzeń wartości zależności, jakie są dopuszczalne dla danego wystąpienia jednostki nieterminalnej, natomiast *SwZ* jest obliczoną jedną wartością faktycznie przysługującą temu wystąpieniu (przez większą część procesu analizy ta wartość pozostaje nieustalona). Ta wartość w niektórych przypadkach również ma pewną strukturę wewnętrzną, dzieje się tak, gdy w pewnej gałęzi drzewa trzeba odtworzyć wartość zależności sprzed przekształcenia jej przez warunek oblzal lub oblnp.

Do reprezentacji tego parametru nie została wykorzystana technika omówiona w punkcie 3.6, ponieważ na wartościach tego parametru wykonywane są nie tylko operacje przecięcia, ale i sumy, więc konieczne byłoby i tak pozaunifikacyjne przetwarzanie wartości. Ponadto jak wspomniałem wyżej wartości tego parametru powinny zostać zrewidowane w ulepszonych wersjach gramatyki (por. również dyskusję Janusza S. Bienia w artykule (Bień 2003)).

W programie *Świgra* reguły (we30–35) zostały połączone w jedną regułę oznaczoną (we30n). Dzięki temu możliwe jest budowanie dla każdej formy czasownikowej tylko jednej konstrukcji werbalnej właściwej. Oto pierwsze trzy z oryginalnych reguł:

kwer1(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*) ⟶ (we30)
  formaczas(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*),
  { równe(*Z*, [np, npx, npxx]) }.

kwer1(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z*) ⟶ (we31)

formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Z***)**,
{ równe(*Z*, [npt, npxt, npxxt, p, px, pxx, pz]),
różne(*T*, roz) }.

kwer1(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z***)** ⟶ (we32)
formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Z***)**,
{ równe(*Z*, [aż1xx, aż2, bo, box, boxx, bowiem, chociaż, co, czy, dopóki, gdy, gdyxx, jak,
jaki, jeśli, kto, który, podczas, ponieważ, że]),
różne(*Wf*, bok),
różne(*T*, roz) }.

A tak przedstawia się odpowiadający im fragment reguły połączonej:

kwer1(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, z(_, *Z*)**)** ⟶ (we30n)
formaczas(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K***)**,
{ *Z* = [np, npx, npxx | *Z1*],
(*T* \== roz–>*Z1* = [npt, npxt, npxxt, p, px, pxx, pz | *Z2*]; *Z1* = *Z2*),
(*T* \== roz, *Wf* \== bok–>*Z2* = [aż1xx, aż2, bo, box, boxx, bowiem, ... | *Z3*]; *Z2* = *Z3*),
... }.

Lista dopuszczalnych wartości zależności *Z* jest konstruowana stosownie do wartości trybu i wyróżnika fleksyjnego formy czasownikowej. Wartości np, npx (czyli np'), npxx (np'') mogą przysługiwać każdej konstrukcji werbalnej właściwej. Jeżeli parametr trybu formy czasownikowej jest różny od roz (operator \== oznacza różność termów, a *If*–>*Then*;*Else* jest konstrukcją warunkową), to do listy dodawane są wartości npt, np't, np''t, p, p', p'' i pz. Lista dopuszczalnych wartości zależności jest podobnie rozbudowywana w następnych fragmentach tego warunku.

W regułach gramatyki, gdzie występowało uzgodnienie parametru zależności, wprowadzone zostały warunki służące do obliczania odpowiednich przecięć zbiorów. Na przykład pierwsza reguła gramatyki, w której parametr zależności zdania równorzędnego jest uzgadniany z parametrem jednostki **znakkonca**, ma w oryginale następującą postać:

wypowiedzenie ⟶ (w1)
zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, *Z***)**,
znakkonca(*Z***)**.

W programie *Świgra* w regule tej użyta jest relacja zrowne(*Z1*, *Z2*, *Z3*), która zachodzi, gdy lista bez powtórzeń *Z3* jest przecięciem list bez powtórzeń *Z1* i *Z2*:

wypowiedzenie ⟶ (w1)
zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, z(*SwZ*, *Z*), _**)**,
znakkonca(*Z1***)**,
{ zrowne(*Z*, [*Z1*], *SwZ*) }.

W wypadku tej reguły zbiór *Z* zależności dopuszczalny dla jednostki **zr** jest porównywany z jedną wartością *Z1* właściwą dla danego znaku końca. Oczywiście jeżeli przecięcie jest niepuste, to *SwZ* = *Z1* i ta właśnie wartość jest używana przy prezentacji wyników jako

wartość zależności dla **zr** i konstytuujących je jednostek. Jest to typowa reguła w której wartość zależności uzyskuje największy poziom uszczegółowienia. Oczywiście są też reguły, w których faktycznie przecinane są zbiory o większej liczbie elementów, np. reguły opisujące budowę zdania elementarnego.

Źródłem dodatkowych komplikacji jest warunek oblzal/3. W GFJP warunek ten występuje w regule (fi4) opisującej realizację frazy finitywnej właściwej przez frazę werbalną i w regule (e19) opisującej rekurencyjną realizację zdania elementarnego. Przy analizie wstępującej oznacza to, że wartość trzeciego argumentu *Ow* tego warunku jest nieustalona — pochodzi ona bowiem od ewentualnego spójnika w zdaniu prostym, którego składnikiem jest zdanie elementarne, którego centrum jest analizowana fraza finitywna. Warunki te trzeba więc było przemieścić do wszystkich reguł opisujących zdanie proste, gdzie wartość parametru *Ow* jest już znana. Na potrzeby wypisywania drzew analizy odtwarzana jest wartość zależności wynikająca z oryginalnych reguł. W tym celu zmienna *SwZ* w odpowiednich jednostkach jest unifikowanana z termem zawierającym oprócz wartości zależności pozostałe argumenty warunku oblzal, zaś procedura wypisująca odtwarza na tej postaci wyjściową wartość zależności.

5.3.2. Parametr rodzaju-liczby

Świdziński pisze w GFJP na s. 86:

> Rodzaj-liczba jest jednym parametrem. Interpretację taką, poważnie upraszczającą opis, umożliwia fakt, że w opisywanych tu konstrukcjach kategoria fleksyjna rodzaju i kategoria fleksyjna liczby nigdy nie wchodzą w uzgodnienia w sposób niezależny od siebie.

J. S. Bień wskazał (Bień 2003, s. 82) na nieoczywistość tego „poważnego uproszczenia". W zasadzie mamy tu do czynienia z dwoma osobnymi parametrami zapisywanymi w formie argumentów infiksowego funktora kropka.

Parametry te występują niezależnie w niektórych regułach elementarnych. Na przykład w regule opisującej analityczny czas przyszły, gdzie formie fleksemu bedzie przysługuje tylko wartość liczbowa, która musi się uzgodnić z wartością liczbową pseudoimiesłowu.

W GFJP występuje 6 wartości ustalonych kategorii rodzaju: męskoosobowy mos, męskozwierzęcy mzw, męskonieżywotny mnż, nijaki nij, żeński żeń, przymnogi pt.

Wartości rodzaju w wynikach analizy morfologicznej bardzo często są niejednoznaczne. W związku z tym program *Świgra* pracuje ze zbiorami „rodzajów elementarnych" a nie z pojedynczymi wartościami (por. 3.6). Oto termy reprezentujące poszczególne „rodzaje elementarne":

```
mos   r(mn(m(zy)),mo)
mzw   r(mn(m(zy)),nmo(m))
mnż   r(mn(m(nzy)),nmo(m))
nij   r(mn(n),nmo(np(n)))
żeń   r(nmn,nmo(z))
pt    r(nmn,nmo(np(p)))
```

i zbiory rodzajów:

mżyw	mos+mzw	r(mn(m(zy)),_))
mnos	mzw+mnż	r(mn(m(_)),nmo(m)))
m	mos+mzw+mnż	r(mn(m(_)),_))
nmo	mzw+mnż+nij+żeń+pt	r(_,nmo(_)))
np	nij+pt	r(_,nmo(np(_))))
mn	mos+mzw+mnż+nij	r(mn(_),_))

Oznaczenia wymienione w pierwszej kolumnie tabeli są używane w drzewach analizy. Oczywiście termy reprezentujące rodzaje elementarne w wynikach analizy są zastępowane symbolami wprowadzonymi przez Świdzińskiego.

5.3.3. Parametr inkorporacji

Parametr inkorporacji w zasadzie nie nastręcza trudności interpretacyjnych. Służy on do opisu zachowania spójników inkorporacyjnych BOWIEM, NATOMIAST, WIĘC, ZAŚ, których cechą charakterystyczną jest, że występują po pierwszym członie spajanej konstrukcji, a nie przed nią. Jak pisze Świdziński (s. 89):

> Inkorporacja jest prymarnie parametrem jednostek zdaniowych: informuje o tym, czy dana jednostka zdaniowa zawiera spójnik inkorporacyjny (nieinicjalny), czy też nie. Dana jednostka zdaniowa, która jest inkorporacyjna, czyli zawiera taki spójnik, ma pierwszy linearnie składnik inkorporacyjny. Inkorporacja zatem to parametr niemal wszystkich jednostek składniowych opisanych w tej pracy — w szczególności wszystkich typów szczegółowych fraz. Wartości ustalone inkorporacji to albo nazwa spójnika inkorporacyjnego (bowiem, natomiast, więc lub zaś), albo ni — wartość nieinkorporacyjna.

Dla zdań względnych parametr inkorporacji został wykorzystany przez Autora do zupełnie innego celu. Wiąże się to z tym, że zdanie względne nie może być inkorporacyjne. Natomiast dla tych zdań konieczne jest szczególne uzgodnienie rodzaju-liczby ze zdaniem nadrzędnym. Mianowicie rodzaj-liczba zaimka względnego musi być taki sam jak zdania nadrzędnego. Wartość ta może być oczywiście różna od rodzaju-liczby zdania podrzędnego (jeśli zaimek względny nie jest podmiotem zdania podrzędnego). Porównajmy:

(94) *Przyszedł chłopiec, którego znałam.*
(95) *Przyszedł chłopiec, który znał gramatykę.*
(96) *Przyszedł chłopiec, którego matka znała gramatykę.*
(97) **Przyszedł chłopiec, którą znałem.*
(98) *[!]*Przyszedł chłopiec, który znała gramatykę.*
(99) **Przyszedł chłopiec, która znała gramatykę.*

Zatem rodzaj-liczba zaimka względnego musi zostać przekazany do zdania nadrzędnego niezależnie od rodzaju-liczby całego zdania. Świdziński wykorzystał do tego celu parametr inkorporacji.

W mechanizm ten wkradł się błąd, który sprawia, że zdanie (98) jest akceptowane przez gramatykę. Mianowicie w regule (no41) parametr rodzaju-liczby *Rl* jednostki **knoink** pozostaje nieustalony:

knoink(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl***)** ⟶ (no41)
  knom(*P*, *I*, *O*, *Neg*, *Z*, *Kl***)**,
  { rozne(*I*, [bo, natomiast, ni, więc, zaś]),
  rowne(*Z*, [co, jaki, kto, który]) }.

To sprawia, że jeśli ta fraza miałaby być podmiotem zdania podrzędnego, jej rodzaj-liczba nie uzgodni się z odpowiadającym parametrem frazy finitywnej. Naprawa sytuacji polega na utożsamieniu parametrów *Rl* i *I* w tej regule:

knoink(*P*, *Rl*, *O*, *Neg*, *Rl*, *Z*, *Kl***)** ⟶ (no41)
  knom(*P*, *Rl*, *O*, *Neg*, *Z*, *Kl***)**,
  { rozne(*Rl*, [bo, natomiast, ni, więc, zaś]),
  rowne(*Z*, [co, jaki, kto, który]) }.

Nie spowoduje to odrzucenia zdania (94) ani (96), ponieważ w tych zdaniach zaimek względny nie jest centrum frazy podmiotowej i jego rodzaj nie uzgadnia się z rodzajem frazy finitywnej.

5.3.4. Parametr korelatywności

Świdziński następująco charakteryzuje parametr korelatywności (GFJP, s. 91):

> Przysługuje on przede wszystkim frazie zdaniowej. Fraza zdaniowa wymagana może mianowicie zawierać korelat, to znaczy element to, tego, temu, tym, poprzedzony, być może, przyimkiem. To, czy fraza zdaniowa zawiera korelat, czy nie, zależy od czasownika, który takiej frazy wymaga. Parametr korelatywności przypisuję frazie werbalnej, a więc i frazie finitywnej, oraz frazie wymaganej — bo fraza zdaniowa wymagana jest realizacją tej ostatniej. Wartości ustalone to albo wartość przypadka, albo oznaczenie przyimka, albo wartość niekorelatywna (nk) — w wypadku fraz werbalnych nie wymagających frazy zdaniowej lub wymagających frazy zdaniowej bez korelatu. Z powodów technicznych, które będą omówione w punktach 8.1–8.4, wprowadzam jeszcze jedną specjalną wartość parametru korelatywności — wartość bezprzecinkową (bp). Wartość ta nie jest nigdy wymagana przez frazę werbalną.

To ostatnie zdanie nie znajduje żadnego odzwierciedlenia w regułach. Określenie „nie jest nigdy wymagana" należy czytać „jest niedopuszczalna jako realizacja wymagania". Tymczasem w GFJP możliwa jest na przykład analiza niepoprawnego zdania

(100) *Wiem że czekali.*

w której *że czekali* jest bezprzecinkową realizacją zdaniowej frazy wymaganej. Dzieje się tak dlatego, że reguła (wy19) dopuszcza wartość bp korelatywności. Możliwa jest też realizacja bezprzecinkowej zdaniowej frazy luźnej, przepuszczanej przez regułę (lu19). W odniesieniu do frazy wymaganej można by powiedzieć, że to wina słownika wymagań czasownikowych, który nie powinien dopuszczać takiej wartości wymagania i że po przygotowaniu odpowiedniego słownika uwzględniającego korelatywność problem zniknie. Nie dotyczy to jednak fraz luźnych. Ponadto fraza zdaniowa o wartości korelatywności różnej od bp i nk (a więc faktycznie zawierająca korelat) może stanowić jedynie realizację frazy wymaganej. Wynika

z tego, że jedyną dopuszczalną wartością korelatywności dla frazy zdaniowej realizującej frazę luźną jest nk.

Podobnie jest z frazami zdaniowymi stanowiącymi składnik fraz nominalnych, przymiotnikowych i przysłówkowych. Dopuszczenie w tych regułach dowolnej wartości korelatywności prowadzi do akceptowania przez GFJP na przykład następujących wypowiedzeń z interpretacją wyróżnionego fragmentu jako frazy zdaniowej:

(101) *![1]*Znam chłopca* ***który czyta***.
(102) *![1]*On jest taki* ***to, jakby nas nie lubił***.
(103) *![1]*Mówił tak* ***o to, jakby był głodny***.

Fraza zdaniowa występuje jako składnik również w realizacjach zdania prostego **zp** i również tam jedyną dopuszczalną wartością jest nk.

W programie *Świgra* wykluczyłem wartość bp parametru K w regule (wy19). Ponadto ustaliłem wartość tego parametru na niekorelatywną nk w regułach: (lu15–lu19), (no2–no4), (p2–p8), (pt2), (ps2).

Korelat pojawia się w strukturze wypowiedzenia w jednej z reguł opisujących jednostkę **fzdkor** (frazę zdaniową z korelatem):

fzdkor(*Tfz*, *K*, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd38)
  kor(*K*, *I***)**,
  przec,
  fzde(*Tfz*, *A*, *C*, *T*, *Neg*, ni**)**.

fzdkor(*Tfz*, nk, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd39)
  przecsp,
  fzde(*Tfz*, *A*, *C*, *T*, *Neg*, *I***)**.

fzdkor(*Tfz*, nk, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd40)
  fzde(*Tfz*, *A*, *C*, *T*, *Neg*, *I***)**.

Parametr ten dla fraz zdaniowych oprócz przekazania informacji o obecności korelatu, wskazuje również, czy fraza zawiera na początku przecinek. Jeżeli tak, korelatywność ma wartość nk, jeśli nie — wartość bp. Ta ostatnia wartość oznacza, że daną frazę zdaniową poprzedza spójnik, przed którym stoi przecinek (w związku z czym nie ma przecinka przed samą frazą).

Jednak tego rozróżnienia zabrakło w przytoczonych regułach, w związku z czym GFJP akceptuje zdania z brakującym przecinkiem przed frazą zdaniową, na przykład:

(104) *![1]*Przyszedł chłopiec który czytał.*

Jak ustaliłem w dyskusji z Autorem, reguła (zd40) zawiera błąd techniczny — wartość korelatywności powinna być bp:

fzdkor(*Tfz*, bp, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd40)
  fzde(*Tfz*, *A*, *C*, *T*, *Neg*, *I***)**.

(Wartość ta zapewne „zawędrowała” z reguły (zd41) (GFJP s. 419), w której jednostka **fzdkor** powinna uzgadniać korelatywność z **fzd**.)

Wśród reguł GFJP opisujących realizacje korelatu **kor** zabrakło reguły opisującej korelat zawierający przyimek, ale nieinkorporacyjny. Regułę taką dodałem w programie:

kor(*K*, ni**)** ⟶ (kor1x)
  przyimek(*K*, *P***)**,
  kor1(*P***)**.

5.3.5. Parametr negacji

Ciekawą cechą parametru negacji jest to, że informuje on nie tylko o tym, czy dana fraza jest zaprzeczona, ale także o źródle tego zaprzeczenia. Wyróżnionym rodzajem zaprzeczenia jest negacja pochodząca od spójnika ANI, która powoduje szczególne ograniczenia łączliwości jednostek, GFJP, s. 88:

> Negacja jest parametrem frazy werbalnej: informuje o zaprzeczoności lub niezaprzeczoności jej centrum — formy czasownikowej, tzn. o obecności wewnątrz tej frazy formy partykuło-przysłówka NIE bezpośrednio przed formą czasownikową lub o braku takiej formy. Parametr negacji, przysługujący wszystkim jednostkom zdaniowym, frazie finitywnej i frazie zdaniowej, formalizuje tę samą opozycję. Istnieją jednostki słownikowe, które mają z definicji wartość ustaloną tego parametru, mianowicie zaimki negatywne (rzeczowne, przymiotne i przysłowne). Również frazie wymaganej, frazie luźnej i wszystkim typom szczegółowym fraz: frazie przyimkowej, frazie nominalnej, frazie przymiotnikowej i frazie przysłówkowej — przysługuje parametr negacji. Służy on różnym celom — na przykład opisaniu wpływu przeczenia na rząd czasownika czy ograniczeń negacji wewnątrz fraz zawierających zaimek negatywny jako składnik (niekoniecznie bezpośredni). Wartości ustalone negacji to: zaprzeczona (ani lub nie; tę pierwszą nazwać można zaprzeczonością odspójnikową, tzn. narzuconą przez kontekst spójnika szeregowego ANI) i niezaprzeczona (tak).

Wymienione trzy wartości parametru negacji w programie *Świgra* mają pewną strukturę. Mianowicie reprezentowana jest opozycja między wartością tak i pozostałymi dwiema. W lewej kolumnie tabeli wymieniono wartości występujące w GFJP, w prawej odpowiadające im termy:

tak	tak
nie	nie(nie)
ani	nie(ani)
{nie, ani}	nie(_)

Zabieg ten jest potrzebny na przykład ze względu na następującą regułę GFJP:

kweneg(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *Neg*, *I*, *Z***)** ⟶ (we21)
  part(nie**)**,
  kweink(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wa*, *Wb*, *Wc*, *K*, *I*, *Z***)**,
  fl(*A*, *C*, *RI*, *O*, nie, ni, np**)**,
  { rowne(*Neg*, [ani, nie]),
  rozne(*Z*, [p, px, pxx, pz]) }.

Przy analizie prowadzonej wstępująco w regule tej nadaje się parametrowi *Neg* zbiór dwóch wartości zaprzeczonych (lub też musiałyby być tworzone dwie interpretacje frazy). Rozstrzygnięcie, o którą z nich chodzi, nastąpi dopiero na poziomie zdaniowym, gdzie być może zainterweniuje spójnik ANI. W programie w tym miejscu parametrowi *Neg* nadawana jest wartość nie(_) reprezentująca niedookreśloną zaprzeczoność.

5.3.6. Parametr typu frazy zdaniowej

Oto opis tego parametru ze s. 91 GFJP:

> Typ frazy zdaniowej jest parametrem frazy zdaniowej. W niniejszym opisie rozróżniam frazy zdaniowe spójnikowe (wartości ustalone parametru typu frazy zdaniowej to oznaczenia spójników podrzędnych początkowych: aż1, aż2, chociaż, choćby, czy, dopóki, gdy, gdyby, jak, jakby, jakoby, jeśli, podczas, ponieważ, zanim, że, żeby), pytajnozależne (pz), względne (co, jaki, kto, który) oraz mieszane (mie1, mie2, mie3). Niektóre frazy zdaniowe spójnikowe, wszystkie frazy zdaniowe pytajnozależne oraz mieszane są wymagane przez frazę werbalną; opisuję je tutaj jako realizacje frazy wymaganej.

Warto zwrócić uwagę, na mieszane realizacje fraz zdaniowych. Mianowicie w regułach (zd45–50) opisano frazy zdaniowe typu mie1, mie2 i mie3.

fzde(mie1, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd45)
  spoj(po, że, *I***)**,
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, ni, że**)**.

fzde(mie1, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd46)
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, pz**)**.

fzde(mie2, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd47)
  spoj(po, że, *I***)**,
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, ni, że**)**.

fzde(mie2, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd48)
  spoj(po, żeby, ni**)**,
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, *I*, żeby**)**.

fzde(mie3, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd49)
  spoj(po, gdy, *I***)**,
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, ni, gdy**)**.

fzde(mie3, *A*, *C*, *T*, *Neg*, *I***)** ⟶ (zd50)
  spoj(po, jak, *I***)**,
  zr(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Neg*, ni, jak**)**.

Intencja opisu jest taka, że frazy mieszane mają składać się z połączonych współrzędnie fraz zdaniowych różnych typów. Na przykład czasownik MÓWIĆ dopuszcza frazę zdaniową typu że, typu żeby, a także typu mie2 (występujące odpowiednio w poniższych przykładach):

(105) *Mówiono nam**, że będzie przyjęcie***.
(106) *Mówiono nam**, żeby przyjść***.

(107) *Mówiono nam**, żeby przyjść oraz że będzie przyjęcie**.*

Wartości mie1, mie2 i mie3 są oczywiście przypisywane czasownikowi w słowniku wymagań czasownikowych.

Niestety według reguły (zd47) sama fraza *, że będzie przyjęcie* jest frazą zdaniową typu mie2 więc fraza zdaniowa w zdaniu (105) może być przypisana zarówno do typu że jak i mie2 co stwarza nadmiarową analizę. Być może zatem intencją Autora było pominięcie fzd(że) wśród wymagań czasownika mówić. Niestety i takie założenie nie pozwala uniknąć nadmiarowych interpretacji, ponieważ omawianej frazie można przypisać również typ mie1, a czasownik mówić ewidentnie dopuszcza zarówno typ mie1 jak i mie2. Poprawienie tego błędu wymagałoby przeniesienia opisu zjawiska na poziom fraz zdaniowych równorzędnych i opisanie ich jako złożonych z faktycznie różnych składników.

5.4. Frazy luźne

Fraza luźna **fl** to każda fraza w strukturze zdania elementarnego, która nie jest argumentem czasownika. Do tej kategorii należą elementy, które są „luźne" w sensie pomijalności. Ale fraza luźna może być także koniecznym elementem danej struktury, tyle że nie wymaganym przez czasownik. Na przykład inicjalnej pytajnej frazy luźnej nie da się usunąć z pytajnego zdania elementarnego nie psując jego pytajności. Nieciągły wariant szyku czasu przeszłego czasowników w zdaniu podrzędnym jest realizowany z pomocą inicjalnej frazy luźnej aglutynacyjnej, opisanej regułą (lu8) (por. rys. 5.2 i C.7).

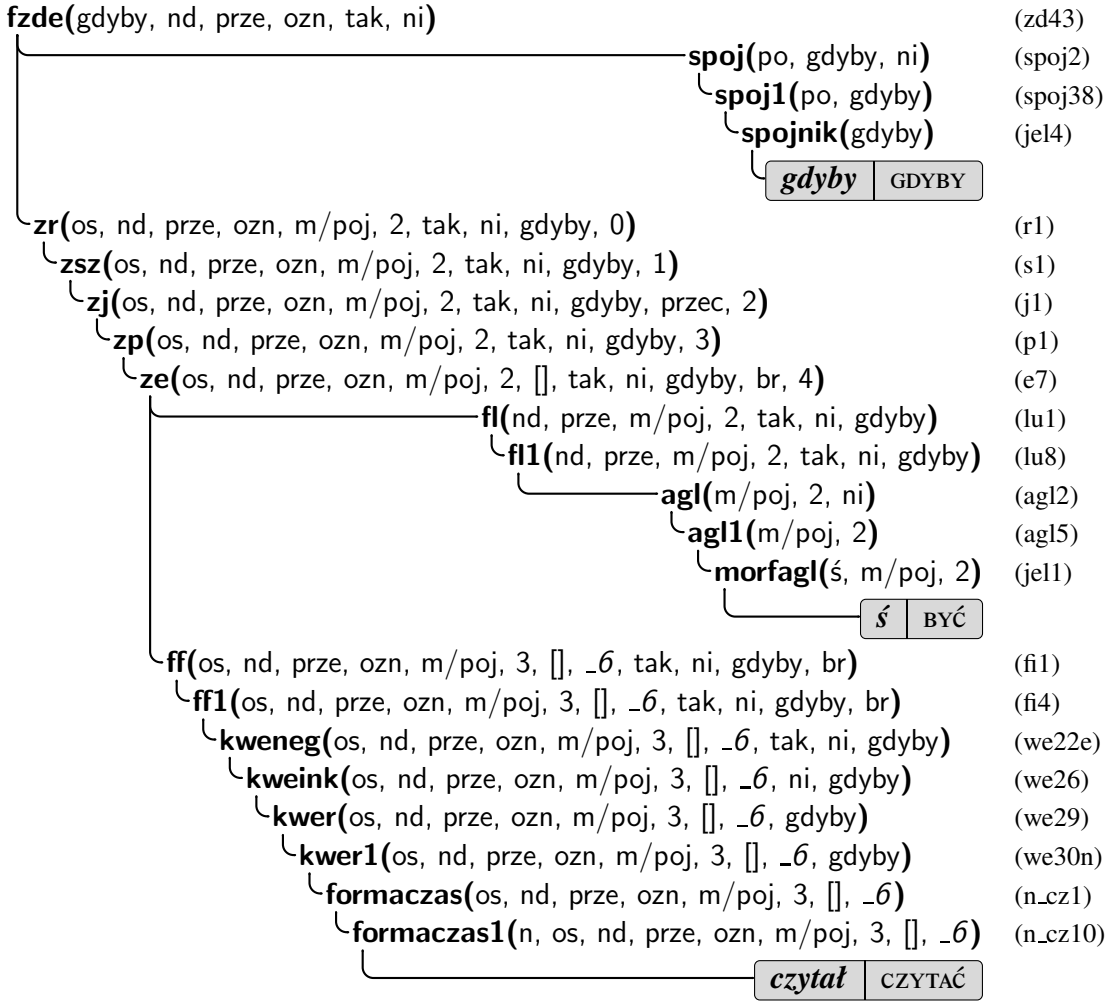
Świdziński uzyskuje w ten sposób jednorodność składu realizacji standardowych zdania elementarnego. Składa się ono zawsze z frazy finitywnej i ewentualnych fraz wymaganych i luźnych. Jednorodność ta jest jednak zupełnie powierzchowna — na przykład pojęcie aglutynacyjnej frazy luźnej zostało stworzone na potrzeby tego opisu i nie niesie żadnej powszechnie zrozumiałej intuicji, aglutynant został niejako siłą wtłoczony w rolę frazy luźnej. Ponadto zabieg ten nie daje oszczędności w liczbie reguł, ponieważ każdemu rodzajowi obligatoryjnej frazy luźnej poświęcono osobną regułę na poziomie zdania elementarnego. Jak sądzę, bardziej czytelnym rozwiązaniem byłoby jawne wprowadzenie w zdaniu elementarnym fraz niższego poziomu.

Oprócz wystąpień fraz luźnych jako elementów zdania elementarnego (ściślej: jako składników bezpośrednich frazy finitywnej i wymaganej) Świdziński przewidział obecność frazy luźnej jako składnika konstrukcji werbalnej z negacją **kweneg**, a więc wewnątrz frazy werbalnej. Jest ona potrzebna w zdaniach, w których fraza werbalna zawiera w sobie frazę wymaganą, na przykład:

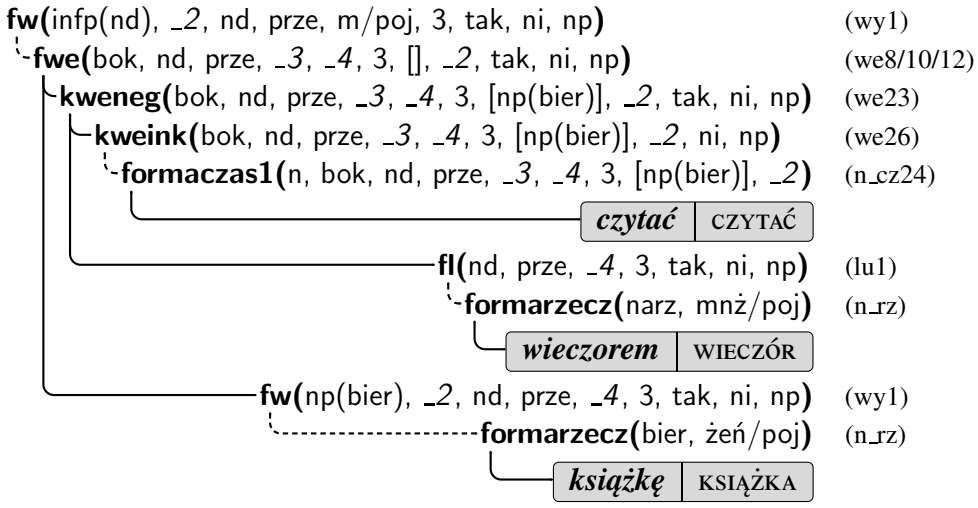
(108) *Chciał **czytać wieczorem książkę**.*
(109) *Odpoczywał, **czytając wieczorem książkę**.*

Wówczas bowiem fraza luźna (w przykładach *wieczorem*) nie może zostać pochłonięta jako składnik frazy finitywnej, jako że znajduje się między konstrukcją werbalną (*czytać, czytając*) a wymaganą przez nią frazą nominalną (*książkę*), por. rys. 5.3. Jednak w wypadku fraz finitywnych powoduje to możliwość przyczepienia frazy luźnej zarówno na poziomie konstrukcji werbalnej z negacją, jak i frazy finitywnej. Sprawia to, że zdanie

(110) *Czytałem wieczorem książkę.*



Rys. 5.2. Drzewo analizy frazy zdaniowej elementarnej *gdybyś czytał*



Rys. 5.3. Drzewo analizy frazy wymaganej *czytać wieczorem książkę*

będzie miało dwie interpretacje. Świdziński nie interpretuje fraz luźnych jako przynależnych do konkretnej frazy nadrzędnej, por. GFJP s. 65:

> Dość daleką konsekwencją prezentowanej tu implikacyjnej orientacji analizy składnikowej jest sposób potraktowania członów luźnych, a więc m. in. tradycyjnych okoliczników. Będę je interpretować jako składniki bezpośrednie jednostki typu zdaniowego lub pewnych typów fraz, i to bez względu na interpretację semantyczną.

Należy więc widzieć frazy luźne jako niepowiązane z resztą struktury wypowiedzenia fragmenty, których miejsce „zaczepienia" jest sprawą czysto techniczną. W związku z tym przedstawiona wielość interpretacji jest nadmiarowa. W programie *Świgra* ograniczyłem możliwość obecności frazy luźnej wewnątrz konstrukcji werbalnej z negacją (reguły (we20–24)) do fraz werbalnych niefinitywnych.

Dodatkowym efektem, powodującym powstanie interpretacji, które często są nadmiarowe (por. p. 6.5) jest to, że wiele fraz może konstytuować zarówno frazę wymaganą, jak i luźną. Frazą luźną i frazą wymaganą może być fraza przyimkowa, nominalna (z wyjątkiem mianownikowej[3] i wołaczowej), przysłówkowa. Niestety w tym wypadku nie widać sposobu ograniczenia wielości interpretacji bez odwołania się do analizy semantyki wypowiedzenia.

5.5. Reguły o pustych prawych stronach

W związku z przyjętą strategią analizy konieczne było wyeliminowanie z gramatyki reguł o pustych prawych stronach. Wymagało to dopisania wariantowych wersji niektórych z reguł, w których mogły wystąpić jednostki o pustej realizacji.

W GFJP jest dziesięć reguł o pustej prawej stronie: (wy5), (lu14), (agl7), (int5), (int7–12). Z tego ostatnie siedem stanowią reguły opisujące puste realizacje przecinków w określonych kontekstach, które omawiam w punkcie 5.6. W bieżącym punkcie opisuję sposób uwzględnienia pozostałych reguł skracających.

Najprostsza jest sprawa reguły (wy5). Opisuje ona pustą realizację frazy wymaganej **fw**, potrzebną do realizacji wymagań typu nic i zdań eliptycznych. Reguła ta jest zbędna w analizatorze *Świgra* ze względu na zmieniony sposób obsługi wymagań czasownikowych, opisany szczegółowo w p. 5.2.

Reguła (lu14) opisuje pustą realizację frazy luźnej **fl**. Fraza luźna występuje jako składnik frazy finitywnej (reguły (fi2–3)), wymaganej (wy2–4) i luźnej (lu2–4), we wszystkich tych wypadkach osobno uwzględniona jest jednak w regułach realizacja bez frazy luźnej. Usunięcie reguły (lu14) tylko poprawia ten fragment opisu — w postaci oryginalnej bowiem dla każdej frazy wymienionych typów, nie zawierającej frazy luźnej, da się skonstruować dwa drzewa: jedno z użyciem reguły pomijającej frazę luźną i drugie z frazą luźną pustą.

Ponadto fraza luźna występuje jako inicjalny składnik zdania elementarnego **ze** w regułach (e1), (e4), (e7–8), (e16) i (e18). Te reguły dzielą się na dwie grupy: w jednej grupie dopuszczalne są jedynie frazy luźne o wartości zależności różnej od np, które nie mogą być realizowane przez frazę pustą (reguła (lu14) ma ustaloną wartość zależności np). Reguły drugiej grupy dopuszczają wartość np ale za to istnieją analogiczne reguły bez frazy luźnej

[3] Reguła (lu6) dopuszcza frazy mianownikowe. Jest to ewidentny błąd, potwierdzony przez Autora i poprawiony w programie *Świgra*.

(np. dla reguły (e1) są to reguły (e2) i (e3)). Tak więc usunięcie reguły (lu14) nie zakłóca pracy tych reguł.

Ostatnim miejscem występowania fraz luźnych są reguły (we20–24) opisujące konstrukcję werbalną z negacją **kweneg**. W tym wypadku konieczne było dopisanie wariantów reguł bez frazy luźnej. W gramatyce programu *Świgra* są one oznaczone (we20e–24e):

kweneg(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, nie(nie), *I*, z(*SwZ*, *NZ*)**)** ⟶ (we20e)
  partykula(nie**)**,
  kweink(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, *I*, z(*SwZ*, *Z*)**)**,
  { zrowne(*Z*, [p, px, pxx, pz], *NZ*) }.

kweneg(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, nie(_), *I*, z(*SwZ*, *NZ*)**)** ⟶ (we21e)
  partykula(nie**)**,
  kweink(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, *I*, z(*SwZ*, *Z*)**)**,
  { zrozne(*Z*, [p, px, pxx, pz], *NZ*) }.

kweneg(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, tak, *I*, z(*SwZ*, *NZ*)**)** ⟶ (we22e)
  kweink(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, *I*, z(*SwZ*, *Z*)**)**,
  { zrozne(*Z*, [aż1xx, boxx, byxx, dopóki, gdyxx, zanimxx], *NZ*) }.

kweneg(*Wf*, nd, *C*, *T*, *RI*, *O*, *Wym*, *K*, tak, *I*, z(*SwZ*, *NZ*)**)** ⟶ (we24e)
  kweink(*Wf*, *A*, *C*, *T*, *RI*, *O*, *Wym*, *K*, *I*, z(*SwZ*, *Z*)**)**,
  { zrowne(*Z*, [aż1xx, boxx, byxx, dopóki, gdyxx, zanimxx], *NZ*) }.

(Reguły odpowiadające (we22) i (we23) są zapisane łącznie w postaci jednej.)

Ostatnia z omawianych reguł, (agl7) opisuje pustą realizację aglutynatu właściwego o opisie **agl1(***RI*, 3**)**. Realizacja ta może wystąpić w regułach (agl1) i (agl2).

Możliwość wystąpienia pustego aglutynatu w regule (agl1) oznacza konieczność dodania reguły (agl1e), opisującej spójnik inkorporacyjny jako trzecioosobową inkorporacyjną realizację aglutynantu.

Na mocy reguły (agl2) możliwa jest pusta realizacja aglutynatu o opisie **agl(***RI*, 3, ni**)**. Jednostka ta występuje po prawej stronie tylko w jednej regule (lu8) i pozwala na pustą realizację frazy luźnej właściwej o opisie **fl1(***A*, *C*, *RI*, 3, *Neg*, ni, *Z***)** gdzie *Z* ma jedną z wartości by",choćby,czyżby,gdyby,jakby,jakoby,żeby. Fraza luźna o takiej wartości zależności może być wyłącznie składnikiem bezpośrednim zdania elementarnego opisanego przez regułę (e7). Realizacja taka składa się z inicjalnej frazy luźnej oraz frazy finitywnej i fraz wymaganych w dowolnym porządku. Aby uwzględnić możliwość pustej realizacji tej frazy luźnej musiałem dodać dwie reguły (o symbolach (e7e1) i (e7e2)) podobne do (e5) i (e6). W regułach tych wartość osoby jest ustalona na 3, wartość zaś inkorporacyjności na ni, dla zależności dopuszczalne są wartości wymienione wyżej.

ze(*Wf*, *A*, *C*, *T*, *RI*, 3, *Wym*, *Neg*, ni, z(*SwZ*, *NZ*), *Ow*, @@@@0**)** ⟶ (e7e1)
  fw(*W0*, *K*, *A*, *C*, *RI*, 3, *Neg*, ni, z(*SwZ3*, *Z3*)**)**,
  { zrowne(*Z3*, [np], *SwZ3*) },
  wymagania([*W0*], *Wym*, *Reszta Wym*,
    ff(*Wf*, *A*, *C*, *T*, *RI*, 3, *Wym*, *K*, *Neg*, ni, z(*SwZ*, *Z*), *Ow***)**,
    [*W1*/**fw(***W1*, *K*, *A*, *C*, *RI*, 3, *Neg*, ni, **z(***SwZ1*, *Z1***))**,
     W2/**fw(***W2*, *K*, *A*, *C*, *RI*, 3, *Neg*, ni, **z(***SwZ2*, *Z2***))**]**)**,
  { resztawym(*Reszta Wym*),

zrowne(*Z*, ['byxx', 'choćby', 'czyżby', 'gdyby', 'jakby', 'jakoby', 'żeby'], *NZ*),
zrowne(*Z1*, [np], *SwZ1*),
zrowne(*Z2*, [np], *SwZ2*) }.

ze(*Wf*, *A*, *C*, *T*, *Rl*, 3, *Wym*, *Neg*, ni, z(*SwZ*, *NZ*), *Ow*, @@@@0**)** ⟶ (e7e2)
ff(*Wf*, *A*, *C*, *T*, *Rl*, 3, *Wym*, *K*, *Neg*, ni, z(*SwZ*, *Z*), *Ow***)**,
{ zrowne(*Z*, ['byxx', 'choćby', 'czyżby', 'gdyby', 'jakby', 'jakoby', 'żeby'], *NZ*) },
wymagane(*Wym*, *ReszteWym*,
[*W1*/**fw(***W1*, *K*, *A*, *C*, *Rl*, 3, *Neg*, ni, **z(***SwZ1*, *Z1***))**,
W2/**fw(***W2*, *K*, *A*, *C*, *Rl*, 3, *Neg*, ni, **z(***SwZ2*, *Z2***))**,
W3/**fw(***W3*, *K*, *A*, *C*, *Rl*, 3, *Neg*, ni, **z(***SwZ3*, *Z3***))**]**)**,
{ resztawym(*ReszteWym*),
zrowne(*Z1*, [np], *SwZ1*),
zrowne(*Z2*, [np], *SwZ2*),
zrowne(*Z3*, [np], *SwZ3*) }.

W sumie eliminacja reguły (agl7) wymagała dodania reguły opisującej realizację aglutynatu i dwóch reguł opisujących realizacje zdania elementarnego.

5.6. Przecinki

W GFJP występują reguły kontekstowe pozwalające na pustą realizację jednostki **przec** (przecinka składniowego) w prawostronnym kontekście dowolnego znaku przestankowego (int7–11) lub w lewostronnym kontekście początku wypowiedzenia (int12). Ponadto jednostka **przecsp** (przecinek pospójnikowy) może mieć pustą realizację w lewostronnym kontekście spójnika.

Niestety zwykła realizacja przecinka nie jest obwarowana w GFJP żadnymi ograniczeniami. Oznacza to, że reguła opisująca realizację jednostki **przec** w postaci przecinka faktycznie pobranego z wejścia może zostać zastosowana również w wypadku, gdy intencją Autora gramatyki było użycie reguły kontekstowej. Z tego powodu GFJP zaakceptuje na przykład oba poniższe zdania:

(111) *[!]*Przyszedł chłopiec, dziewczyno, .*
(112) *Przyszedł chłopiec, dziewczyno.*

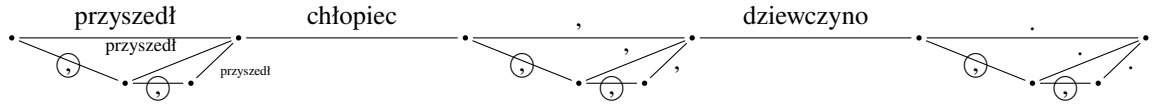
Jak napisałem w punkcie 5.5, w związku z przyjętą strategią analizy eliminuję z gramatyki reguły skracające. W wypadku przecinków dodatkowe utrudnienie stanowi kontekstowość. W regułach tych występuje w szczególności kontekst lewostronny, który nie jest dopuszczalny w znanych mi realizacjach DCG.

Dla tych kilku reguł nie warto było wprowadzać do analizatora ogólnego mechanizmu kontekstowości. Przede wszystkim trzeba by zarazem rozwiązać problem pustych prawych stron reguł. Ponadto uważam, że opis kwestii interpunkcyjnych w GFJP i tak wymaga przeformułowania, zapewne w sposób podobny do stosowanego w regułach opisujących frazy zdaniowe. W regułach tych konieczność pojawienia się przecinka na początku frazy jest sygnalizowana przez specjalną wartość parametru (por. punkt 5.3.4 o korelatywności). Ponieważ obecna wersja gramatyki opisuje już zjawisko o charakterze pozycyjnym — spójniki inkorporacyjne, więc większość jednostek gramatyki niesie wśród parametrów informację

o swoim inicjalnym elemencie. Przekonstruowanie tego mechanizmu tak, aby uwzględnić informację o koniecznej obecności przecinka na początku lub końcu danej jednostki nie powinno być trudne.

W programie *Świgra* przyjąłem więc rozwiązanie, które można uznać za tymczasowe i które nie jest w pełni zadowalające z punktu widzenia teoretycznego. Mianowicie pustą realizację jednostki **przec** osiągam poprzez dodanie pewnych elementów do grafu wejścia.

Na przykład dla wypowiedzenia (112) graf morfologiczny po modyfikacji prezentuje się następująco (prezentujemy tylko segmenty):



Łuki opatrzone symbolem ⟨,⟩ reprezentują puste realizacje przecinka. Ich opis morfologiczny wygląda następująco: segment jest pusty, identyfikator leksemu to atom przecinek (','), znacznik morfosyntaktyczny — interp. Z poziomu reguł element ten jest dostępny jako term morf('', ',', interp). Przy takiej strukturze wejścia wystarczy jedna reguła opisująca jednostkę **przec**:

przec ⟶ (int13)
[morf(_, ',', interp)**]**.

Dopuszcza ona realizację tej jednostki zarówno przez faktyczny przecinek, który ma segment ',', jak i przez przecinek dodany o segmencie pustym.

Łuki pozwalające na pustą realizację przecinka są dopisywane do każdego łuku reprezentującego znak interpunkcyjny (to symuluje reguły z prawostronnym kontekstem znaku interpunkcyjnego) i do pierwszego łuku w wypowiedzeniu (lewostronny kontekst początku wypowiedzenia).

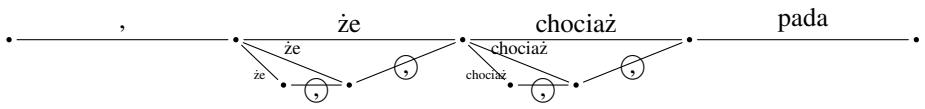
Pustą realizację jednostki **przecsp** opisuję podobnie. Mianowicie do każdego łuku w grafie wejścia, reprezentującego jeden ze spójników z listy w regule (int5), dopisuję łuki z opisem morf('', '', przecsp). Łuki te pasują do prawej strony reguły (int5), która w programie ma postać następującą:

przecsp ⟶ (int5)
[morf('', '', przecsp)**]**.

Pusta realizacja jednostki **przecsp** jest potrzebna na przykład w wypowiedzeniu (113) (przykład Świdzińskiego):

(113) *Wiem, że chociaż pada, Piotr umarł.*
(114) *[!]*Wiem, że, chociaż pada, Piotr umarł.*

Po dopisaniu łuków przecsp fragment grafu wejścia dla tego wypowiedzenia wygląda następująco:



Zauważmy, że tak dopisane łuki dają rozsądne zachowanie programu w przypadku niejednoznaczności morfologicznej segmentu spójnikowego (np. *nim*) — dopisany łuk jest

w sposób konieczny kojarzony z poprzedzającym go łukiem stanowiącym interpretację spójnikową.

Przedstawiony mechanizm nie daje pełnej zgodności z literą GFJP — aby tak było, trzeba by dopisać nieskończoną liczbę przecinków. W istocie liczbę potrzebnych przecinków można by oszacować z góry, ponieważ każdy pusty przecinek musi znaleźć swoje miejsce na jakimś poziomie hierarchii jednostek GFJP, nie jest to jednak proste ze względu na rekurencję (trzeba by uwzględniać długość wypowiedzenia i liczbę sensownych przejść przez reguły rekurencyjne).

Obecnie program dopisuje w każdym miejscu po dwa puste przecinki, co wystarcza do analizy przykładów Świdzińskiego, więc można oczekiwać, że tym bardziej wystarcza w wypowiedzeniach, z którymi można faktycznie spotkać się w tekstach polskich.

Świgra podobnie jak GFJP akceptuje zdanie (111) i (114). Natomiast jednostkę **przecsp** *Świgra* traktuje odrobinę subtelniej niż GFJP, mianowicie nie jest dopisywany łuk z opisem przecsp, jeżeli następny element wejścia jest znakiem interpunkcyjnym. W takim bowiem wypadku powstawałyby dwa drzewa analizy: jednostka **przecsp** mogłaby mieć pustą realizację zarówno jako pospójnikowa jak i jako przedinterpunkcyjna.

Wadą przedstawionego mechanizmu jest to, że psuje on nieco efektywność analizy, ponieważ program musi rozpatrywać warianty jednostek kończące się faktycznym lub dodanym przecinkiem.

5.7. Podsumowanie

Profesor Świdziński zastrzega się w książce, że pisząc gramatykę nie miał na celu możliwości realizacji komputerowej. Przedstawioną przez niego gramatykę należy więc traktować jako pewną teorię lingwistyczną. Jak argumentuję w punkcie 5.1, Świdziński próbuje opisać zbyt szczegółowe jednostki, różniące się na tyle subtelnymi własnościami, że są one nie do wyrażenia w formalizmie gramatyk metamorficznych. W wyniku tego dosłownie traktowana GFJP każdemu akceptowanemu wypowiedzeniu przypisuje nieskończenie wiele drzew, co nie pokrywa się chyba z intencjami Autora.

Można sformułować jeszcze kilka wątpliwości, co do zasad, według których prowadzony jest opis. Nie jest jasne wedle jakiej zasady jednostce nieterminalnej przysługują lub nie przysługują pewne parametry, czy parametry opisują funkcję pełnioną przez jednostkę (a więc cechę zależną od kontekstu, w którym się ona znalazła), czy jej skład wewnętrzny. Omówię pokrótce te problemy.

Jakie parametry

Można chyba przyjąć, że Świdziński wyposaża jednostkę składniową w pewien parametr wtedy, gdy trzeba odnotować jakąś cechę tej jednostki, podlegającą uzgodnieniu z innymi jednostkami. Jednak niektóre jednostki występujące w GFJP mają parametry, które nie biorą udziału w uzgodnieniach.

Na takiej „dekoracyjnej” zasadzie zdaniu elementarnemu przysługują parametry *Wa*, *Wb* i *Wc*. Nie biorą one udziału w żadnych uzgodnieniach. Uzgodnienia następują wewnątrz zdania elementarnego, pomiędzy jego składnikami. Widać to w szczególności w regułach (e14–e19) w których zdanie elementarne jest realizowane rekurencyjnie przez zdanie równorzędne, w związku z czym parametry wymagań mają zawsze wartość nieustaloną.

Następujące stwierdzenie Świdzińskiego ze strony 168 GFJP należałoby chyba sformułować bardziej zasadniczo: nie ma żadnego powodu w procesie analizy, aby zdaniu elementarnemu przysługiwały parametry wymagań, obecność tych parametrów nie wpływa język gramatyki.

> Z porównania list parametrów zdania elementarnego oraz zdania równorzędnego widać, że charakterystyka gramatyczna tego ostatniego jest uboższa. Zanik parametrów wymagania przy przejściu z poziomu zdania elementarnego na poziom zdania równorzędnego jest zrozumiały sam przez się i nie powoduje żadnych kłopotów.

W ramach prac nad ulepszoną wersją GFJP warto byłoby chyba zweryfikować zasadność wyposażania poszczególnych jednostek w parametry. Przy tej okazji niektóre parametry można byłoby ustrukturyzować. Dotyczy to zwłaszcza parametru zależności (por. p. 5.3.1).

Funkcja czy budowa

W punkcie 5.3.1 przedstawiam wątpliwości interpretacyjne dotyczące parametru zależności, który czasem zdaje sprawę z budowy wewnętrznej konstrukcji, a czasem z kontekstu, w którym się ona znalazła. Patrząc na rzecz inaczej, można powiedzieć, że czasem opisywana jest budowa, a czasem funkcja pełniona przez konstrukcję w wypowiedzeniu.

Świdziński w zasadzie opisuje funkcje poszczególnych konstrukcji. Dlatego wśród reguł opisujących poszczególne poziomy konstrukcji zdaniowych powtarza się podział na konstrukcje niezależne, pytajne, względne itd. Powtarzają się przy tym grupy nieomal identycznych reguł, różniących się tylko zestawem dopuszczalnych wartości zależności, która właśnie zdaje sprawę z funkcji pełnionej przez zdanie danego rodzaju.

Taki opis jest dość nietypowy dla formalizmu gramatyk metamorficznych, który daje dużo bardziej zwarty opis, jeżeli skupić się na budowie poszczególnych tworów językowych, rozumianej tylko jako zestaw jednostek składowych. Metoda zastosowana przez Świdzińskiego powoduje nieefektywność realizacji komputerowej, ponieważ znacząco zwiększa się liczba konstrukcji, które trzeba rozpatrywać. W programie *Świgra* efekt ten udało się częściowo zniwelować poprzez operowanie na zbiorach dopuszczalnych wartości zależności, jednak zbiory te ulegałyby mniejszemu rozczłonkowaniu gdyby dodatkowo połączyć wzmiankowane reguły zdaniowe.

Sądzę, że w ulepszonej wersji GFJP warto byłoby rozważyć zmianę zasady opisu na opartą jedynie na budowie (składzie) opisywanych tworów (co w punkcie 5.3.1 nazwałem interpretacją zależności typu „musi”). W wypadku ulepszonych wersji programu *Świgra* taka zmiana wydaje mi się koniecznością, jeżeli celem miała by być znacząco lepsza efektywność analizy.

6. Wyniki eksperymentów

6.1. Adekwatność gramatyki Świdzińskiego

Za podstawowy materiał do eksperymentalnej weryfikacji GFJP przyjąłem przykładowe wypowiedzenia zawarte w aneksie do książki Świdzińskiego. Przykłady te, w liczbie 660, ilustrują poszczególne reguły gramatyki. Zbiór ten zawiera zarówno przykłady, które wedle intencji Autora należą do języka opisywanego przez GFJP (nazywam je w uproszczeniu poprawnymi), jak i przykłady, które nie powinny być zaakceptowane (niepoprawne).

Zbiór przykładów został poddany analizie z użyciem dwóch wariantów algorytmu. Pierwszy wariant to opisany w punkcie 3.3 i 3.4 analizator o strategii wstępującej konstruujący upakowany las analiz. Wariant drugi to prosty analizator tablicowy skonstruowany według podręcznika Gazdara i Mellisha (1989), również pracujący w porządku wstępującym. Nazwijmy te warianty odpowiednio analizatorem L i analizatorem T.

Tabela 6.1 zestawia podstawowe wyniki analizy zbioru przykładów Świdzińskiego. Należy zaznaczyć, że analizator T dla 21 przykładów nie zmieścił się w limicie czasu pracy ustawionym na 8 minut na przykład. W pozostałych wypadkach wygenerował te same drzewa co analizator L (dane w górnej części tabeli pochodzą więc z wyników analizatora L). Kolumny tabeli zatytułowane akc. i nieakc. opisują odpowiednio przykłady zaakceptowane i odrzucone przez program *Świgra*. Przez wartości przeciętne rozumiem medianę. Wartości te w istocie cechuje duży rozrzut, jak będzie widać na następnych ilustracjach.

Tabela 6.1. Wyniki analizy zbioru przykładów Świdzińskiego

	poprawnych		niepoprawnych	
	akc.	nieakc.	akc.	nieakc.
liczba przykładów	515		145	
	469	46	34	111
	91%	9%	23%	77%
przeciętnie drzew	10		14	
	analizator L			
przeciętny czas (s)	0,26	0,36	0,20	0,15
przec. l. kroków	267905	273601	201369	183400
	analizator T			
przeciętny czas (s)	3,85	50,86	2,42	1,6
przec. l. kroków	332506	1435570	264918	248407

Jak widać, analizator dość dobrze radzi sobie z przykładami poprawnymi. Przyczyną odrzucenia 46 wypowiedzeń poprawnych są, w moim przekonaniu, błędy w gramatyce,

a nie w programie *Świgra*. Kilka z tych przykładów analizuję w punkcie 6.4. Poprawienie tych błędów wymagałoby intensywnych konsultacji z Autorem GFJP. Próba „zaatakowania" kilku z tych przykładów wykazała, że problemy są nieoczywiste i wymagają dużego nakładu pracy również ze strony Konsultanta. Dlatego w ramach tej pracy nie było możliwe ich poprawienie.

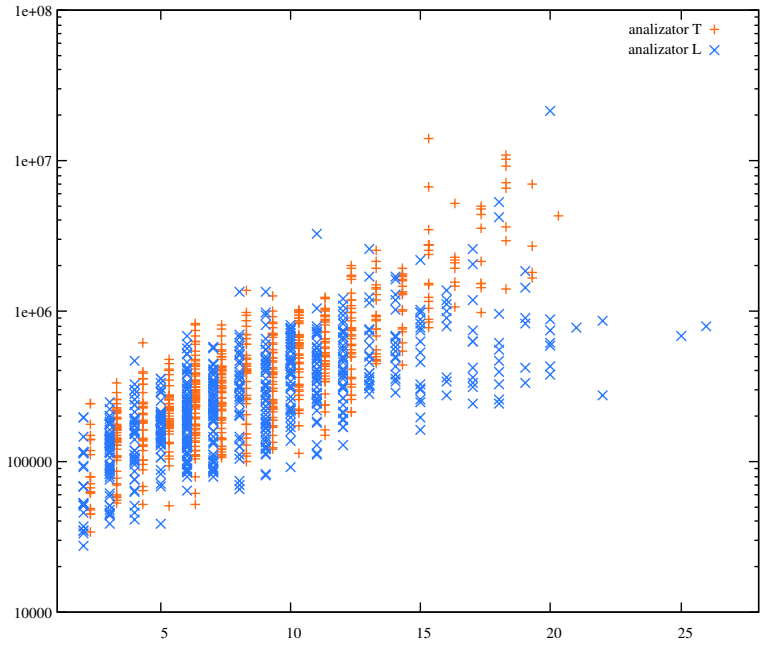
Dość wysoka jest liczba 23% zaakceptowanych przykładów niepoprawnych. Przykłady niepoprawne są w książce przypisane do pewnych reguł gramatyki. W większości przypadków ilustrowana reguła faktycznie nie może zostać użyta do analizy danego przykładu. Wypowiedzeniu jest natomiast nadawana zupełnie inna struktura. Przykłady takich wypowiedzeń przedstawiam w punkcie 6.5.

Adekwatność obserwacyjna w sensie Bańki (1990, 1985), czyli stosunek liczby poprawnych odpowiedzi do rozmiaru danych testowych, na tym zbiorze przykładów wynosi 88%.

Czasy analizy podane w tabeli i następujących dalej ilustracjach zostały zmierzone na współczesnym komputerze klasy PC (procesor Pentium 4 1,8GHz, system Debian GNU/Linux).

6.2. Efektywność analizy

Rysunek 6.1 przedstawia zależność liczby wywołań i nawrotów procedur prologowych (kroków wywodu, ang. *logical inferences*) potrzebnych do zanalizowania poszczególnych



Rys. 6.1. Zależność liczby kroków wywodu od długości wejścia

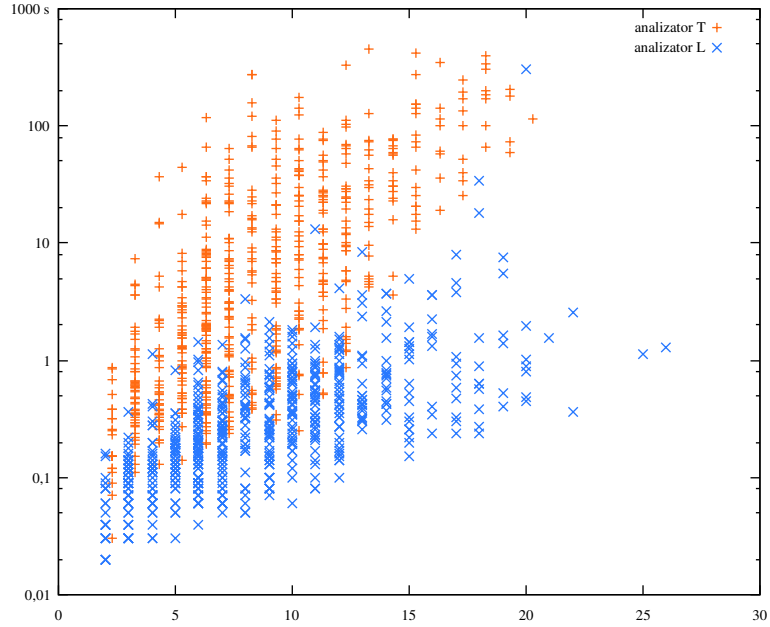
przykładów w funkcji ich długości — liczby segmentów na wejściu. (Dla zwiększenia czytelności krzyżyki dla analizatora T zostały przesunięte odrobinę w prawo). Jak widać, „chmury" krzyżyków są dosyć rozproszone. Oprócz długości wypowiedzenia, na liczbę kroków muszą mieć wpływ inne czynniki. Faktycznie wiele zależy od konstrukcji badane-

go wypowiedzenia, można wskazać wypowiedzenia tej samej długości bardzo różniące się liczbą analiz i potrzebnym czasem.

Dla badanych wypowiedzeń analizator T potrzebuje większej liczby kroków. Po części może to być spowodowane tym, że analizator T został zrealizowany w sposób dość uproszczony. Jednak akurat na złożoność liczoną wywołaniami procedur powinno to mieć niewielki wpływ. Na przykład wyszukanie pasującej reguły gramatyki jest jednym wywołaniem procedury w analizatorze T, podobnie jak w analizatorze L. Sądzę więc, że obserwowane różnice świadczą po prostu, że analizator tablicowy jest nieco bardziej skomplikowanym algorytmem.

Można natomiast uznać za prawdopodobne, że w gramatyce Świdzińskiego nie ma fragmentów przypominających gramatykę z punktu 3.5, dla której analizator tablicowy miałby szansę być istotnie sprawniejszy. W związku z tym realizacja bardziej sprawnego analizatora tablicowego z wykorzystaniem techniki stosowanej w analizatorze L prawdopodobnie nie przyniosłaby istotnego zmniejszenia czasów analizy.

Wyraźną różnicę między analizatorem T i L widać na rysunku 6.2 przedstawiającym czas analizy w sekundach w zależności od długości wypowiedzeń. Jak sądzę, zasadniczym



Rys. 6.2. Zależność czasu analizy od długości wejścia

czynnikiem powodującym tę różnicę jest bardziej wyrafinowana konstrukcja analizatora L. Mianowicie w analizatorze L każda reguła gramatyki jest tłumaczona na jedną klauzulę predykatu dla pewnego nieterminala. Gdy potrzebne jest zastosowanie reguły jest wywoływany ten właśnie predykat. W analizatorze T wszystkie reguły gramatyki są przechowywane w formie jednego predykatu rule, więc wybranie potrzebnej reguły wymaga wyszukania odpowiednich jego klauzul. Ponadto w analizatorze L reguła jest uruchamiana, jako procedura prologowa, w analizatorze T zaś jej kolejne elementy są interpretowane przez program.

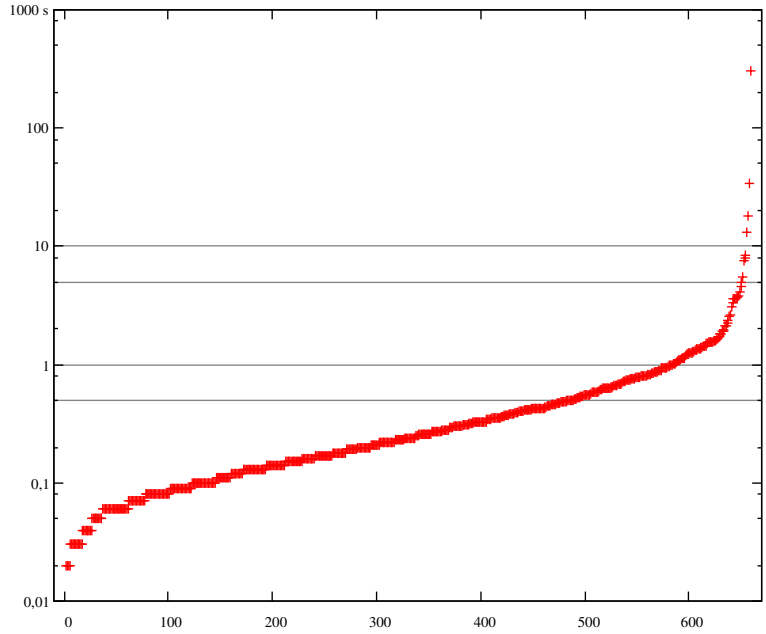
Jak widać na ilustracji, czasy analizy są silnie zróżnicowane. W sumie analizator L potrzebował 11 minut i 28 sekund na przetworzenie wszystkich wypowiedzeń. Analizator T pracował przez 6 godzin, 55 minut i 32 sekundy (z czego 4h 7m 32s przetwarzał wypo-

wiedzenia dla których zdążył udzielić odpowiedzi). Wypowiedzeniem, które analizator L przetwarzał najdłużej jest przykład ilustrujący regułę (j19):

(115) *Gdybyśmy mieli czas, to ja nie zostanę, on nie pójdzie, ona nie przyjdzie ani dziecko nie uśnie.*

Analiza trwała 302 sekundy, znaleziono 12 282 drzewa analizy, las wyników składa się z 18 050 łuków. Jak na taką liczbę łuków, liczba drzew jest stosunkowo niewielka. Wynika stąd, że w analizie tego przykładu rozważana jest bardzo duża liczba fraz, które nie przydają się do konstrukcji drzew dla całego wypowiedzenia.

Analizator T nie zdążył przetworzyć żadnego wypowiedzenia dłuższego niż 20 segmentów.

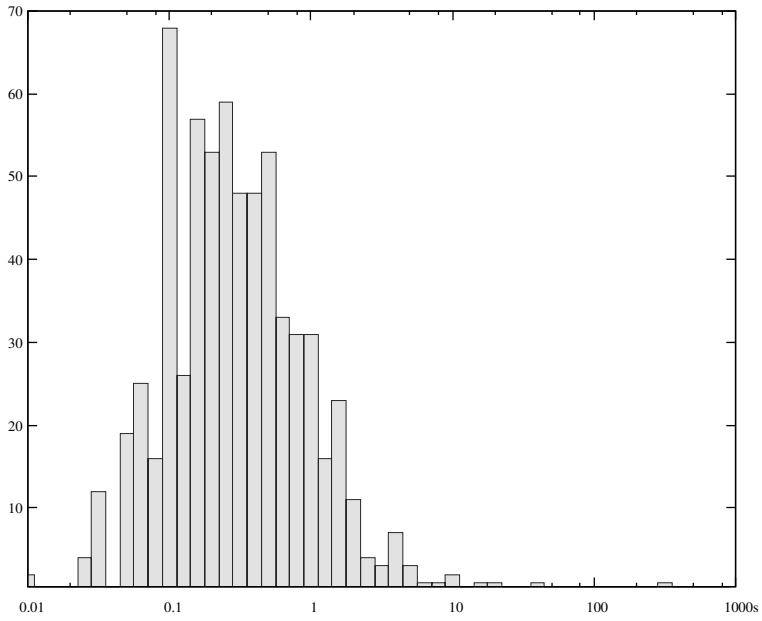


Rys. 6.3. Posortowane rosnąco czasy analizy

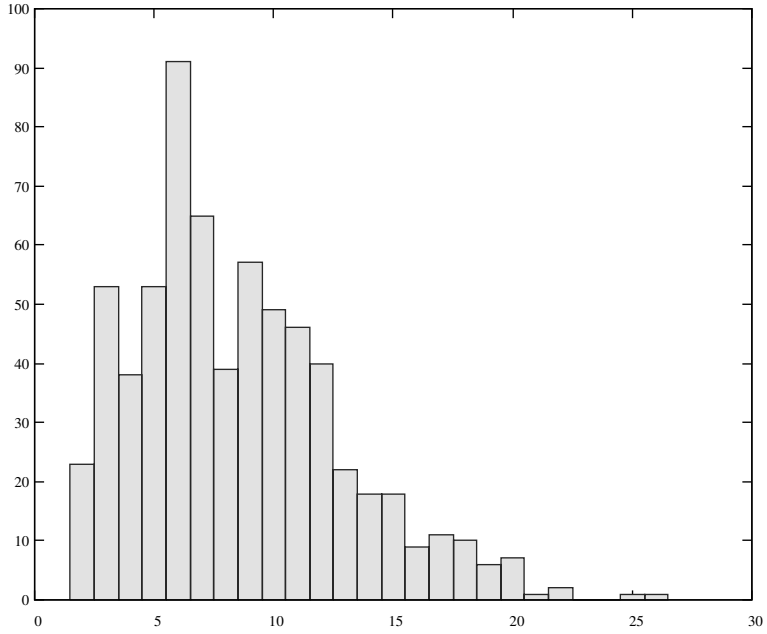
Przyjrzyjmy się teraz bliżej rozrzutowi czasów analizy. Rysunek 6.3 przedstawia czasy analizy 660 wypowiedzeń dla analizatora L posortowane rosnąco. Możemy z niego zorientować się, że 482 wypowiedzenia, czyli 73% zostało zanalizowanych w czasie nieprzekraczającym 0,5 sekundy. 89% wypowiedzeń wymagało nie więcej niż 1 sekundy. 98,8% wymagało nie więcej niż 5 s, wreszcie 99,4% nie więcej niż 10 s (4 wypowiedzenia wymagały ponad 10 sekund).

Rysunek 6.4 przedstawia zliczenia poszczególnych czasów analizy. Zakres czasów analizy od 0,01 s do 1000 s po zlogarytmowaniu został podzielony na 50 równych części, w których dokonano zliczeń.

Jako punkt odniesienia warto pokazać zliczenia długości wypowiedzeń w zbiorze testowym (są to liczby słów z uwzględnieniem znaków interpunkcyjnych).



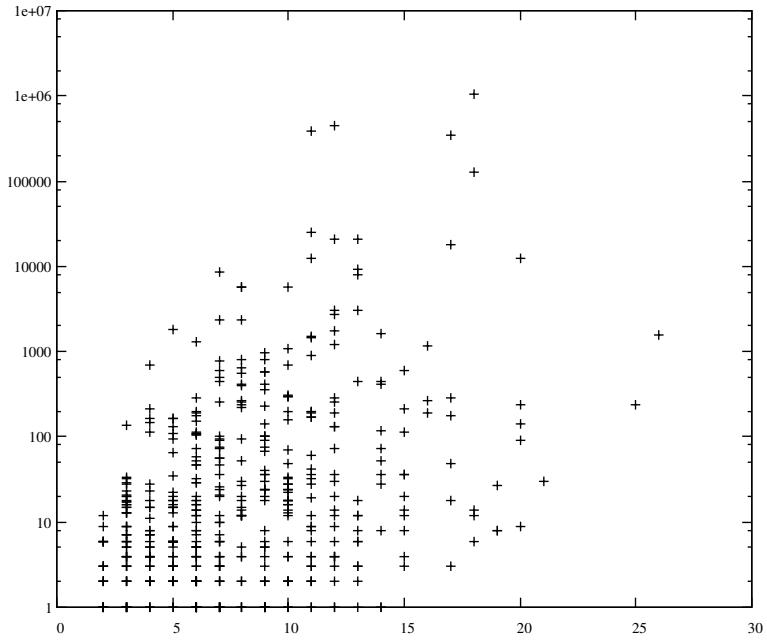
Rys. 6.4. Zliczenia czasów analizy (algorytm L)



Rys. 6.5. Zliczenia długości analizowanych wypowiedzeń

6.3. Liczba drzew analizy

Przyjrzyjmy się teraz liczbie drzew generowanych przez program. Rysunek 6.6 przedstawia tę liczbę w zależności od długości wypowiedzenia. Liczby drzew są silnie rozproszone,



Rys. 6.6. Zależność liczby drzew od długości wejścia (dla przykładów akceptowanych przez program)

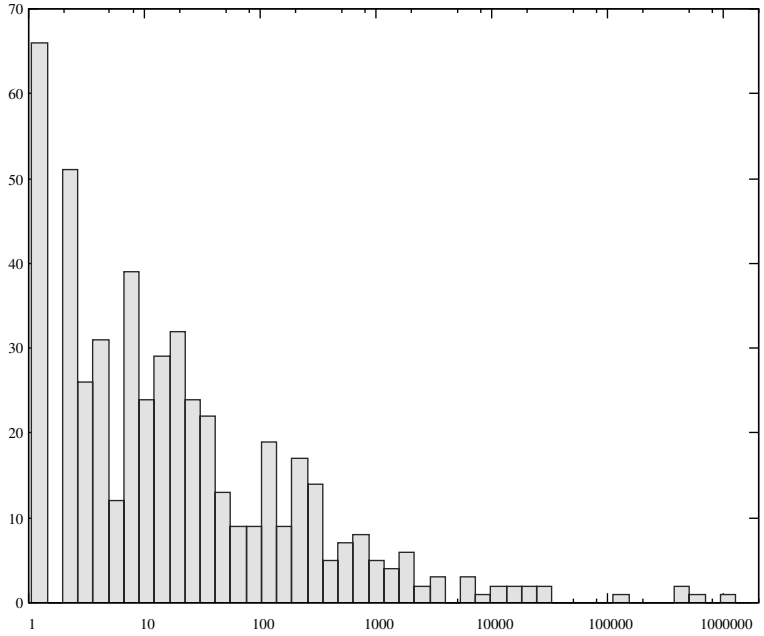
co jest zgodne z oczekiwaniami — można skonstruować bardzo długie wypowiedzenia, które będą miały jedno drzewo analizy. Możliwe też są wypowiedzenia krótkie, ale o wielu analizach.

Rysunek 6.7 przedstawia, dla ilu wypowiedzeń w wyniku analizy uzyskano daną liczbę drzew. Podobnie, jak przy zliczeniach czasów, zakres od 1 do 2 000 000 drzew podzielono po zlogarytmowaniu na 50 równych części. Ze względu na skalę logarytmiczną na rysunku brak zliczenia dla wypowiedzeń odrzuconych przez program. Było ich w sumie 157.

Nieco zaskakujące jest może maksimum liczby drzew. W sumie dla 660 wypowiedzeń program wygenerował 2 543 013 drzew. Dla wypowiedzenia ilustrującego regułę (j17)

(116) *Mieliśmy czas, aż ani ja nie zostałem, ani on nie poszedł, ani ona nie przyszła.*

program wygenerował 1 029 654 drzew (trwało to 18 sekund, las wyników ma 5861 łuków). Źródłem niejednoznaczności tego zdania jest akurat problem techniczny, który można uznać za błąd analizatora morfologicznego. Mianowicie w obecnej wersji programu *Morfeusz* ignorowane jest rozróżnienie między wielkimi i małymi literami, w związku z czym segment *ani* oprócz interpretacji spójnikowej został uznany za wykładnik leksemu Ania (który ma 4 możliwe interpretacje przypadkowo-liczbowe). Tego rodzaju niejednoznaczności między jednostką funkcyjną i „zwykłym" rzeczownikiem są jednak możliwe (np. *poza*, *bez*, *Ani*), więc przykład ten sygnalizuje pewien problem ogólny.



Rys. 6.7. Zliczenia liczb drzew

6.4. Przykłady wypowiedzeń poprawnych nie akceptowanych przez program

Przedstawię teraz najkrótsze nie akceptowane przez program zdania poprawne. Program odrzuca następujące zdania, które mają ilustrować regułę (lu19):

(117) *Czytano, co one wiedziały, książkę.*
(118) *Czytano, bo one wiedziały, książkę.*

Jeśli podstawić w miejsce *co* spójnik innego typu wymienionego w regule (lu19), a więc *chociaż*, *czy*, *jak*, *jeśli*, *podczas* lub *ponieważ*, otrzyma się zdanie akceptowane. We wszystkich tych zdaniach występuje fraza zdaniowa elementarna uzyskana za pomocą reguły (zd42). Reguła ta nie dopuszcza wartości co ani bo typu spójnika występującego w tej frazie. Rozważane zdania zostałyby zaakceptowane, gdyby poszerzyć listę dopuszczalnych wartości w regule (zd42).

Takie rozszerzenie nie wydaje się właściwe w wypadku spójnika co, jest on bowiem charakterystyczny dla fraz względnych. Właściwsze byłoby zatem użycie reguły (zd44). Fragment *, co one widziały* faktycznie daje się zanalizować z użyciem tej reguły, ale uzyskiwana fraza zdaniowa elementarna, ma w pozycji inkorporacyjności wartość nij/poj — co jest typowe dla fraz względnych. Niestety w definicji frazy finitywnej dopuszczana jest tylko fraza luźna o wartości inkorporacyjności ni. Być może właściwym sposobem dopuszczenia omawianej analizy jest zmiana warunków w definicji frazy finitywnej[1].

Fraza zdaniowa elementarna występuje w wielu miejscach w gramatyce i rozszerzenie jej definicji mogłoby stworzyć możliwość akceptowania jakichś zdań niepoprawnych.

[1] W rozmowie profesor Świdziński potwierdził, że właściwa jest ta ostatnia koncepcja.

Wprowadzenie takiej zmiany wymagałoby więc zakrojonej na większą skalę współpracy z Autorem GFJP.

Z następujących zdań pierwsze, stanowiące przykład dla reguły (we27), nie jest akceptowane, w przeciwieństwie do zdania drugiego:

(119) *Przeczytano to, póki nie przyszła.*
(120) *Przeczytano to, zanim nie przyszła.*

Wydaje się, że oba zdania powinny mieć przypisaną bardzo podobną strukturę. W analizie drugiego zdania znów wykorzystywana jest reguła (zd42), na której analiza pierwszego zdania musi utknąć. Być może należy więc rozszerzyć listę dopuszczalnych spójników w (zd42) również o wartość póki.[2]

6.5. Przykłady wypowiedzeń niepoprawnych akceptowanych przez program

Bardzo częstym powodem akceptowania przez program wypowiedzeń niepoprawnych są nieoczekiwane interwencje fraz luźnych. Program akceptuje na przykład następujące wypowiedzenie ilustrujące regułę (wy9), oznaczone przez Świdzińskiego jako niepoprawne:

(121) ?*Ona nie czytała książkę.*

Według reguły (wy9) zanegowana biernikowa fraza wymagana może być realizowana przez frazę nominalną w dopełniaczu (dopełniacz negacji). Tak więc *książkę* nie może być frazą wymaganą przez czasownik CZYTAĆ — który wymaga frazy w bierniku. Jednak fraza *książkę* może także zostać zanalizowana jako luźna na mocy reguły (lu6). Wypowiedzenie otrzymuje więc taką interpretację, jak następujące.

(122) *Ona nie czyta godzinę.*

Wydaje się, że akceptowanymi bez wątpliwości frazami luźnymi w bierniku są głównie określenia czasu. Trudno byłoby ograniczyć zbiór fraz, które zostaną dopuszczone w tej pozycji, środkami czysto składniowymi. Niestety takie określenia czasu mogą zawierać zaskakująco wiele leksemów oprócz oczywistych, jak GODZINA, MINUTA, CHWILA, itp. Na przykład dobrą miarą upływu czasu jest ZDROWAŚKA i KOLEJKA. Wśród graczy w golfa mógłby to chyba być również DOŁEK.

(123) *Ona nie żyje zdrowaśkę.*
(124) *Ona czeka kolejkę.*
(125) *Ona nie spuszcza wzroku z ekranu dołek, a Piotr już dwa dołki nie może się uspokoić.*

Większość fraz, które mogą stanowić realizację frazy wymaganej, może także stanowić frazę luźną. Tak więc tego rodzaju interpretacje są bardzo częste wśród drzew analizy.

[2] Profesor Świdziński wyjaśnił, że zdanie z *póki* nie powinno być akceptowane, przez pomyłkę zabrakło przy nim gwiazdki.

Zakończenie

W pracy została przedstawiona komputerowa realizacja gramatyki M. Świdzińskiego — program *Świgra*. W szczególności zaprezentowano efektywny algorytm analizy składniowej dla gramatyk metamorficznych. Zawarte w programie *Świgra* środowisko obliczeniowe dla gramatyk logicznych można wykorzystać z innymi gramatykami tego typu. Dla wygody eksperymentów z gramatykami niebagatelne znaczenie ma gotowy moduł analizy morfologicznej i przejrzysta postać graficzna uzyskiwanych drzew analizy.

Gramatyka Świdzińskiego okazała się poddawać realizacji komputerowej. Uzyskany program komputerowy wykazuje dość dużą zgodność z intencjami Autora GFJP. Należy więc uznać, że jest on dobrym punktem wyjścia do prac nad rozbudową opisu.

Uważam, że obecny stan programu *Świgra* stanowi kres możliwości lokalnego wprowadzania poprawek do GFJP. Teraz pora na fazę zasadniczej przebudowy — typową fazę rozwoju produktu informatycznego. Za najważniejsze kwestie uważam uporządkowanie znaczenia parametru zależności (por. p. 5.3.1) i zmniejszenie liczby nadmiarowych interpretacji. Przy tej okazji można podjąć próbę lepszego ustrukturyzowania i innych parametrów. Dla wygody opisu być może warto rozważyć alternatywne formalizmy gramatyczne.

Ulepszony opis powinien uwzględniać zjawiska opisane w GFJP pobieżnie lub nie opisane wcale, niektóre zanalizowane w nowszej literaturze przedmiotu. Chodzi m.in. o frazy liczebnikowe, równorzędnie złożone frazy rzeczownikowe i przymiotnikowe, specyficzne własności składniowe imiesłowów przymiotnikowych i gerundiów. Wydaje się celowe w szczególności wykorzystanie opisu równorzędnych fraz nominalnych przedstawionego przez Szpakowicza i Świdzińskiego (1990).

Warto poddać analizie automatycznej większe zbiory wypowiedzeń polskich pochodzących z autentycznych tekstów. Eksperymenty takie mogą ujawnić kolejne luki opisu i nowe problemy. W perspektywie wydaje się możliwe uzyskanie narzędzia atrakcyjnego z punktu widzenia potrzeb informatycznych, na przykład dokonującego ekstrakcji fraz nominalnych (ang. *NP chunking*) na potrzeby ekstrakcji informacji.

Obecna wersja programu *Świgra* wykazuje również niedoskonałości odziedziczone po modułach towarzyszących. Słownik analizatora morfologicznego *Morfeusz* ciągle wymaga ulepszeń. W tej materii znaczącą poprawę przyniesie wykorzystanie wyników realizowanego obecnie projektu KBN *Słownik gramatyczny języka polskiego* (kierowanego przez Z. Saloniego). Oprócz tego brakuje formalnego opisu związków frazeologicznych i dobrego słownika wymagań czasownikowych. Te elementy wymagają dalszych prac.

Program *Świgra* będzie dostępny bezpłatnie do celów badawczych. Mam nadzieję, że zwiększy to krąg osób zaangażowanych w formalny opis języka polskiego.

Dodatek A. Instrukcja użytkowania programu *Świgra*

Analizator składniowy Świgra jest zrealizowany w SWI-Prologu, do pracy wykorzystuje analizator morfologiczny *Morfeusz*. Program zawiera wyświetlarkę drzew analizy zrealizowaną za pomocą biblioteki graficznej XPCE. Program był testowany z wersją 5.2.13 SWI-Prologu i 6.2.13 XPCE pod systemem Linux.

Programu można używać na dwa sposoby: interakcyjnie lub wsadowo. Praca interakcyjna (p. A.1) odbywa się w sesji interpretera Prologu. W odpowiedzi na zapytanie wyświetlane są drzewa analizy. W trybie wsadowym (p. A.2) wypowiedzenia powinny być przygotowane w formie pliku z zapytaniami. Wyniki analizy są również zapisywane do plików w postaci termów reprezentujących las drzew analizy. Można uzyskać ich wizualizację w formacie PDF.

Program składa się z kilku modułów. Część z nich realizuje środowisko obliczeniowe dla gramatyk zbliżonych do DCG (z rozszerzeniami ale i ograniczeniami opisanymi w rozdziale 2.2), w szczególności sprzęg z analizą morfologiczną i strategię analizy składniowej. Część modułów jest ściśle związana z GFJP — dotyczy to np. predykatów realizujących warunki z gramatyki Świdzińskiego. W sensie technicznym jednym z takich modułów jest wynik tłumaczenia reguł gramatycznych na klauzule prologowe.

Reguły w formie opisanej w punkcie 2.1 są tłumaczone na Prolog w sposób opisany w p. A.3. W przypadku wprowadzenia zmian w regułach gramatycznych konieczne jest wygenerowanie prologowej wersji gramatyki. W pracy wsadowej dla efektywności ładuje się program z binarnego zrzutu pamięci interpretera Prologu. W przypadku zmiany w regułach lub w którymś z modułów analizatora konieczne jest ponowne wygenerowanie takiego zrzutu.

A.1. Praca interakcyjna

Aby przystąpić do pracy interakcyjnej należy uruchomić interpreter SWI-Prologu z rozszerzeniem XPCE. Pod Linuksem należy przejść do katalogu z plikami analizatora i wydać polecenie `xpce`. Pod MS Windows XPCE jest zawarte w interpreterze `plwin.exe`. Po uruchomieniu interpretera należy załadować plik `pforest.pl`, aby pracować z analizatorem działającym na upakowanych lasach:

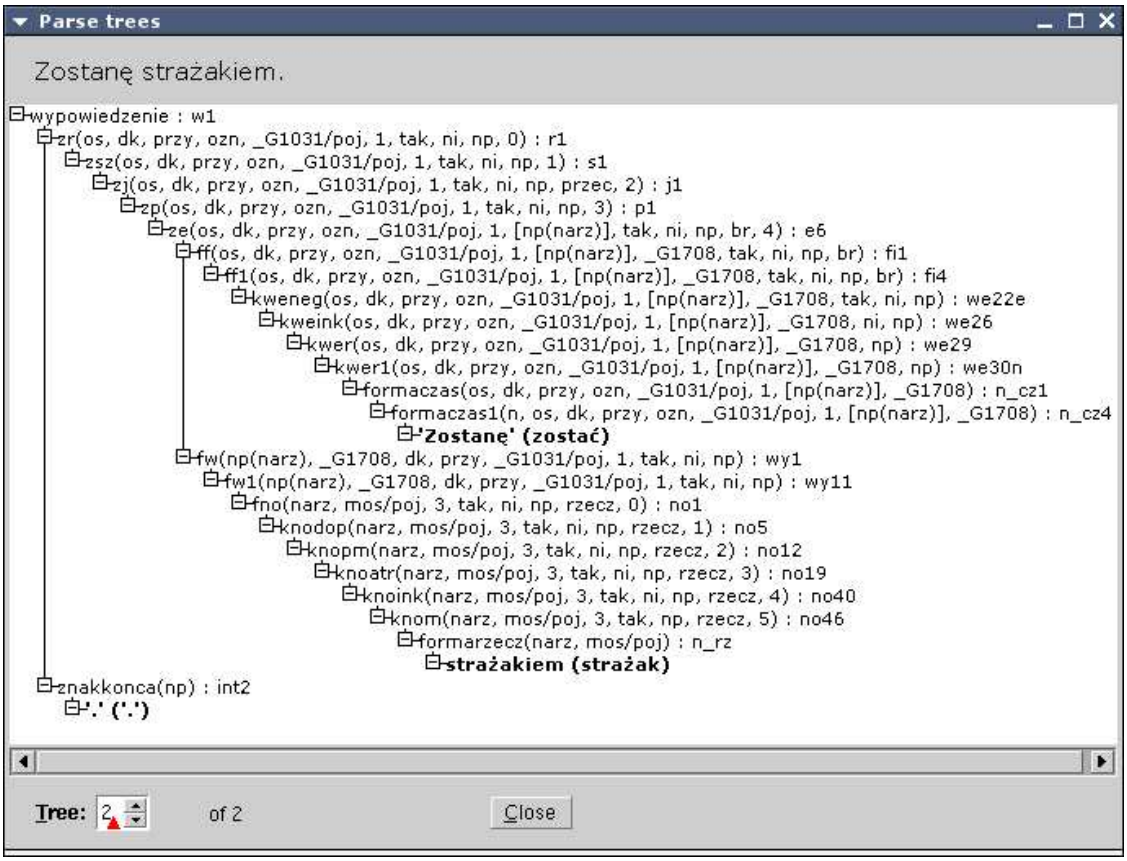
```
?- [pforest].
```

Aby używać analizatora tablicowego, należy zamiast tego załadować plik `pchart.pl`.

Do dyspozycji użytkownika jest kilka predykatów, z których podstawowym jest `analiza/1`. Jego argumentem powinno być wypowiedzenie w postaci pojedynczego atomu. Na przykład:

```
?- analiza('Zostanę strażakiem.').
```

W wyniku tego zapytania zostanie otwarte okienko przedstawione na ilustracji A.1, w którym można przeglądać drzewa odpowiadające podanemu wypowiedzeniu (przyciski ze strzałkami w dole okienka). Brak drzew oznacza porażkę analizy (potwierdzoną komentarzem w oknie interpretera; natomiast samo zapytanie zawsze odnosi sukces).



Rys. A.1. Drzewa analizy wyświetlone w oknie programu *Świgra*

Wygodnym predykatem do badania gramatyczności fragmentów wypowiedzeń jest analiza/2. Pierwszym argumentem tego predykatu jest fragment tekstu do analizy, drugim natomiast nazwa jednostki nieterminalnej, jako która tekst ten miałby być zanalizowany. Na przykład za pomocą następującego zapytania można sprawdzić, jaką charakterystykę mogą mieć frazy finitywne **ff** realizowane przez słowo *zostałem*:

```
?- analiza('zostałem', ff).
```

Za pomocą predykatu analiza1/2 można zadawać pytania jeszcze bardziej szczegółowe. Jego drugim argumentem musi być jednostka nieterminalna z argumentami, które mogą być częściowo ukonkretnione. Na przykład za pomocą następującego zapytania można sprawdzić, czy słowo *zostałem* może zostać zinterpretowane jako fraza finitywna o niedokonanej wartości parametru aspektu:

```
?- analiza1('zostałem', ff(_,nd,_,_,_,_,_,_,_,_,_,_,_,_)).
```

A.2. Praca wsadowa

W pracy wsadowej jest wykorzystywany zrzut pamięci interpretera Prologu przechowywany w pliku `gfjp-bin`. Sposób tworzenia tego pliku opisano w następnym punkcie. Wariant analizatora, który ma być użyty, należy wybrać przy tworzeniu zrzutu pamięci.

Najprostsza forma wsadowego uruchomienia analizatora polega na użyciu skryptu `przyk2pdf`, który uruchamia program dla wypowiedzenia podanego jako jedyny argument skryptu i zapisuje odpowiadające mu drzewa analizy w formacie PDF w pliku `zsypprzyk.pdf`. Oto przykładowe wywołanie programu:

```
./przyk2pdf 'Ja zostałem.'
```

Jeżeli do przetworzenia jest większa grupa przykładów należy postępować następująco. Ze względu na czasochłonność niektórych obliczeń wygodnie jest móc ograniczyć czas analizy poszczególnych przykładów. Dlatego zaleca się podawanie poszczególnych wypowiedzeń do analizy w osobnych plikach. Najwygodniej jest przygotować nowy katalog siostrzany do katalogu `parser/` (a więc taki, z którego katalog z programem widać jako `../parser/`). W tym katalogu umieścić zestaw plików o nazwach `przyk`*numer*`.doa` zawierających po jednym wywołaniu któregoś z predykatów omówionych w poprzednim punkcie. Na przykład:

```
:-analiza('Ja zostałem.').
:-halt.
```

Następnie należy przejść do katalogu z przykładami i uruchomić skrypt `runme` poleceniem:

```
../parser/runme
```

Spowoduje to przetworzenie każdego z plików, którego nazwa jest zgodna z podanym wyżej schematem. W skrypcie `runme` zawarte jest ograniczenie na czas liczenia jednego przykładu (zmienna `timelimit`, wartość wyrażona w sekundach).

Skrypt można bezpiecznie uruchomić ponownie — przetwarza on tylko te przykłady, dla których nie ma w katalogu pliku z wynikami (plik z rozszerzeniem `.trees`). Aby wymusić ponowne liczenie przykładów należy usunąć odpowiednie pliki z wynikami.

Pojedynczy plik z przykładem można też przetworzyć za pomocą skryptu `runone`. Jego argumentem jest nazwa pliku do przetworzenia.

A.3. Przygotowanie gramatyki

Gramatyka w formie zbliżonej do DCG (por. p. 2.1), aby mogła zostać załadowana do interpretera prologu, jest tłumaczona na klauzule programu (por. p. 2.3).

Jedynym elementem tego przekształcenia specyficznym dla GFJP jest obsługa predykatów wymagania i wymagane, odpowiedzialnych za realizację zdania elementarnego (zob. 5.2).

Tłumaczenia GFJP na klauzule programu dokonuje się następującym poleceniem (skryptem):

```
./gengfjppl
```

Powoduje ono przetłumaczenie reguł gramatyki Świdzińskiego z pliku `gfjp.dcg` i dodanych reguł preterminalnych z pliku `gfjp-elem.dcg` i zapisanie wyniku jako `gfjp.pl`. Ten plik wystarcza do pracy interakcyjnej z programem.

Aby utworzyć zrzutu pamięci interpretera Prologu konieczny do pracy wsadowej należy wykonać następujące polecenie

```
./genparser
```

Skrypt ten wywołuje w szczególności poprzedni. Domyślnie tworzony zrzut zawiera analizator w wariancie pracującym na upakowanym lesie (analizator L, por. 6.1). Użycie opcji `-c` spowoduje wygenerowanie zrzutu dla analizatora tablicowego (analizatora T).

Inne zbiory reguł (np. w pliku `reguly.dcg`) można przetłumaczyć na Prolog za pomocą polecenia

```
./dcg2pl reguly
```

Dodatek B. Postać drzew analizy

Wizualizowane drzewo jest drzewem analizy, więc każdy jego węzeł odpowiada zastosowaniu jednej reguły gramatycznej. W każdym węźle wewnętrznym zapisywana jest jednostka nieterminalna stojąca po lewej stronie pewnej reguły wraz z przypisanymi jej wartościami argumentów. Dziećmi tego wierzchołka są elementy prawej strony reguły. Kolejności od lewej do prawej w regule odpowiada kolejność z góry do dołu w drzewie.

W każdym liściu drzewa znajduje się element terminalny ujęty w ramkę. Podawany jest segment (kursywą) i identyfikator leksemu (kapitalikami). Interpretację morfosyntaktyczną można wywnioskować z opisu węzła nieterminalnego dominującego nad danym terminalnym. Dopisane kontekstowo przecinki (por. 5.6) mają zgodnie z oczekiwaniami pusty segment.

Korzeń drzewa znajduje się na górze diagramu. Kreski ciągłe łączą każdy z węzłów wewnętrznych z jego dziećmi w drzewie. Do jednostek stanowiących prawą stronę jednej reguły (współskładników) dochodzą kreski poziome wychodzące ze wspólnej kreski pionowej.

W drzewach skróconych pokazywane są jedynie te węzły, które mają więcej niż jedno dziecko i ich dzieci. Dodatkowo pokazywane są wszystkie węzły preterminalne.[1] W skróconym drzewie stosowane są kreski przerywane tam, gdzie łączone są węzły, które nie są rodzicem i dzieckiem. (Czyli kreska przerywana zastępuje pewien nierozgałęziający się fragment gałęzi drzewa).

Po prawej stronie diagramu podane zostały numery reguł GFJP, które interweniowały przy tworzeniu poszczególnych wierzchołków drzewa. Dodane przeze mnie reguły preterminalne mają nazwy zaczynające się od (n_). Niektóre reguły oryginalnej gramatyki zostały połączone. Mamy wtedy do czynienia z oznaczeniami takimi jak: (we30n) ((we30) i następne), (we2/4/6) — połączenie reguł (we2), (we4) i (we6). Nazwy reguł kończące się literą e wskazują reguły dodane w ramach eliminacji reguł skracających (por. 5.5).

Nieustalone wartości parametrów (zmienne wolne) są zapisywane w formie podkreślenia i liczby (np. *_0*). Wszystkie wystąpienia tej samej zmiennej są oznaczone tą samą liczbą.

W wizerunkach drzew można dostrzec następujące różnice w stosunku do drzew, które uzyskałoby się za pomocą nie zmienionej gramatyki Świdzińskiego:

— Zamiast trzech parametrów wymagań *Wa*, *Wb*, *Wc* odpowiednie jednostki nieterminalne mają jeden argument będący prologową listą wymagań. Należy zaznaczyć, że w drzewie analizy na takiej liście pojawiają się wyłącznie wymagania, które zostały zrealizowane w danym wypowiedzeniu.
— Parametry wymagań mają inną postać. Opisałem ją w punkcie 5.2.1.
— Liczba fraz wymaganych **fw** zdominowanych przez zdanie elementarne **ze** może być różna — od zera do trzech. Nie ma w drzewie pustych fraz wymaganych. W wersji

[1] Wyjątkowo w drzewie 1.4 w p. 1.2 uwidoczniono więcej węzłów.

oryginalnej każda realizacja zdania elementarnego ma trzy frazy wymagane — z czego część pustych.
— Nie są tworzone drzewa, które różnią się od pewnych innych tylko obecnością pustej frazy luźnej.
— Jednostki należące do jałowych cykli (zob. 5.1) mają dodatkowy ostatni argument będący liczbą naturalną.
— Występuje więcej oznaczeń rodzaju. Symbole reprezentujące zbiory rodzajów elementarnych zestawiłem w tabeli na stronie 93.
— Parametr rodzaju-liczby w GFJP jest termem z funktorem . (kropka), na przykład mos.poj. Ponieważ kropka jest funktorem służącym w Prologu do reprezentowania list, interpretery Prologu uparcie wypisują taki term w mało czytelnej równoważnej formie [mos | poj]. Aby tego uniknąć w programie *Świgra* parametr rodzaju-liczby jest termem z funktorem / (ukośnik), np. mos/poj.
— Znak prim ' w wartościach parametru zależności (np. np't) zastępuje litera x (np. npxt).

B.1. Lista jednostek nieterminalnych

agl(*Rl*, *O*, *I*) — aglutynant
agl1(*Rl*, *O*) — aglutynant właściwy
condaglt(*L*, *O*) — warunkowy morfem aglutynacyjny [dodana w programie *Świgra*]
ff(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Neg*, *I*, *Z*, *Ow*) — fraza finitywna
ff1(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Neg*, *I*, *Z*, *Ow*) — fraza finitywna właściwa
fl(*A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z*) — fraza luźna
fl1(*A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z*) — fraza luźna właściwa
fno(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*) — fraza nominalna
formaczas(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*) — forma czasownikowa [nie definiowana w GFJP]
formaczas1(*S*, *Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*) — forma czasownikowa właściwa [dodana w programie *Świgra*]
formaprzym(*P*, *Rl*, *St*) — forma przymiotnikowa [nie definiowana w GFJP]
formaprzysl(*St*) — forma przysłówkowa [nie definiowana w GFJP]
formarzecz(*P*, *Rl*) — forma rzeczownikowa [nie definiowana w GFJP]
fpm(*Pm*, *P*, *Neg*, *I*, *Z*, *Kl*) — fraza przyimkowa
fps(*St*, *Neg*, *I*, *Z*, *Kl*) — fraza przysłówkowa
fpt(*P*, *Rl*, *St*, *Neg*, *I*, *Z*, *Kl*) — fraza przymiotnikowa
fw(*Tfw*, *K*, *A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z*) — fraza wymagana
fw1(*Tfw*, *K*, *A*, *C*, *Rl*, *O*, *Neg*, *I*, *Z*) — fraza wymagana właściwa
fwe(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Neg*, *I*, *Z*) — fraza werbalna
fzd(*Tfz*, *K*, *A*, *C*, *T*, *Neg*, *I*) — fraza zdaniowa
fzde(*Tfz*, *A*, *C*, *T*, *Neg*, *I*) — fraza zdaniowa elementarna
fzdj(*Tfz*, *K*, *A*, *C*, *T*, *Neg*, *I*, *Oz*) — fraza zdaniowa jednorodna
fzdkor(*Tfz*, *K*, *A*, *C*, *T*, *Neg*, *I*) — fraza zdaniowa z korelatem
fzdsz(*Tfz*, *K*, *A*, *C*, *T*, *Neg*, *I*) — fraza zdaniowa szeregowa
knom(*P*, *Rl*, *O*, *Neg*, *Z*, *Kl*) — konstrukcja nominalna
knoatr(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja nominalna z atrybutem

knodop(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja nominalna z dopełniaczem
knoink(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja nominalna z inkorporacją
knopm(*P*, *Rl*, *O*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja nominalna z frazą przyimkową
kor(*K*, *I*) — korelat
kor1(*P*) — korelat właściwy
kprzym(*P*, *Rl*, *St*, *Neg*, *Z*, *Kl*) — konstrukcja przymiotnikowa
kprzysl(*St*, *Neg*, *Z*, *Kl*) — konstrukcja przysłówkowa
kpsink(*St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przysłówkowa z inkorporacją
kpspm(*St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przysłówkowa z frazą przyimkową
kpsps(*St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przysłówkowa z frazą przysłówkową
kptink(*P*, *Rl*, *St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przymiotnikowa z inkorporacją
kptno(*P*, *Rl*, *St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przymiotnikowa z frazą nominalną
kptpm(*P*, *Rl*, *St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przymiotnikowa z frazą przyimkową
kptps(*P*, *Rl*, *St*, *Neg*, *I*, *Z*, *Kl*) — konstrukcja przymiotnikowa z frazą przysłówkową
kweink(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *I*, *Z*) — konstrukcja werbalna z inkorporacją
kweneg(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Neg*, *I*, *Z*) — konstrukcja werbalna z negacją
kwer(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Z*) — konstrukcja werbalna
kwer1(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *K*, *Z*) — konstrukcja werbalna właściwa
morfagl(*F*, *Rl*, *O*) — morfem aglutynacyjny
partykula(*F*) — partykuła
przec — przecinek
przecsp — przecinek przedspójnikowy
przyimek(*Pm*, *P*) — przyimek
przyzlo(*S*, *Rl*, *Wym*, *K*) — składowa czasu przyszłego [dodana w programie *Świgra*]
pyt(*F*, *I*) — pytajnik partykułowy
spoj(*Tsp*, *Oz*, *I*) — spójnik
spoj1(*Tsp*, *Oz*) — spójnik właściwy
spojnik(*Oz*) — spójnik
wypowiedzenie — wypowiedzenie
zaimneg(*Kgz*, *P*, *Rl*, *O*, *Kl*) — zaimek negatywny [nie definiowana w GFJP]
zaimno(*Kgz*, *P*, *Rl*, *O*, *Kl*) — zaimek nieokreślony
zaimos(*P*, *Rl*, *O*) — zaimek osobowy [nie definiowana w GFJP]
zaimprzym(*F*, *P*, *Rl*) — zaimek przymiotny
zaimprzys(*F*) — zaimek przysłowny
zaimpyt(*Kgz*, *P*, *Rl*, *O*, *Kl*) — zaimek pytajny
zaimrzecz(*H*, *P*, *Rl*) — zaimek rzeczowny
zaimwzg(*Kgz*, *P*, *Rl*, *O*, *Kl*) — zaimek względny
ze(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Wym*, *Neg*, *I*, *Z*, *Ow*) — zdanie elementarne
zj(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Neg*, *I*, *Z*, *Oz*) — zdanie jednorodne
znakkonca(*Z*) — znak końca
zp(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Neg*, *I*, *Z*) — zdanie proste
zr(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Neg*, *I*, *Z*) — zdanie równorzędne
zsz(*Wf*, *A*, *C*, *T*, *Rl*, *O*, *Neg*, *I*, *Z*) — zdanie szeregowe

B.2. Lista parametrów

A	aspekt
C	czas
F	forma (ściślej: segment)
H	identyfikator leksemu
I	inkorporacja
K	korelatywność
Kgz	klasa gramatyczna zaimka
Kl	klasa
L	liczba
Neg	negacja
O	osoba
Ow	ograniczenie wewnętrzne
Oz	oznaczenie spójnika
P	przypadek
Pm	przyimek
R	rodzaj
Rl	rodzaj-liczba
S	wymaganie *się*
St	stopień
T	tryb
Tfw	typ frazy wymaganej
Tfz	typ frazy zdaniowej
Tsp	typ spójnika
Wf	wyróżnik fleksyjny
Wym	wymagania
Z	zależność

B.3. Lista wartości parametrów

Na poniższej liście pomijam listy wartości parametrów określone słownikowo, a więc wartości mogące przysługiwać parametrowi *F*, *H*, *Oz* i *Pm*.

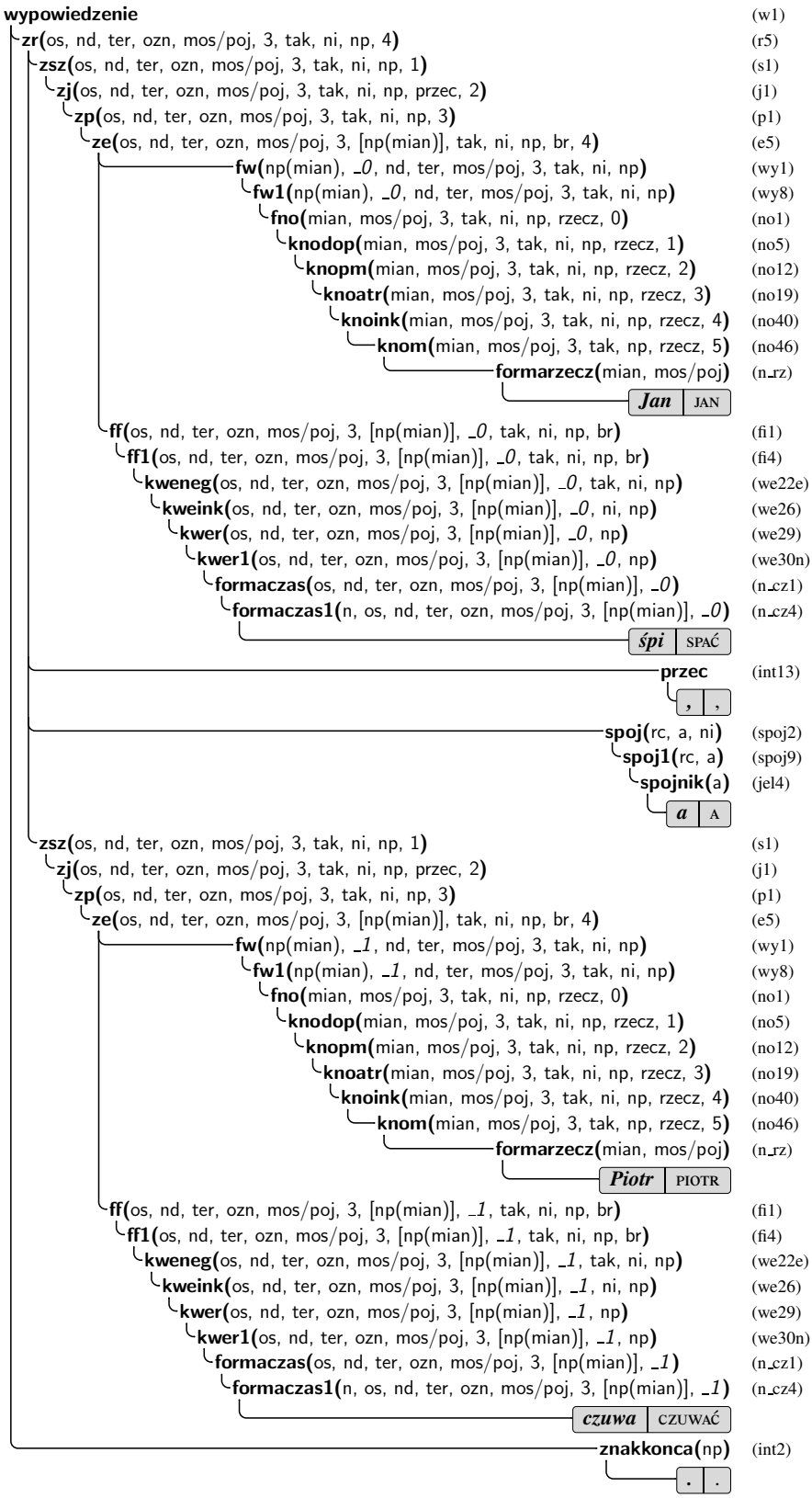
Wartość	Parametr	Opis
1	*O*	pierwsza
2	*O*	druga
3	*O*	trzecia
adjp(*P*)	*Tfw*	fraza przymiotnikowa w przypadku *P*
advp	*Tfw*	fraza przysłówkowa
ani	*Neg*	zaprzeczona odspójnikowa
aż1	*Tfz*,*Z*	spójnikowa
aż1xx	*Z*	typu aż z ograniczeniem czasu, aspektu i negacji
aż2	*Tfz*,*Z*	spójnikowa
bier	*P*	biernik
bo	*Tfz*,*Z*	spójnikowa
bok	*Wf*	bezokolicznikowy

bos	*Wf*	bezosobowy
bowiem	*I*	inkorporacyjna
bowiem	*Tfz*,*Z*	spójnikowa
box	*Z*	z ograniczeniem czasu
boxx	*Z*	z ograniczeniem czasu, aspektu i negacji
bp	*K*	bezprzecinkowa
br	*Ow*	brak ograniczenia
byxx	*Z*	z ograniczeniem czasu, aspektu i negacji
cel	*P*	celownik
chociaż	*Tfz*,*Z*	spójnikowa
choćby	*Ow*,*Tfz*,*Z*	spójnikowa
co	*Kl*	zaimek typu co
co	*Tfz*,*Z*	względna
czy	*Tfz*,*Z*	spójnikowa
czyżby	*Z*	kontekstowa
dk	*A*	dokonany
dop	*P*	dopełniacz
dopóki	*Ow*,*Tfz*,*Z*	spójnikowa
gdy	*Ow*,*Tfz*,*Z*	spójnikowa
gdyby	*Ow*,*Tfz*,*Z*	spójnikowa
gdyxx	*Z*	z ograniczeniem czasu, aspektu i negacji
infp(*A*)	*Tfw*	fraza czasownikowa w bezokoliczniku o aspekcie *A*
jak	*Tfz*,*Z*	spójnikowa
jakby	*Tfz*,*Z*	spójnikowa
jaki	*Tfz*,*Z*	względna
jakoby	*Tfz*,*Z*	spójnikowa
jeśli	*Tfz*,*Z*	spójnikowa
kto	*Kl*	zaimek typu кто
kto	*Tfz*,*Z*	względna
który	*Tfz*,*Z*	względna
m	*R*	męski (osobowy, żywotny lub nieżywotny)
mian	*P*	mianownik
mie1	*Tfz*	mieszana 1
mie2	*Tfz*	mieszana 2
mie3	*Tfz*	mieszana 3
miej	*P*	miejscownik
mn	*R*	męski (dowolny) lub nijaki
mno	*L*	mnoga
mnos	*R*	męski nieosobowy (żywotny lub nieżywotny)
mnż	*R*	męski nieżywotny
mos	*R*	męski osobowy
mzw	*R*	męski zwierzęcy
mżyw	*R*	męski żywotny (osobowy lub zwierzęcy)
n	*S*	*się* niedopuszczalne
naj	*St*	najwyższy
narz	*P*	narzędnik
natomiast	*I*	inkorporacyjna
nd	*A*	niedokonany
ni	*I*	nieinkorporacyjna

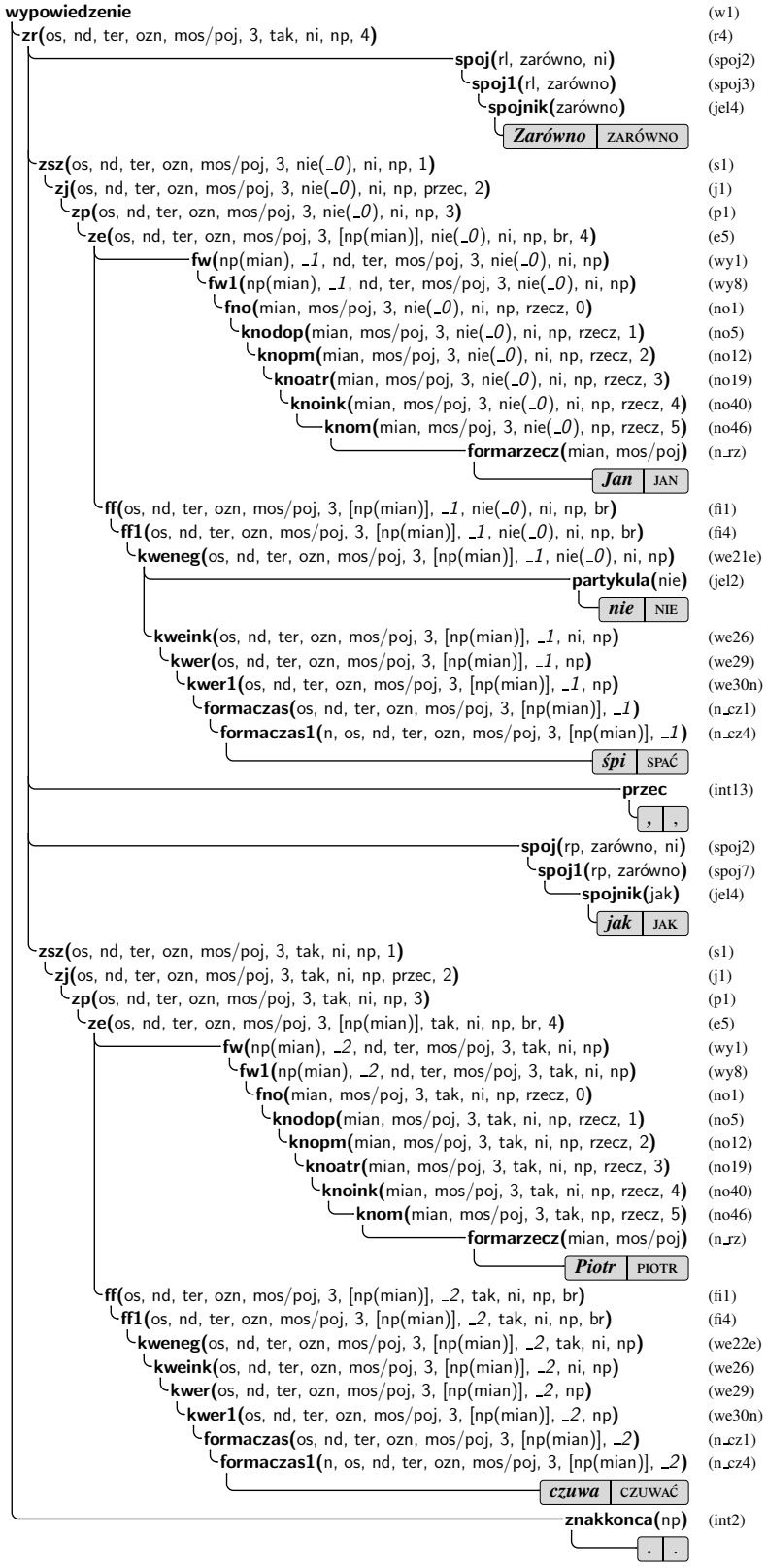
nie	*Neg*	zaprzeczona
nij	*R*	nijaki
nk	*K*	niekorelatywna
nmo	*R*	nie męski osobowy (dowolny z wyjątkiem mos)
np	*R*	nijaki lub przymnogi
np	*Z*	niepytajna
np(*P*)	*Tfw*	fraza nominalna w przypadku *P*
npt	*Z*	niepytajna z ograniczeniem trybu
npx	*Z*	niepytajna z ograniczeniem czasu
npxt	*Z*	niepytajna z ograniczeniem czasu i trybu
npxx	*Z*	niepytajna z ograniczeniem czasu, aspektu i negacji
npxxt	*Z*	niepytajna z ograniczeniem czasu, trybu, aspektu i negacji
os	*Kl*	zaimek osobowy
os	*Wf*	osobowy
ozn	*T*	oznajmujący
p	*Z*	pytajna
pc	*Tsp*	podrzędny centralny
pi	*Tsp*	podrzędny inkorporacyjny
pl	*Tsp*	podrzędny lewy
po	*Tsp*	podrzędny początkowy
podczas	*Tfz*,*Z*	spójnikowa
poj	*L*	pojedyncza
ponieważ	*Tfz*,*Z*	spójnikowa
pp	*Tsp*	podrzędny prawy
prepnp(*Pm*, *P*)	*Tfw*	fraza przyimkowa
prze	*C*	przeszły
przec	*Oz*	przecinkowy
przy	*C*	przyszły
przym	*Kgz*	przymiotna
przym	*Kl*	przymiotnik
przysl	*Kgz*	przysłowna
przysl	*Kl*	przysłówek
psu	*Wf*	imiesłowowy uprzedni
psw	*Wf*	imiesłowowy współczesny
pt	*R*	przymnogi
px	*Z*	pytajna z ograniczeniem czasu
pxx	*Z*	pytajna z ograniczeniem czasu, aspektu i negacji
pz	*Tfz*,*Z*	pytajnozależna
rc	*Tsp*	równorzędny centralny
ri	*Tsp*	równorzędny inkorporacyjny
rl	*Tsp*	równorzędny lewy
row	*St*	równy
roz	*T*	rozkazujący
rp	*Tsp*	równorzędny prawy
rzecz	*Kgz*	rzeczowna
rzecz	*Kl*	rzeczownik
s	*S*	*się* wymagane
sentp(*Tfz*)	*Tfw*	fraza zdaniowa typu *Tfz*
sz	*Tsp*	szeregowy

szk	*Tsp*	szeregowy końcowy
tak	*Neg*	niezaprzeczona
ter	*C*	teraźniejszy
tk	*Kl*	wartość specjalna
war	*T*	warunkowy
więc	*I*	inkorporacyjna
więc	*Ow*	
wol	*P*	wołacz
wyz	*St*	wyższy
wz	*Kl*	zaimek względny
zaim	*Kl*	zaimek
zanim	*Tfz*,*Z*	spójnikowa
zanimxx	*Z*	zanim z ograniczeniem czasu, aspektu i negacji
zaś	*I*	inkorporacyjna
że	*Tfz*,*Z*	spójnikowa
żeby	*Tfz*,*Z*	spójnikowa
żeń	*R*	żeński

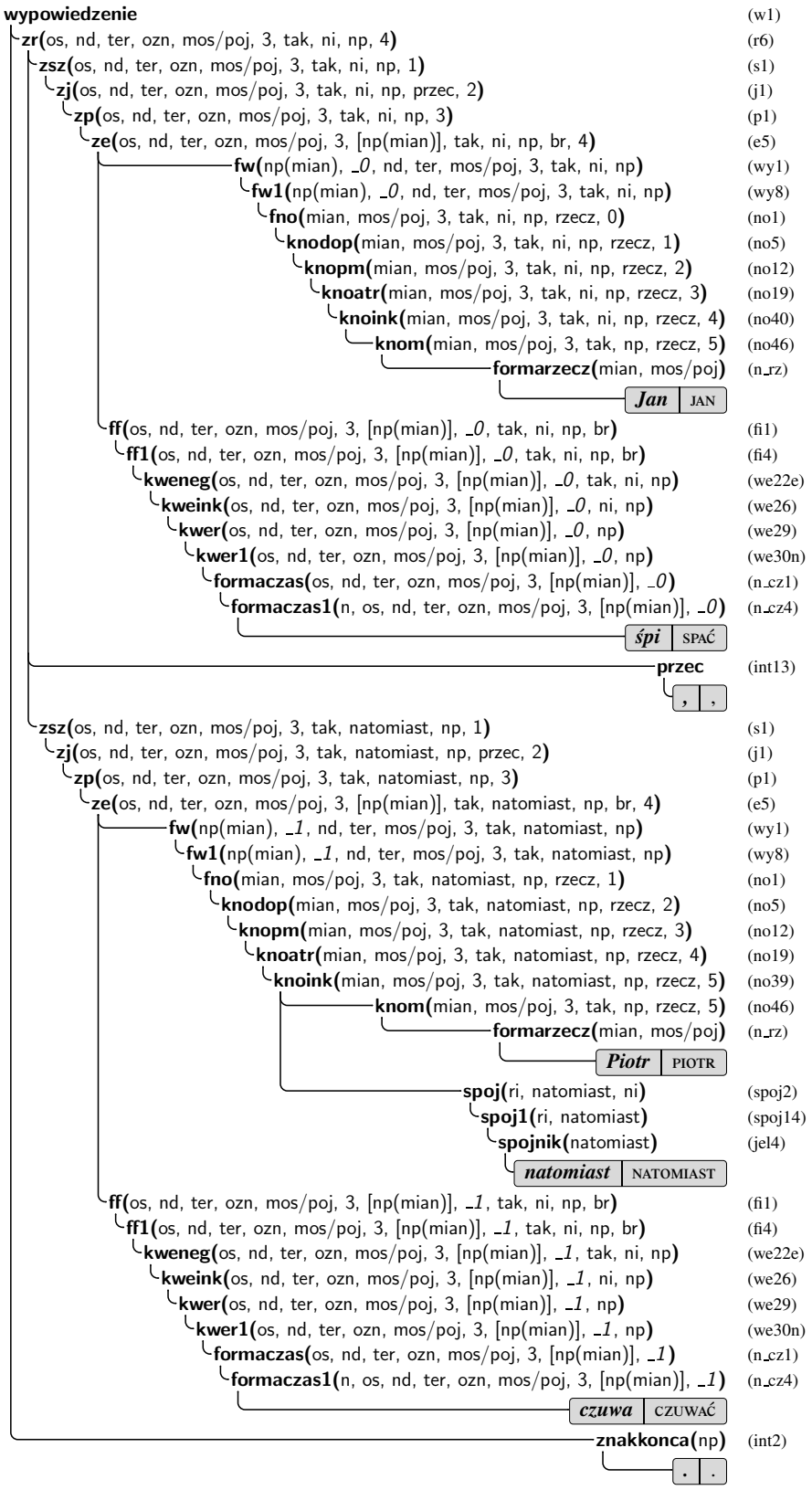
Dodatek C. Przykładowe wyniki analizy



Rys. C.1. Drzewo analizy zdania *Jan śpi, a Piotr czuwa.* Przykład zdań zespolonych spójnikiem równorzędnym centralnym A.

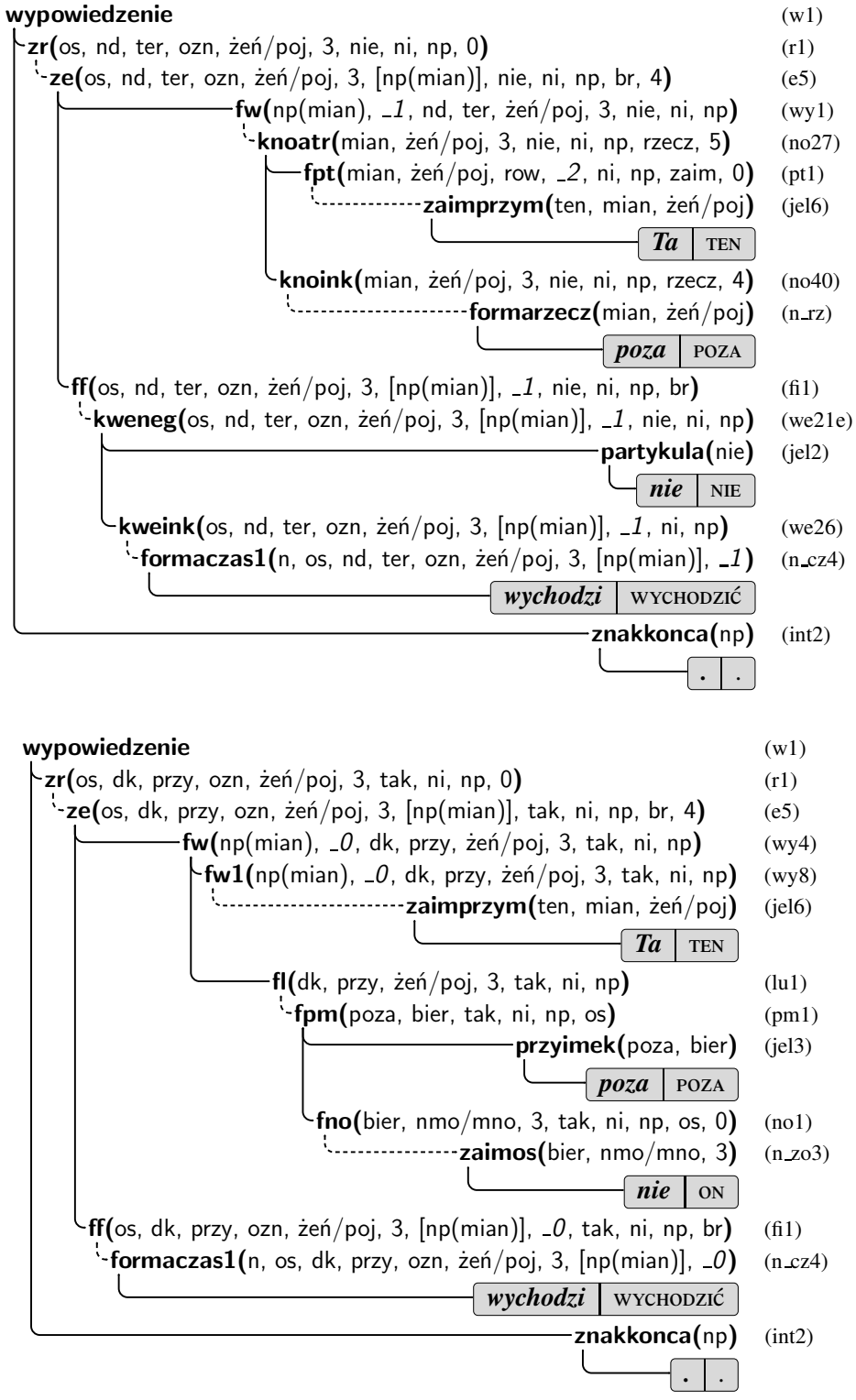


Rys. C.2. Drzewo analizy zdania *Zarówno Jan nie śpi, jak Piotr czuwa.* Spójnik równorzędny niecią-gły ZARÓWNO. . . JAK składa się ze spójnika równorzędnego lewego i prawego o tym samym oznaczeniu typu spójnika zarówno.

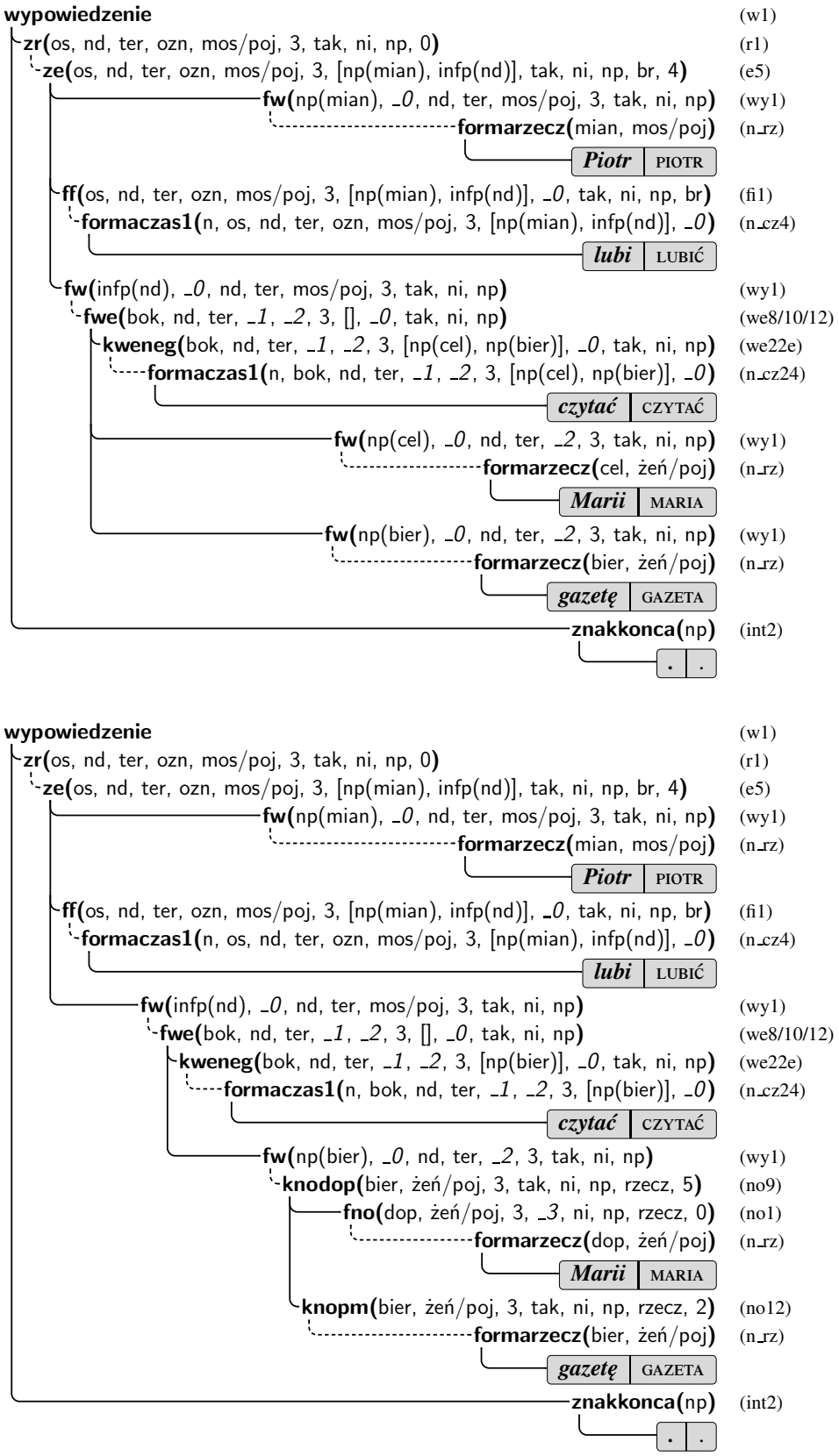


Rys. C.3. Drzewo analizy zdania *Jan śpi, Piotr natomiast czuwa.* Spójnik inkorporacyjny NATOMIAST spajający dwa zdania szeregowe **zsz** występuje wewnątrz drugiego ze spajanych zdań. Jest to sygnalizowane wartością natomiast parametru inkorporacji tego zdania, wymaganą przez regułę (r6).

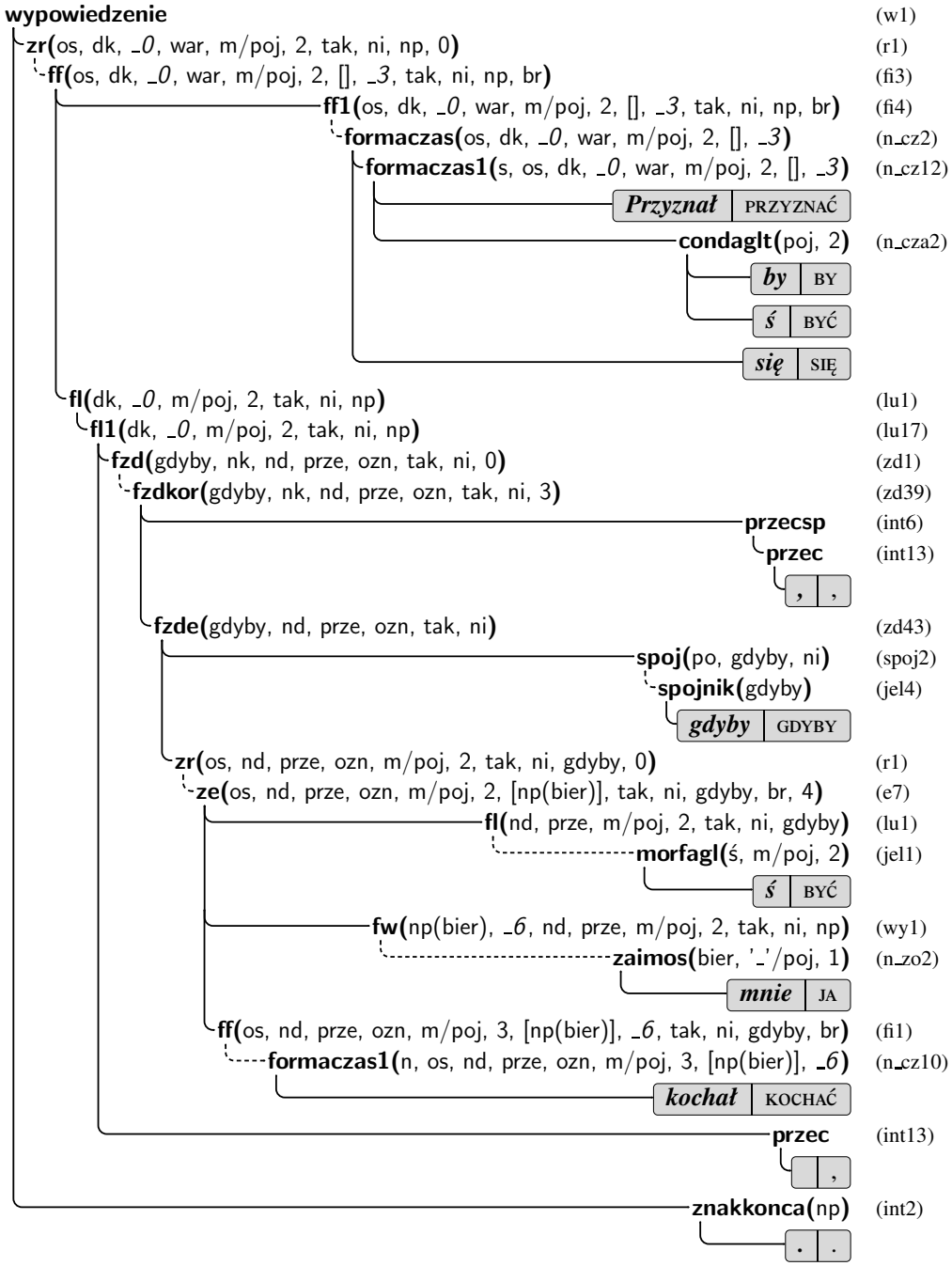
Rys. C.4. Skrócone drzewa analizy zdania *Jan przyszedł i Piotr wyszedł i Maria czytała i Tomek płakał.* Każde ze zdań elementarnych w tym wypowiedzeniu ma tylko jedną analizę. Rysunek przedstawia wszystkie sposoby połączenia ich parami za pomocą spójnika *i* (por. p. 3.1).



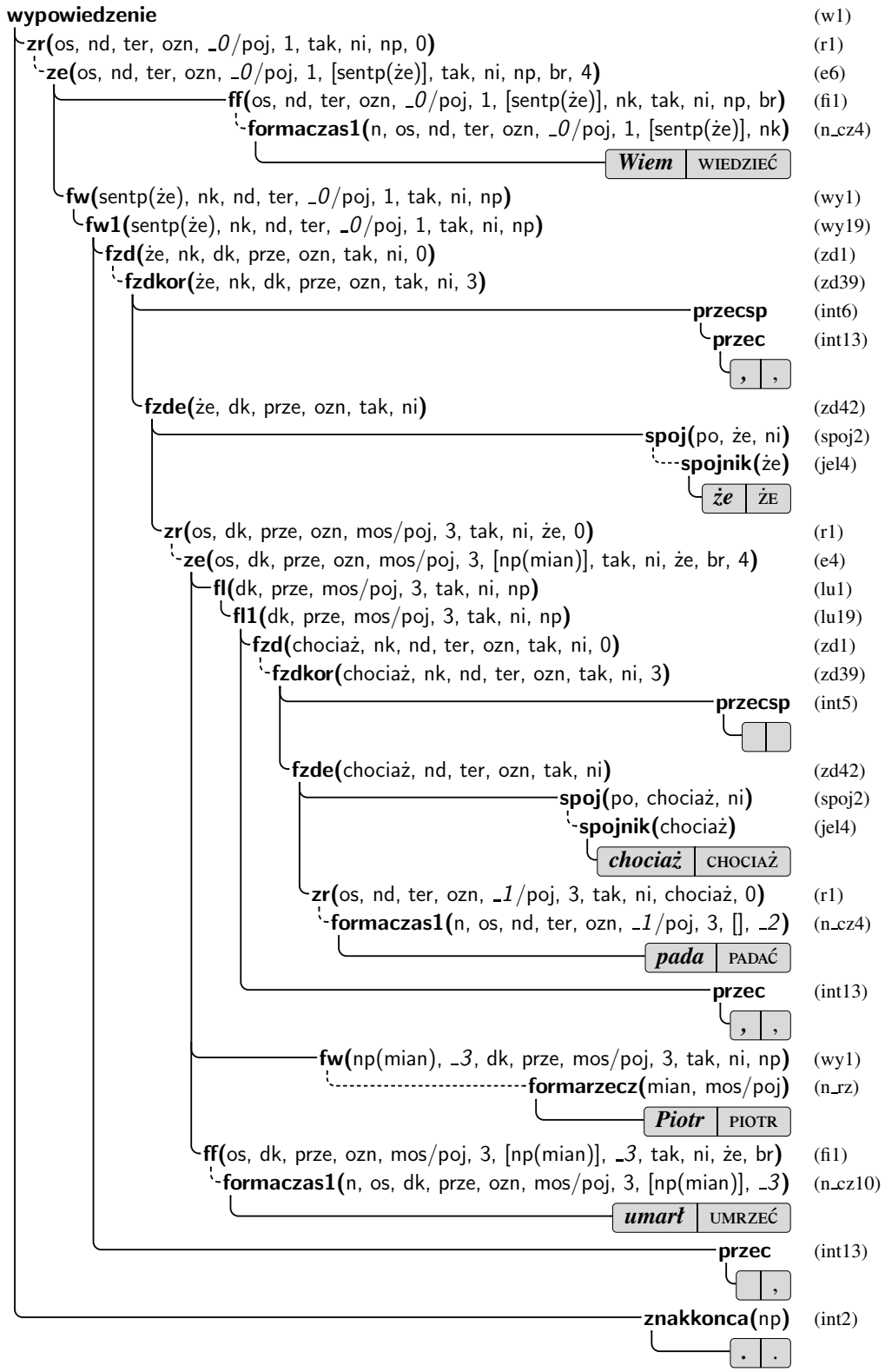
Rys. C.5. Dwa ze skróconych drzew analizy zdania *Ta poza nie wychodzi*. Interesująca jest tutaj niejednoznaczność między interpretacją segmentu *poza* jako formy rzeczownika lub przyimka (por. p. 4.6).



Rys. C.6. Dwa ze skróconych drzew analizy zdania *Piotr lubi czytać Marii gazetę*. W tych drzewach wymagania czasownika CZYTAĆ są realizowane wewnątrz frazy werbalnej **fwe**. (Por. p. 5.2.2)



Rys. C.7. Skrócone drzewo analizy zdania *Przyznałbyś się, gdybyś mnie kochał.* Zdanie podrzędne, wprowadzone spójnikiem GDYBY, stanowi tu frazę luźną, która jest realizowana przez frazę zdaniową. W jej wnętrzu występuje zdanie elementarne z inicjalną aglutynacyjną frazą luźną — jej specyficzne własności są sygnalizowane przez wartość gdyby parametru zależności.



Rys. C.8. Skrócone drzewo analizy zdania *Wiem, że chociaż pada, Piotr umarł*. Przykład zdania, w którym konieczna jest pusta realizacja jednostki **przecsp** po spójniku *że*, por. p. 5.6.

Bibliografia

H. Abramson, V. Dahl (1989) *Logic Grammars*, Symbolic Computation, Springer-Verlag.

M. Bańko (1985) *Analiza polskich fraz rzeczownikowych testem adekwatności i efektywności parsera Szpakowicza*, praca magisterska (opiekun J. S. Bień), Instytut Informatyki UW.

— (1990) *Niektóre problemy oceny adekwatności gramatyk (na przykładzie fragmentu gramatyki Szpakowicza)*, Studia Gramatyczne IX, s. 55–72.

J. Bień, K. Szafran, M. Woliński (2000) *Experimental Parsers of Polish*, w: *3. Europäische Konferenz "Formale Beschreibung slavischer Sprachen, Leipzig 1999"*, *Linguistische ArbeitsBerichte* 75, s. 185–190, Institut für Linguistik, Universität Leipzig.

J. S. Bień (1991) *Koncepcja słownikowej informacji morfologicznej i jej komputerowej weryfikacji*, Rozprawy Uniwersytetu Warszawskiego, Wydawnictwa Uniwersytetu Warszawskiego.

— (1996) *Komputerowa weryfikacja opisu składni polskiej*, TR 96-06 (227), Instytut Informatyki Uniwersytetu Warszawskiego, Warszawa.

— (1997) *Komputerowa weryfikacja formalnej gramatyki Świdzińskiego*, Biuletyn PTJ LII, s. 147–164.

— (2000) *Zestaw testów do weryfikacji i oceny analizatorów języka polskiego. Sprawozdanie merytoryczne (nieznacznie zmodyfikowane) z projektu KBN 8 T11C C 002 13*. Dokument elektroniczny: `ftp://ftp.mimuw.edu.pl/pub/users/polszczyzna/tajp`.

— (2001) *O pojęciu wyrazu morfologicznego*, w: W. Gruszczyński, *et al.* (red.), *Nie bez znaczenia... Prace ofiarowane Profesorowi Zygmuntowi Saloniemu z okazji jubileuszu 15000 dni pracy naukowej*, s. 31–42, Wydawnictwo Uniwersytetu w Białymstoku, Białystok.

— (2003) *Konstrukcje werbalne z AŻ w gramatyce Świdzińskiego*, w: *Studia z gramatyki i leksykologii języka polskiego. Prace dedykowane Profesor Marii Szupryczyńskiej*, s. 79–92, Wydawnictwo Uniwersytetu Mikołaja Kopernika.

— (2004) *An Approach to Computational Morphology*, w: M. Kłopotek, S. Wierzchoń, K. Trojanowski (red.), *Intelligent Information Processing and Web Mining*, Advances in Soft Computing, s. 191–199, Springer.

— (w druku) *Wyrazy morfologiczne i morfosyntaktyczne w praktyce*, w: *Tom jubileuszowy dla Krystyny Kallas*, .

J. S. Bień, Z. Saloni (1982) *Pojęcie wyrazu morfologicznego i jego zastosowanie do opisu fleksji polskiej (wersja wstępna)*, Prace Filologiczne XXXI, s. 31–45.

S. Billot, B. Lang (1989) *The Structure of Shared Forests in Ambiguous Parsing*, w: *Meeting of the Association for Computational Linguistics*, s. 143–151.

A. Colmerauer (1978) *Metamorphosis grammar*, w: L. Bolc (red.), *Natural Language Communication with Computers*, Lecture Notes in Computer Science 63, s. 133–189, Springer-Verlag.

J. Earley (1970) *An efficient context-free parsing algorithm*, Commun. ACM 13, nr 2, s. 94–102, ISSN 0001-0782.

G. Gazdar, C. Mellish (1989) *Natural Language Processing in Prolog*, Addison-Wesley.

Y. Matsumoto, M. Kiyono, H. Tanaka (1985) *Facilities of the BUP parsing system*, w: *Natural Language Understanding and Logic Programming*, s. 97–106, North-Holland, Amsterdam.

Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, H. Yasukawa (1983) *BUP: a bottom-up parser embedded in PROLOG*, New Generation Computing 1, s. 145–158.

C. Mellish (1988) *Implementing Systemic Classification by Unification*, Computational Linguistics 14, nr 1, s. 40–51.

G. van Noord (1995) *The Intersection of Finite State Automata and Definite Clause Grammars*, w: *Meeting of the Association for Computational Linguistics*, s. 159–165.

F. Pereira, D. H. D. Warren (1980) *Definite clause grammars for language analysis–a survey of the formalism and a comparison with augmented transition networks*, Artificial Intelligence 13, s. 231–278.

A. Przepiórkowski, P. Bański, Łukasz Dębowski, E. Hajnicz, M. Woliński (2003) *Konstrukcja korpusu IPI PAN*, Polonica XXII–XXIII, s. 33–38.

A. Przepiórkowski, M. Woliński (2003a) *A Flexemic Tagset for Polish*, w: *Proceedings of the Workshop on Morphological Processing of Slavic Languages, EACL 2003*, s. 33–40.

— (2003b) *A Morphosyntactic Tagset for Polish*, w: P. Kosta, J. Błaszczak, J. Frasek, L. Geist, M. Żygis (red.), *Investigations into Formal Slavic Linguistics (Contributions of the Fourth European Conference on Formal Description on Slavic Languages)*, s. 349–362.

— (2003c) *The Unbearable Lightness of Tagging: A Case Study in Morphosyntactic Tagging of Polish*, w: *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03), EACL 2003*, s. 109–116.

S. G. Pulman (1996) *Unification encodings of grammatical notations*, Computational Linguistics 22, nr 3, s. 295–327, ISSN 0891-2017.

Z. Saloni (1974) *Klasyfikacja gramatyczna leksemów polskich*, Język Polski LIV, z.1, s. 3–13, z.2, s. 93–101.

— (1976) *Cechy składniowe polskiego czasownika*, Ossolineum, Wrocław.

— (2000) *Wstęp do koniugacji polskiej*, Wydawnictwo Uniwersytetu Warmińsko-Mazurskiego, Olsztyn.

— (2001) *Czasownik polski. Odmiana, słownik*, Wiedza Powszechna, Warszawa.

Z. Saloni, M. Woliński (2003) *A Computerized Description of Polish Conjugation*, w: P. Kosta, J. Błaszczak, J. Frasek, L. Geist, M. Żygis (red.), *Investigations into Formal Slavic Linguistics (Contributions of the Fourth European Conference on Formal Description on Slavic Languages)*, s. 373–384.

Z. Saloni, M. Świdziński (2001) *Składnia współczesnego języka polskiego*, Wydawnictwo Naukowe PWN, Warszawa, wyd. piąte.

K. Szafran (1993) *Automatyczna analiza fleksyjna tekstu polskiego (na podstawie* Schematycznego indeksu a tergo *Jana Tokarskiego)*, Rozprawa doktorska, Wydział Polonistyki UW.

— (1996) *Analizator morfologiczny SAM-95: opis użytkowy*, TR 96-05 (226), Instytut Informatyki Uniwersytetu Warszawskiego, Warszawa.

— (1997) *Automatyczne hasłowanie tekstu polskiego*, Polonica XVIII, s. 51–63.

S. Szpakowicz (1978) *Automatyczna analiza składniowa polskich zdań pisanych*, nie publikowana rozprawa doktorska, Instytut Informatyki UW.

— (1983) *Formalny opis składniowy zdań polskich*, Wydawnictwa Uniwersytetu Warszawskiego.

S. Szpakowicz, M. Świdziński (1981a) *Formalna definicja równorzędnej grupy nominalnej we współczesnej polszczyźnie pisanej*, Maszynopis powielony. Warszawa. Przedruk jako Szpakowicz i Świdziński (1990).

— (1981b) *Zarys klasyfikacji schematów zdaniowych we współczesnej polszczyźnie pisanej*, Polonica VII, s. 5–35.

— (1982) *The syntactic structure of nominal phrases in contemporary Polish*, w: *Explizite Beschreibung und automatische Textbearbeitung*, Prague.

— (1986) *A computational description of contemporary Polish*, w: A. Bogusławski, B. Bojar (red.), *Od kodu do kodu*, s. 155–162, Warszawa.

— (1990) *Formalna definicja równorzędnej grupy nominalnej we współczesnej polszczyźnie pisanej*, Studia Gramatyczne IX, s. 9–54.

M. Świdziński (1983) *Tzw. wypowiedzenia złożone w gramatyce formalnej współczesnej polszczyzny pisanej*, w: *Język. Teoria — dydaktyka VI*, s. 3–41, Kielce.

— (1986) *Elipsa w gramatyce formalnej współczesnej polszczyzny pisanej*, Prace Filologiczne XXXIII, s. 357–364.

— (1987) *Formalny opis składniowy polskich zdań o składniku zdaniowym*, praca habilitacyjna, Wydział Polonistyki UW, Warszawa. (maszynopis powielony).

— (1992) *Gramatyka formalna języka polskiego*, Rozprawy Uniwersytetu Warszawskiego, Wydawnictwa Uniwersytetu Warszawskiego, Warszawa.

— (1993) *Od interpretacji do faktów językowych: weryfikacja empiryczna gramatyki formalnej*, Biuletyn PTJ XLIX, s. 15–24.

— (2000) *Negativity Transmission in Polish Constructions with Participles and Gerunds*, Journal of Slavic Linguistics 8, s. 263–293.

J. Tokarski (2002) *Schematyczny indeks a tergo polskich form wyrazowych*, red. Zygmunt Saloni, Wydawnictwo Naukowe PWN, Warszawa, wyd. drugie.

L. Valient (1975) *General Context Free Recognition in Less Than Cubic Time*, J. Computer and System Sciences 10, s. 308–315.

M. Woliński, A. Przepiórkowski (2001) *Projekt anotacji morfosynktaktycznej korpusu języka polskiego*, Prace IPI PAN 938, Instytut Podstaw Informatyki Polskiej Akademii Nauk.

M. Woliński (2003) *System znaczników morfosyntaktycznych w korpusie IPI PAN*, Polonica XXII–XXIII, s. 39–55.