ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: GEOMETRIA I GRAFIKA INŻYNIERSKA z.2 Nr kol. 1383

ŻYNEL Mariusz
Instytut Matematyki
Uniwersytet w Białymstoku

KOMPUTEROWO WSPOMAGANA GEOMETRIA: WERYFIKACJA, AUTOMATYZACJA, WIZUALIZACJA[1)]

Streszczenie. Wizualizacja nie zawsze jest wystarczającym narzędziem w geometrii, czasem pojawia się potrzeba weryfikacji logicznej poprawności rozumowania. Większość zagadnień dotyczących skończonych geometrii ma czysto kombinatoryczny charakter. Uzasadnia to zastosowanie w geometrii komputerowych systemów weryfikacji oraz automatyzacji dowodzenia.

COMPUTER AIDED GEOMETRY: VERIFICATION, AUTOMATIZATION, VISUALIZATION

Summary. Sometimes visualization is not sufficient in geometry and verification of logical correctness of a reasoning is necessary. Most problems concerning finite geometry are of purely combinatorial nature, so that reasonings could be automated. This justifies the application of the proof-checkers and provers in geometry.

1. Wstęp

Bardzo szybki rozwój informatyki oraz znaczne obniżenie kosztów produkcji sprzętu komputerowego powodują, że komputery znajdują bardzo wiele zastosowań we współczesnym świecie, a przykład Deep Blue pokazuje, jak wiele one dzisiaj potrafią. Zastosowanie komputerów w geometrii, ale i w matematyce w ogóle, ma trzy

[1)] Praca została wykonana w ramach grantu KBN : 8T11C01812

aspekty. Buduje się systemy sprawdzające poprawność rozumowań człowieka. Istnieje też wiele mniejszych lub większych systemów, które automatyzują procesy dedukcyjne. Zagadnienie weryfikacji wiąże się z koncepcją encyklopedii usystematyzowanej wiedzy matematycznej. Natomiast ogromna mnogość programów inżynierskich typu CAD pozwala wykorzystać komputery w geometrii do wizualizacji.

2. System weryfikacji rozumowań

Problem logicznej weryfikacji rozumowań w geometrii pojawia się równie często, o ile nie częściej, niż problem trafnej interpretacji graficznej. Mamy z nim do czynienia w algebraicznym dowodzie twierdzenia Pascala-Brianchona w analitycznej przestrzeni rzutowej, jak i w konstrukcji punktu Lemoine'a w trójkącie.

Idea automatyzowania weryfikacji rozumowań nie jest nowa. Sięga ona czasów Pascala i Leibnitza. Pierwsze próby budowy maszyny logicznej podejmowane były już w XIII wieku przez Ramona Llull.

Jednym z systemów służących do formułowania i weryfikacji logicznej dowodów jest Mizar. Jest on jedynym, który posiada bibliotekę. Autorem systemu jest Andrzej Trybulec. Początki projektu Mizar sięgają roku 1973. Pierwsza implementacja robiona była jeszcze na kartach perforowanych. System Mizar obejmuje trzy elementy: język, oprogramowanie i bazę danych.

Język Mizar jest formalnym językiem wywodzącym się z matematycznego żargonu. Precyzyjna gramatyka zapewnia czytanie i kontrolę logicznej poprawności przez komputer. Na poziomie semantycznym język Mizar jest bardzo bliski żargonowi matematycznemu, natomiast na poziomie syntaktycznym jest mocno uproszczony, co nie powoduje komplikacji składni. Zamiast niektórych fraz używa się nawet skrótów. Na przykład w formule egzystencjalnej zamiast słowa *exists* pisze się **ex**, a zamiast *such that* krócej - **st**.

W przeciągu kilkudziesięciu lat wykształciła się specyficzna terminologia języka Mizar. I tak, skrypt napisany w języku Mizar nazywa się *artykułem mizarowym*. Składa się on z dwóch części: *deklaracji środowiska* i *tekstu właściwego*. Deklaracja środowiska rozpoczyna się słowem **environ** i zawiera (ewentualnie pusty) zestaw *dyrektyw*, które określają, jakie informacje zgromadzone w bazie danych mają być za-

importowane do wykorzystania w artykule. Dyrektywa **vocabulary** określa zestaw słowników, z których pobrane zostaną symbole. Dla każdego symbolu niezbędna jest informacja o liczbie i typach argumentów, które mogą być z nim użyte. Dyrektywami określającymi, z jakich artykułów należy pobrać te informacje, są odpowiednio **constructors i notation**.

Mizar odróżnia pojęcia zdefiniowane przy użyciu jednego symbolu, ale z różną liczbą argumentów lub różnymi typami argumentów. Na przykład w zależności od kontekstu relacja $a|k$ może być zinterpretowana jako a incyduje z k lub a dzieli k. Zależy to od deklaracji środowiska oraz od typów zmiennych a i k.

Wśród dyrektyw środowiskowych są dyrektywy **clusters** i **definitions**. Pozwalają one dowodzić poprzez rozwinięcie definicyjne automatycznie, co eliminuje konieczność powoływania się na twierdzenia definicyjne. Pojęciem *klaster* określa się zestaw atrybutów. Każdy typ powstaje przez dołączenie do modu, ewentualnie pustego, klastra atrybutów.

Wyróżniamy trzy rodzaje klastrów. Klastry egzystencjalne stwierdzają istnienie obiektu o żądanym typie, np.:

```
definition
  cluster strict partial linear up-2-dimensional up-3-rank
  Vebleian IncProjStr;
end;
```

stwierdza istnienie przestrzeni rzutowej. Klastry warunkowe stwierdzają, że obiekt posiada pewne własności - atrybuty, pod warunkiem, że posiada inne, mocniejsze, np.:

```
definition
  cluster Pappian → Desarguesian IncProjSp;
end;
```

jest równoważne twierdzeniu:

```
  for PS being Pappian IncProjSp holds PS is Desarguesian Inc-
ProjSp;
```

mówiącemu, że każda papusowa przestrzeń rzutowa jest przestrzenią desarguesową. Lista atrybutów w poprzedniku może być pusta, np.:

```
definition
  let X be empty set;
  cluster → empty Subset of X;
end;
```

jest równoważne z faktem, że każdy podzbiór zbioru pustego jest pusty. I wreszcie trzeci rodzaj to klastry funktorowe określające typ wyniku operacji, np:

```
definition
  let CPS be CollProjectiveSpace;
  cluster IncProjSp_of (CPS) → partial linear up-2-dimensional
  up-3-rank Vebleian;
end;
```

stwierdza, że struktura incydencyjna określona na przestrzeni rzutowej zdefiniowanej w terminach relacji współliniowości posiada wszystkie atrybuty przestrzeni rzutowej. Mechanizm klastrów jest bardzo silny. Poza tym, że pozwala wnioskować przez rozwinięcie definicyjne, to typ jest rozszerzany automatycznie.

Dyrektywa **theorems** umożliwia powołania na twierdzenia, a dyrektywa **schemes** na schematy w wymienionych artykułach. Schematy są to twierdzenia, w których występują zmienne wolne drugiego rzędu. Dobrym przykładem schematu jest schemat indukcji naturalnej sformułowany następująco:

```
scheme Ind { P[Nat] } :  for k holds P [k]
  provided
  P[0] and
  for k st P[k] holds P [k + 1];
```

Jest jeszcze jedna dyrektywa **requirements** o specjalnym znaczeniu.

W tekście właściwym artykułu mizarowego wydzielić można *bloki*. Słowo **begin** rozpoczyna sekcję. W artykule mizarowym musi być przynajmniej jedna sekcja. Podział artykułu na sekcje podyktowany jest wyłącznie względami edytorskimi i nie ma wpływu na poprawność tekstu. Najmniejszym, poprawnym artykułem mizarowym jest więc tekst zawierający dwa słowa:

```
environ
begin
```

Rozróżnia się następujące bloki:

rezerwacja - miejsce, gdzie przypisywane są typy zmiennym globalnym,

definicja - miejsce, gdzie definuje się konstruktory; konstruktory termów nazywane są *funktorami*, konstruktory formuł nazywane są *predykatami*, a konstruktory typów *modami*,

definicja struktury - miejsce wprowadzania nowych struktur -obiektów składających się z pól o zadanym typie, do których odwołuje się poprzez selektory pól,

twierdzenie - fakt eksportowany do bazy,

schemat - twierdzenie zawierające zmienne wolne drugiego rzędu, eksportowane do bazy,

pomocniczy item - lokalne twierdzenie lub definicja obiektu prywatnego, nie eksportowane do bazy.

Przykładem definicji struktury incydencyjnej przestrzeni rzutowej jest poniższa definicja z artykułu [1]:

```
struct IncProjStr (# Points → non empty set,
                     Lines → non empty set,
                     Inc → Relation of the  Points, the Lines #);
```

Podział artykułu mizarowego na dwie części ma jeszcze jeden, bardziej techniczny charakter. W systemie Mizar są dwa oddzielne programy, które przetwarzają artykuł. Deklaracja środowiska przetwarzana jest przez program zwany *akomodatorem*. Na podstawie deklaracji przygotowuje z bazy danych informacje, które będą wykorzystane przez program *weryfikator* przy obróbce tekstu właściwego. Weryfikator dokonuje analizy syntaktycznej, zgodności wzorców konstruktorów, zgodności typów oraz analizy logicznej poprawności wnioskowań. Wynikiem jego działania jest, ewewntualnie pusta, lista wykrytych błędów w tekście właściwym. W weryfikatorze zastosowano technikę *errors recovery*, co pozwala nie przerywać pracy programu po wystąpieniu pierwszego błędu. Może się jednak zdarzyć, że po wykryciu błędu w tekście po nim następującym odnotowany zostanie następny błąd, albo że błąd zostanie źle zlokalizowany. Na przykład niedomknięcie itemu średnikiem może spowodować wystąpienie błędów w poprawnym tekście za nim. Problem ten jest charakterystyczny dla wszystkich języków programowania.

Artukuł mizarowy jest zwykłym plikiem tekstowym w rozszerzonym ASCII. Są oczywiście specyzowane ograniczenia, jeśli chodzi o stosowanie znaków kontrolnych,

identyfikatorów i symboli. Na przykład znak U został zarezerwowany jako suma mnogościowa pary zbiorów, więc nie może on być użyty jako identyfikator. Podobnie symbole L oraz B zostały zarezerwowane w słownikach odpowiednio dla relacji współliniowości i leżenia między. Wydaje się, że właściwą polityką jest zakaz wprowadzania symboli złożonych z mniej niż trzech znaków. Długość linii tekstu źródłowego ograniczona jest do 80 znaków.

W artykule miazarowym dowodzi się twierdzenia, definuje[2)] nowe pojęcia i obiekty, które mogą być wykorzystane w innych artykułach. Od 1989 roku poprawne logicznie artykuły mizarowe gromadzone są w postaci specjalnej bazy danych, zwanej *Main Mathematical Library*. Podstawą dla tej bibiloteki jest aksjomatyka teorii mnogości Tarskiego-Grothendiecka oraz arytmetyka liczb naturalnych i rzeczywistych. Dedukcja naturalna w Mizarze oparta jest na systemie Jaśkowskiego z wykorzystaniem w dowodach tak zwanych *stałych lokalnych*. Warto tutaj podkreślić, że w MML znajdują się artykuły nie tylko matematyczne. Jest w niej pokaźna seria artykułów z zakresu dowodzenia poprawności programów.

W momencie przyjmowania artykułu do bazy część eksportowalna jest wyabstrahowywana i przechowywana w postaci pliku zwanego *abstraktem*. Usuwane są wszystkie dowody oraz lokalne lematy i prywatne definicje. Program, który generuje pliki bazy danych - *extractor*, jest rozpowszechniany w pakiecie Mizara. Możliwe jest więc budowanie własnych baz danych.

Sukcesywnie artykuły mizarowe, a właściwie ich abstrakty, tłumaczone są automatycznie na język angielski i publikowane *w Formalized Mathematics*. Od 1996 roku, poza wydawnictwem papierowym, wiedza zgromadzona w bibliotece reprezentowana jest w internetowym wydawnictwie elektronicznym *Journal of Formalized Mathematics* jako w pełni hypertekstowe dokumenty HTML.

W tej chwili w MML znajduje się 477[3)] artykułów, napisanych przez ponad 70 autorów. Większość artykułów jest na poziomie podstaw, choć są tam oryginalne, nowe wyniki z topologii. W MML można znaleźć efekt formalizacji geometrii w postaci około 50 artykułów napisanych przez 15 autorów. Tematyka tych prac skupiona jest na przestrzeniach afinicznych i rzutowych w ujęciu syntaktycznym i analitycznym

[2)] Poprawność definicji jest również dowodzona.

[3)] Stan na 2 czerwca 1997 r.

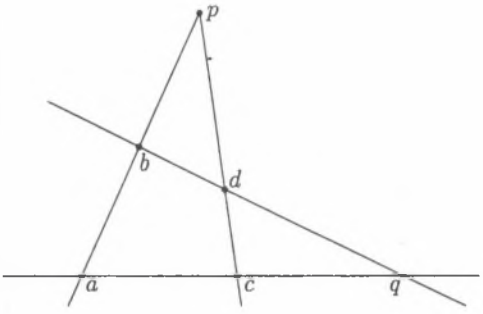
oraz na przekształceniach w tych geometriach. Rozpoczęte są prace nad formalizacją bieżących aktualnych wyników.

Artykuły z zakresu geometrii są dość stare. Powstawały, gdy nie było w Mizarze jeszcze wielu dostępnych obecnie mechanizmów, między innymi atrybutów i klastrów. Formalizacja jest zadaniem trudnym. Poza tym, że wymaga znajomości reguł gramatycznych i technicznych, to wymaga jeszcze umiejętności doboru elastycznej reprezentacji zagadnień w języku Mizar. Należy też pisząc artykuł zwracać uwagę na formaty wprowadzanych pojęć. Ważne jest, aby można je było czytelnie odzwierciedlić w *Formalized Mathematics* w postaci TEX'a. Jest to kwestia stylu pisania artykułu i dotyczy również wydawnictw tradycyjnych. Ponieważ Mizar jest automatem, to dbałość o szczegóły powinna być znacznie większa. Drobne sprawy mogą mieć tutaj ogromne następstwa. Dobrym przykładem jest rewizja bibiloteki, która miała na celu usunięcie założenia o niepustości nośnika w strukturze przestrzeni topologicznej. W kilka lat po napisaniu artykułu, w którym została zdefiniowana przestrzeń topologiczna, przy miksowaniu topologii z teorią krat, okazało się, że warto rozpatrywać przestrzenie topologiczne o pustych nośnikach.

Charakterystyczną trudnością, która pojawia się przy formalizacji geometrii, jest duża ilość zmiennych w twierdzeniach i definicjach, a co za tym idzie, obszerność i słaba czytelność formuł. Rozważmy rzutowy aksjomat Veblena w geometrii, której strukturę określa `IncProjStr`. Zgodnie ze składnią Mizara atrybut, mówiący, że taka struktura spełnia warunek Veblena, możemy zdefiniować następująco:

```
definition
  let IPS be IncProjStr;
  attr IPS is  Vebleian means
  for a, b, c, d, p , q being POINT of IPS
  for M, N, P, Q being LINE of IPS st a|M & b|M & c|N & d|N &
  p|M & p|N & a|P & c|P & b|Q & d|Q &  not p|P & not p|Q & M<>N
  holds
  ex q being POINT of IPS st q|P & q|Q;
end;
```

W powyższym zdaniu występuje 10 zmiennych - 6 punktów i 4 proste. Dopiero rysunek 1 wyjaśnia, o co właściwie chodzi.



Rys.1. Rzutowy aksjomat Veblena

Formułę można nieco skrócić, wprowadzając dodatkowe dwa predykaty a,b|P oraz a|P,Q, stwierdzające odpowiednio, że punkty a i b leżą na prostej P oraz punkt a leży na prostych P i Q. Jeszcze lepszym rozwiązaniem jest określenie relacji przecinania się prostych, to znaczy:

```
definition
  let IPS be  IncProjStr;
  let P, Q be LINE of  IPS;
  pred P intersects Q means
    ex a being POINT of IPS st a|P, Q;
  symmetry;
end;
```

Dodefiniowując taki predykat dla większej liczby argumentów można definicji atrybutu nadać następujący wygląd:

```
definition
  let IPS be IncProjStr;
  attr IPS is  Vebleian means
  for p being POINT of IPS
  for M, N, P, Q being LINE of IPS st p|M, N  & not p|P, Q &
M<>N
       & M, N intersect P, Q
```

```
  holds
    P intersects Q;
end;
```

Liczbę zmiennych ograniczyliśmy do 5. Poprawiona chyba została również czytelność twierdzenia definicyjnego. Jedyną wadą takiego rozwiązania jest to, że definicja nie jest wyrażona w terminach elementarnych.

Mechanizm atrybutów pozwala wprowadzać nowe typy przez dołączanie do typów macierzystych zestawów atrybutów. Pozwala to uniknąć definicji, gdzie twierdzenie definicyjne jest koniunkcją 14 zdań, jak w artykule [2]. Mechanizm klastrów pozwala zawężać typ, w wyniku czego otrzymujemy bardziej szczególne klasy obiektów, pochodnych od obiektów macierzystych. W wyniku rewizji w artykule [1] definicje przestrzeni rzutowej i jej wariantów zostały rozbite na atrybuty. Wydaje się, że powinno to być również zrobione z pozostałymi artykułami, nie tylko z geometrii.

Język Mizara jest w tej chwili mocno ustabilizowany. Zmiany w gramatyce są niezwykle rzadkie. Znacznie częstsze zmiany robione są w oprogramowaniu i bibliotece. Dotyczą one zwykle spraw technicznych.

3. Automatyczne dowodzenie własności konfiguracji

W klasycznej geometrii afinicznej i rzutowej ważną rolę odgrywają twierdzenia konfiguracyjne. Twierdzenia te mówią, że w ustalonym zbiorze punktów i prostych, jeśli spełnione są założenia incydencyjne, pewien punkt incyduje z pewną prostą, innymi słowy - konfiguracja domyka się. Formalnie twierdzenia te mają następującą postać:

Dane są punkty $a_1, a_2, \ldots, a_n$, *proste* $A_1, A_2, \ldots, A_m$ *oraz relacja:*

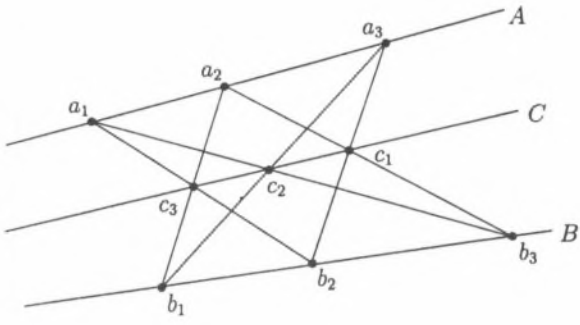
$$I \subseteq \{1, \ldots, n\} \times \{1, \ldots, m\}.$$

Ustalmy pewną parę $(i_0, j_0) \in \{1, \ldots, n\} \times \{1, \ldots, m\} \setminus I$ *nie będącą w relacji I. Zakładamy, że jeśli* $(i,j) \in I$, *to punkt* a_i *leży na prostej* A_j *oraz że jeśli* $(i,j) \notin I$ *i para* (i,j) *jest różna od pary* (i_0, j_0), *to prosta* A_j *nie przechodzi przez punkt* a_i. *Zakładamy też dodatkowo, że dane punkty i proste są parami różne. Wnioskujemy stąd, że punkt* a_{i_0} *leży na prostej* A_{j_0}.

Dodatkowe założenie o tym, że dane punkty i proste są parami różne, nazywamy warunkiem nietrywialności. Przykładem twierdzenia konfiguracyjnego jest twierdzenie Pappusa:

Jeśli $a_1,a_2,a_3|A$, $b_1,b_2,b_3|B$, $a_1,b_2|D_{12}$, $a_2,b_1|D_{21}$, $a_1,b_3|D_{13}$, $a_3,b_1|D_{31}$, $a_2,b_3|D_{23}$, $a_3,b_2|D_{32}$, $c_1|D_{23}$, D_{32}, $c_2|D_{13}$, D_{31}, $c_3|D_{12}$, D_{21}, $c_1,c_2|C$, *to* $c_3|C$.

W powyższym przykładzie warunek nietrywialności został opuszczony. W praktyce, by zagwarantować poprawność aksjomatów konfiguracyjnych, można warunki nietrywialności osłabić jeszcze bardziej i dopuścić incydencje spoza zbioru I.



Rys.2. Aksjomat Pappusa

Inne typowe twierdzenia konfiguracyjne to aksjomat Desarguesa i aksjomat Fano. W geometrii rzutowej ważną rolę odgrywa również warunek Veblena i aksjomat o sieci. Nie są one twierdzeniami konfiguracyjnymi z uwagi na to, że ich tezy są zdaniami egzystencjalnymi.

Zwykle wizualizacja takich twierdzeń (rys.2) nie nastręcza wiele trudności. W praktyce, gdy te twierdzenia stosujemy, rzecz się komplikuje. Pojawiają się dodatkowe punkty, proste oraz powiązania incydencyjne. Nawet przy starannym rysunku trudno jest dostrzec odpowiednią podkonfigurację, spełniającą założenia. Z drugiej strony, problem jest czysto kombinatoryczny. Można by zatem do pracy użyć komputera. W tym właśnie celu został skonstruowany system Ariadna. Jego autorem jest Krzysztof Prażmowski.

Pracę z systemem Ariadna rozpoczyna się przygotowując *tekst źródłowy*, zawierający opis konfiguracji oraz serię dyspozycji. Tekst ten musi być sformułowany zgod-

nie z gramatyką języka systemu (por.[5]). W opisie konfiguracji określa się uniwersum punktów i prostych przez odpowiednią rezerwację identyfikatorów oraz w formie założeń określa się relacje, jakie zachodzą między nimi.

```
reserve a, b, c, d  for point;
reserve M, N, P, Q for line;
asumptions a|P & a|Q & P<> Q & b|M & b|N;
```

Relacjami tymi mogą być incydencję i równości lub ich negacje. Nie ma przy tym wymogu, aby określone były wszystkie relacje. O punkcie a i prostej P możemy nie wiedzieć, czy ze sobą incydują czy nie. Powyższe informacje można obrazowo przedstawić w postaci trzech macierzy. Wiersze pierwszej z nich odpowiadają prostym, a kolumny punktom. Każdy jej wyraz przyjmuje jedną z trzech wartości odpowiadających incydencji, nieincydencji lub nieokreśloności. Druga macierz niesie informację o równości punktów, a trzecia o równości prostych. Wyrazy tych macierzy również przyjmują jedną z trzech wartości odpowiadających równości, różności lub nieokreśloności relacji. Przyjęto następującą umowę:

- 1 - relacja nie zachodzi,
0 - relacja jest nieokreślona,
1 - relacja zachodzi.

Zauważmy, że macierz druga i trzecia są kwadratowymi macierzami symetrycznymi, z jedynkami na przekątnej. Takie macierze generowane są automatycznie przez program analizujący poprawność syntaktyczną i kodujący informacje wejściowe na podstawie tekstu źródłowego.

W pierwszym kroku, po sprawdzeniu poprawności syntaktycznej tekstu źródłowego, system znajduje bezpośrednie konsekwencje przyjętych założeń w definicji konfiguracji. Potrzebna jest w tym celu bardzo słaba logika, oparta na dwóch regułach:
ekstensjonalność - jeśli a = b, P = Q i a|P (a ∤ P), to b|Q (b ∤ Q), wraz ze zwrotnością, symetrią i przechodniością równości,
PLS - aksjomat częściowej przestrzeni prostych - jeśli a,b|P,Q, to a=b lub P=Q.

W praktyce system stosuje wszystkie warianty PLS. Może się zdarzyć, że już na tym etapie została stwierdzona sprzeczność. Dalsze przetwarzanie w takiej sytuacji nie ma sensu i jest przerywane.

Dalej w tekście źródłowym umieszczone są dyspozycje dla systemu. Sa cztery typy dyspozycji. Pierwszy z nich to zapytania proste. Zapytanie formułujemy podając,

jakie interesują nas obiekty zdefiniowanej konfiguracji, jak relacja zachodząca między nimi i w oparciu o jaki aksjomat ma być wywnioskowana. Możliwe są pytania proste w oparciu wyłącznie o definicję konfiguracji. Relacje, o które możemy pytać, to współliniowość, współpękowość, incydencje oraz przecinanie się. Wbudowanymi w system aksjomatami są: aksjomat Desarguesa, Pappusa, Fano, warunek Veblena i aksjomat o sieci. Użytkownik może również zdefiniować własne aksjomaty i dowodzić na ich podstawie. Zapytanie może wyglądać następująco:

```
question collinear a, b, c  by Desargues;
```

Jako odpowiedź możemy uzyskać:

```
3-tuples (a1,a2,a3)and (b1,b2,b3)are called perspective trian-
gles,
and
point o is called the perspective  center of the Desargues Con-
figuration;
```

Drugim typem dyspozycji są zapytania o wskazówki. Są to pytania w nieco osłabionej wersji. Szukana podkonfiguracja nie musi spełniać warunków nietrywialności. W odpowiedzi dostajemy informację, jakie dodatkowe relacje muszą być spełnione, aby żądana relacja mogła być dowiedziona. Składnia tej dyspozycji jest taka, jak zapytania, z tym że słowo identyfikujące konstrukcję **question** zastąpione jest słowem **hint**.

Zarówno w zapytaniach, jak i zapytaniach o wskazówki system wykonuje tylko jeden przebieg, tzn. odpowiedni aksjomat stosowany jest tylko raz, a przesłanki, na których opiera się system, są to wyłącznie informacje z opisu konfiguracji i ich bezpośrednie konsekwencje otrzymane przez zastosowanie reguły ekstensjonalności i aksjomatu PLS. Dyspozycja typu zapytania z założeniem, identyfikowana przez system jako konstrukcja **moreover**, powoduje dynamiczne powiększenie definicji konfiguracji o dodatkowe obiekty i relacje. W pierwszym kroku system dowodzi poprawności dołączenia obiektów i relacji na podstawie zadanego aksjomatu, a w drugim sprawdza konsekwencje tego dołączenia. W przypadku stwierdzenia sprzeczności system przerywa pracę.

Przedstawione dyspozycje nie zmieniają uniwersum. System pytany jest o różne własności konfiguracji oraz symuluje rosnącą o niej wiedzę. W praktyce rozważa się pomocnicze obiekty oraz ustala relacje, jakie zachodzą między oryginalnymi i po-

mocniczymi obiektami. Oczywiście, rozważany aksjomat musi uzasadniać istnienie obiektów pomocniczych. Jeśli na przykład badamy konfigurację spełniającą aksjomat Pappusa, to punkty przecięć przeciwległych boków sześciokąta automatycznie traktujemy jako współliniowe, nawet gdy odpowiednia prosta nie jest w definicji konfiguracji. Rozważany aksjomat jest wtedy traktowany jako stwierdzenie istnienia pewnych punktów i prostych związanych pewnymi relacjami. W systemie Ariadna możemy formułować dyspozycje typu zagadnienia, które dołączają do definicji konfiguracji obiekty pomocnicze. Zagadnienia rozpoczynają się w tekście źródłowym słowem **problem**. Po wydaniu takiej dyspozycji system próbuje możliwie wiele razy stosować żądany aksjomat, by domykać konfigurację.

Aktualnie system Ariadna ma znaczne ograniczenia co do rozmiarów badanych konfiguracji. Maksymalnie może wystąpić 40 punktów i 40 prostych. Obecne prace nad systemem mają na celu umożliwienie badania znacznie większych konfiguracji. Przekodowanie oprogramowania na pracę w trybie chronionego dostępu do pamięci pozwoli używać większych zasobów o pamięci. Poprawiony też ma być interfejs użytkownika, a system ma pracować w środowisku Windows.

System Ariadna wyposażony jest w interfejs do współpracy z Mizarem. Można zażądać od systemu, aby wygenerował sformułowane twierdzenie wraz z dowodem, zgodnie ze specyfikacją języka Mizar.

Konfigurację można charakteryzować przez podanie jej parametrów numerycznych, tzn. ilości punktów, prostych, rzędu punktów i prostych, długości nieskracalnych łańcuchów prostych, łączących dwa punkty i wiele innych. Badanie skończonych geometrii pociąga za sobą potrzebę kombinatorycznego przeliczania różnych parametrów. Znajdują tutaj zastosowanie systemy przetwarzania symbolicznego. Bardzo użyteczny w tym względzie jest Maple. Nie tylko usprawnia rachunki, ale ułatwia również prace edytorskie przy przygotowywaniu publikacji.

4. Wizualizacja konfiguracji

Zagadnienie rozwiązywania konfiguracji geometrycznych wiąże się z problemem wizualizacji tychże konfiguracji. Dosyć łatwo rysuje się konfiguracje o liczbie punktów i prostych nie przekraczającej 10. Trudności zaczynają się wraz z komplikacją konfi-

guracji. Zauważmy, że spośród uniwersum punktów i prostych dowolnie możemy narysować wyłącznie dwie proste i dwa punkty. Położenie reszty z nich może być jednoznacznie zdeterminowane tym wyborem. Otrzymamy co najwyżej konfiguracje izomorficzne z dokładnością do obrotu i jednokładności.

Wydaje się, że algorytm znajdowania reprezentacji graficznej oparty winien być na rozwiązaniach układów równań liniowych. Niestety, jak dotąd nie udało się opracować niezawodnego algorytmu. Istniejące rozwiązania radzą sobie jedynie z małymi konfiguracjami.

LITERATURA

1. LEOŃCZUK W., PRAŻMOWSKI K.: Incidence Projective Spaces, Formalized Mathematics, 2(3), 1990
2. ORYSZCZYSZYN H., PRAŻMOWSKI K.: a Construction of Analytical Ordered Trapezium Spaces, Formalized Mathematics, 2(4), 1990
3. PRAŻMOWSKI K.: Geometric Background of the Theory of Configurations for the system Ariadna, Ariadna system documentation
4. PRAŻMOWSKI K.: System Ariadna for Handling with Configurations and Its Basic Parts, Ariadna system documentation
5. PRAŻMOWSKI K.: Grammar of *. cnf texts, Ariadna system documentation
6. TRYBULEC A.: Some Features of the Mizar Language, ESPRIT Workshop, Torino, 1993, available from MizUG
7. TRYBULEC A.: The Mizar Project, manuscript
8. TRYBULEC A., RUDNICKI P.: A Collection of TeXed Mizar Abstracts, University of Alberta, Canada, 1989

Recenzent: Dr hab. Krzysztof Prażmowski
Uniwersytet w Białymstoku

Abstract

The obvious application of computers in geometry, especially in engineer geometry, is visualization and construction assistance. However, verification of logical correctness of a reasoning seems to be so necessary as the accurate graphic interpretation. One of the proof-checking systems is Mizar. It provides the metalanguage close to mathematical vernacular and rigorous enough to enable processing and verifying by a computer software. The goal of writing Mizar article is to prove theorems and introduce new concepts. Logically valid Mizar articles are stored in the data-base so that, they can be referenced by authors. Geometry, and foundations of geometry in particular, occurred hard to formalize. It requires some exceptional methods to be worked out.

In classic affine and projective geometry the configurational theorems are often discussed. Usually, it is relatively easy to visualize such theorems as appropriate configurations. In practice, when the universe of points and lines is large, it may be difficult to find the consequences of configurational theorems. Visualization is not sufficient. On the other hand, the nature of problems concerning finite geometry is purely combinatorial. Therefore, the automated reasoning is possible. The computer system Ariadna supports finding proofs based on configurational axioms. A geometry is defined by an incidency relation and a set of conditions. Ariadna finds the consequences of the assumptions and then tries to answer the questions.