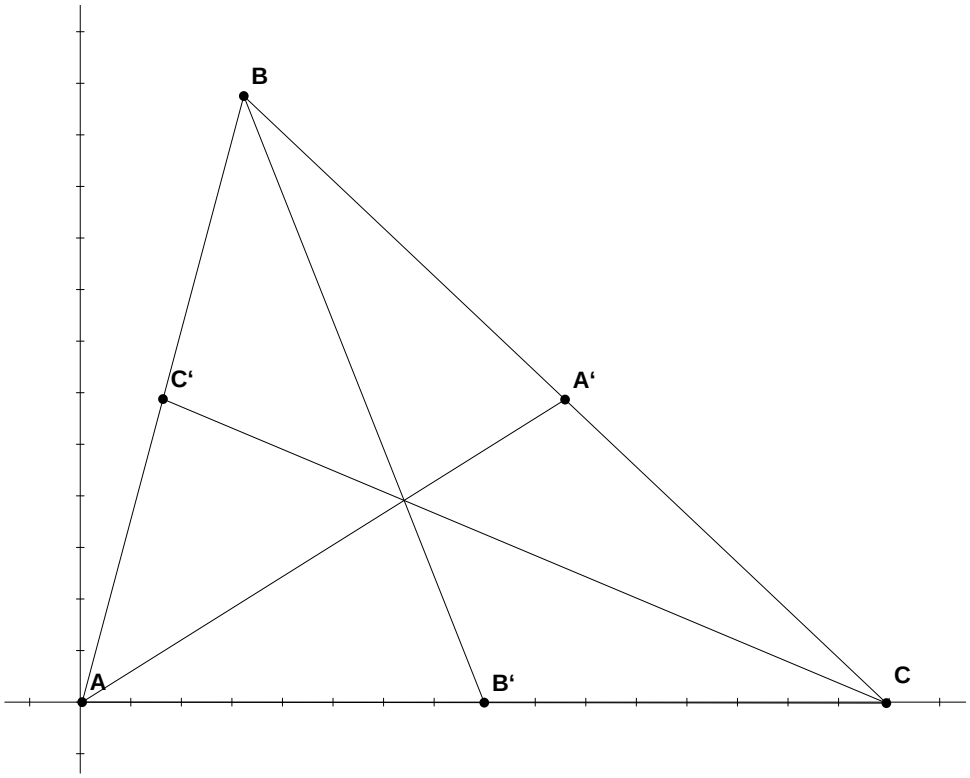
SINGULAR ALGORITHMS FOR WU'S METHOD

CHRIS STRADER

1. Introduction to Automated Theorem Proving

The main tool used when studying polynomial systems and doing polynomial computations is a Gröebner basis. However, a Gröebner basis computation can often require a vast amount of computer resources. This can be problematic since there are many applications that can be modelled with polynomial systems. One such application is proving geometric theorems mechanically. The basic idea is to take a constructive geometric statement and translate the geometry into a set of polynomial equations. Since a computer can work with algebraic statements, algorithms can be constructed that then prove the theorem algebraically.

In order to see how the conversion of geometric theorems to algebraic statements is done consider the Median Theorem: The medians of a triangle are concurrent.



By choosing the coordinate axes appropriately the translation into polynomials becomes quite simple. If the x-axis is one side of the triangle with a vertex as the origin then one will be left with 3 free parameter choices and 7 fixed parameters.The notation we use throughout this paper is that the u_i's are freely chosen parameters and the x_j's are fixed parameters that depend on the u_i's. The polynomials are listed in the Maple calculation below. To see how one mechanically proves a theorem once the equivalent polynomial system is obtained some additional definitions are needed.

The following definition is an oversimplification of what is needed to mechanically prove theorems, but it will introduce the general concept and show us where to exert our efforts to effectively prove theorems mechanically.

Definition 1. *We say that the conclusion g follows strictly from the hypotheses* $h_1, ..., h_n$ *if* $g \in I(V) \subset \mathbb{R}\,[u_1, ..., u_m, x_1, ..., x_n]$, *where* $V{=}V(\langle h_1, ..., h_n\rangle)$.

The following proposition is our test to determine if g follows strictly from a set of hypotheses.

Proposition 2. *If* $g \in \sqrt{\langle h_1, ..., h_n\rangle}$, *then g follows strictly from* $h_1, ..., h_n$.

Proof. The proof is quite simple. If g $\in \sqrt{\langle h_1, ..., h_n\rangle}$ then $g^s \in \langle h_1, ..., h_n\rangle$ for some s. Hence g vanishes whenever $\langle h_1, ..., h_n\rangle$ vanish. □

We observe the first drawback of this approach. If our coefficient field is not algebraically closed, then the converse of Proposition 2 will not hold. However, this approach is useful because of its algorithmic flavor; we can use the radical membership algorithm to determine if $g \in \sqrt{\langle h_1, ..., h_n\rangle}$.

Now I will use Maple to show that the median theorem polynomials satisfy these conditions.

```
> with(Groebner):
> h1 := 2*x3-u3:
> h2 := 2*x4-u1:
> h3 := 2*x5-u2:
> h4 := 2*x2-u2:
> h5 := (x1-u1)-(u3-x1):
> h6 := x7*(x6-x1)-x6*(x7-x2):
> h7 := x7*(u1-x3)-u2*(x6-x3):
> g := x7*(x6-x4)-(x7-x5)*(x6-u3):
> J1 := [h1,h2,h3,h4,h5,h6,h7,1-g*y]:
> GBJ1 := gbasis(J1, plex(u1,u2,u3,x1,x2,x3,x4,x5,x6,x7,y));

                              GBJ1 := [1]
```

First note that the ordering did not have to be pure lexicographic, any ordering will suffice. However, these calculations can become very costly. Furthermore, there is a major drawback to this method. Since the u_i's are arbitrary choices one could choose them in such a way that the theorem becomes degenerate. These degenerate cases could cause the test of the theorem, i.e. is $g \in \sqrt{\langle h_1, ..., h_n \rangle}$, to fail. Although we now have two huge hurtles, efficiency and degeneracies, to overcome we are not at an impasse.

Wu Wen-Tsun developed his own approach for proving theorems in 1977. While Wu was working he realized that much of the algebraic ground work for his method was already provided by J.F. Ritt. Wu then implemented much of Ritt's algebra to create an algorithm that is more efficient and covers the degeneracies better than the Gröebner basis approach. I will now discuss the algorithms that I wrote in Singular that use Wu's basic method. These algorithms solve the efficiency portion of the problem.

2. Pseudo Division

The main tool used in Wu's algorithm is pseudo division. Although the name may indicate otherwise, pseudo division is actually just a slight variation on regular polynomial division in one variable followed by a clearing of denominators. This is why Wu's method is computationally superior to the Gröebner basis method; all that is needed to perform Wu's basic method is a series of divisions in one variable, which is easily handled by a computer. By Wu's basic method I refer to the algorithm that proves theorems but does not handle degeneracies (this is the class of theorems for which I wrote code). More complicated theorems will yield significantly more complicated Gröebner bases, but more complicated theorems do not affect Wu's basic method because the basic method only involves polynomial division.

Now I will explain my Singular pseudo division algorithm. Consider $f, g \in k[u_1, ..., u_m, x_1, ..., x_n]$. The notation used in my Singular code for pseudo dividing f by g with respect to the variable x_i is $\text{pdiv}(g, f, x_i)$. When working in Singular only one ring can be active at a time. The first steps in the routine pdiv create a new ring in which all the variables except x_i become parameters. Essentially the following map Φ is created:

$$\Phi : k[u_1, ..., u_m, x_1, ..., x_n] \hookrightarrow k(u_1, ...u_m, x_1, ..., \hat{x_i}, ..., x_n)[x_i]$$
$$\Phi(u_i) = u_i, \Phi(x_i) = x_i$$

Here is the first part of the Singular code:

```
proc pdiv (poly g,poly f,int z) {

\\ First I need to initialize integers with the same names as the
\\ variables in my original ring so that I can create a new ring
```

```
\\ with the ui's and xj's (with j!=z) as parameters.

 int j;
 for(j=1; j<=7; j=j+1)
 {
   if (j!=z) {int x(j);}
 }

 int u(1..7);

\\ I need several conditional statements to handle various choices
\\ of z. Note that this method is not totally general. The
\\ code needs to be slightly changed depending on the number
\\ of variables in the original ring.


if (z==1) {ring S=(0,u(1..7),x(2..7)), var(z), Dp;}

if (z==7) {ring S=(0,u(1..7),x(1..6)), var(z), Dp;}

if (z!=1 && z!=7) {ring
S=(0,u(1..7),x(1..z-1),x(z+1..7)),var(z),Dp;}

\\ imap is the map \Phi described earlier.

poly pf= imap(r,f);

poly pg= imap(r,g);

\\ The polynomials in the new ring are then sent to my division
\\ (DIV) procedure. DIVcount merely counts the number of times the
\\ while loop executes in DIV.

poly prem = DIV(pg,pf); int i = DIVcount(pg,pf);

\\ This next step clears the denominator of the remainder so
\\ that the remainder can be passed back to the priginal ring.

prem= prem*(leadcoef(pg))^i;

\\ Now I just set the original ring, imap the remainder, and return
\\ the remainder.
```

```
setring r;

poly rem =imap(S,prem); return(rem)

}
```

To see why the clearing of the denominator is necessary and to observe why the leading coefficient of g is the only term that could appear in the denominator of the remainder observe my DIV procedure.

```
proc DIV (poly g, poly f) {

\\ This procedure divides f by g.

 poly q = 0;
 poly r = f;

\\ This next condition is necessary because if g is a unit, and hence has
\\ deg=0, then g will divide f and the remainder will be 0. This will
\\ lead to divisions by zero when successive pseudo divisions are done.

 if (deg(g)==0) {return (g)}

\\ Upon each pass through the while loop the remainder r will pick up a
\\ factor of lead(g) in the denominator. When I pass r back to the
\\ original ring I need to first multiply r by lead(g) raised to the
\\ number of times the while loop was executed. DIVcount returns the
\\ number of times the while loop was executed.

 else
  {
   while ( r != 0 && deg(g) <= deg(r) )
   {
    q = q + (lead(r)/lead(g));
    r = r - (lead(r)/lead(g))*g;

   }
   return(r)
  }
}
```

The reason that clearing of denominators is necessary is that the division takes place in a polynomial ring with a coefficient fraction field $k(u_1, ...u_m, x_1, ..., \hat{x_i}, ..., x_n)$ and the remainder is passed back to the original ring with the coefficient field of just k. Thus to make sense, the remainder cannot have coefficients that exist outside the base field k.

3. Triangulation of polynomial Systems

Now that I have a system of polynomials from a theorem I want to obtain the following triangular form:

$$\begin{aligned} f_1 &= f_1(u_1, ..., u_n, x_1), \\ f_2 &= f_2(u_1, ..., u_n, x_1, x_2), \\ &\vdots \\ f_n &= f_n(u_1, ..., u_n, x_1, ..., x_n) \end{aligned}$$

This form can be obtained by the following general algorithm:
1. Input an ideal I = $\langle h_1, ...h_n \rangle$.
2. Take h_i where the degree of h_i in x_n is 1. Then perform pdiv(h_j, h_i, x_n) for all j$\neq$i. Repeat this for $x_{n-1}, ..., x_1$.
3. If there is a variable x_i such that there is no polynomial h_j such that the deg(h_j, x_i) = 1 then choose any h_j and $h_j^{'}$ such that deg(h_j, x_i), deg$(h_j^{'}, x_i) > 0$.
4. Let prem = pdiv$(h_j, h_j^{'}, x_i)$ and then do pdiv$(h_j^{'}, \text{prem}, x_i)$. Within finitely many steps you will obtain a pseudo remainder that will satisfy the condition in Step 2 in x_i. Return to Step 2.

My triangulation procedure is not this general. This is due to the fact that most geometric theorems are in near triangular form if the polynomial equivalents are chosen cleverly enough. Thus my triangulation procedure merely loops through the input ideal n times and performs pseudo divisions that are sufficient to triangulate most theorems (I have not yet come across a theorem that my procedure would not triangulate, but one should note that the procedure is not totally general). Here is the Singular code:

```
proc triang (ideal w) {

\\ First I'll initialize 2 integers for loop counters.

 int a,b;


\\ The following nested loop determines which pseudo divisions take place.
\\ The outer loop starts at 1 and the inner loop performs n-1 pseudo
\\ divisions in order to eliminate x(2),...,x(n) from the first
\\ polynomial. Then the outer loop is activated again and it commands the
\\ inner loop to perform n-2 pseudo divisions with the second polynomial.
\\ This continues and when both loops are completed the triangular system
\\ is returned.
```

```
 for(a=1; a<size(w); a=a+1)
 {
  for(b=a+1; b<=size(w); b=b+1)
   {
    w[a]= pdiv(w[a],w[b],(b));
   }
 }

return(w) }
```

To see why this system is useful consider the following proposition.

Proposition 3. *Suppose that $f_1 = \cdots = f_n = 0$ are the triangular equations obtained from $h_1 = \cdots = h_n = 0$ by the algorithm described above. Then $V(\langle h_1, ..., h_n\rangle) \subset V(\langle f_1, ..., f_n\rangle)$.*

Proof. By the pseudo division algorithm, all the pseudo remainders can be written as $r = lead(g)^i h_2 - qh_1$ for h_1 pseudo divding h_2, where i is the number of times the while loop is executed and $q \in k[u_1, ..., u_m, x_1, ..., x_n]$. This implies $r \in \langle h_1, h_2\rangle \subset \langle h_1, ..., h_n\rangle$. Hence, $\langle f_1, ..., f_n\rangle \subset \langle h_1, ..., h_n\rangle$ and taking the variety of the two ideals proves the theorem since the variety operation is inclusion reversing.

□

4. Successive Pseudo Division

Now we will see why the triangular form and pseudo division will help us prove theorems. First, I need to define the notion of successive pseudo division. Let $f_1, ..., f_n$ be the triangular system of a set of hypotheses and let g be the conclusion. Successive pseudo division is defined to be the following computations:

$$\begin{aligned} R_{n-1} &= pdiv(f, f_n, x_n) \\ R_{n-2} &= pdiv(R_{n-1}, f_{n-1}, x_{n-1}) \\ &\vdots \\ R_0 &= pdiv(R_1, f_1, x_1) \end{aligned}$$

Now I can state the theorem that is at the heart of Wu's method.

Theorem 4. *Consider a set H of hypotheses $h_1, ..., h_n$ and a conclusion g. If $R_0 = 0$ when successive pseudo division is computed with g and $f_1, ..., f_n$, the triangular form of H, then g follows from H.*

Proof. Just consider the following substitution method for writing R_0:

$$\begin{aligned} R_{n-1} &= pdiv(f, f_n, x_n) = d_n^{s_n} g - q_n f_n \\ R_{n-2} &= d_{n-1}^{s_n - 1}(d_n^{s_n} g - q_n f_n) - q_{n-1} f_{n-1} \\ R_{n-3} &= d_{n-2}^{s_n - 2}(d_{n-1}^{s_n - 1} d_n^{s_n} g - d_{n-1}^{s_n - 1} q_n f_n - q_{n-1} f_{n-1}) - q_{n-2} f_{n-2} \\ &\vdots \\ R_0 &= d_1^{s_1} d_2^{s_2} \cdots d_n^{s_n} g - (A_1 f_1 + \ldots + A_n f_n) \end{aligned}$$

Note that $A_1, ..., A_n \in \mathbb{R}\,[u_1, ..., u_m, x_1, ..., x_n]$ and that d_i is the leading coefficients of f_i in x_i. Thus if $R_0 = 0$ then when the f_i's vanish either g vanishes on one of the d_i's vanishes. However, a d_i vanishing corresponds to a degenerate geometric configuration, so we can conclude that g follows from H when $R_0 = 0$. □

The Singular procedure for successive pseudo division is the most straightforward. The procedure just takes an ideal in the form $\langle f_1, ..., f_n, g\rangle$ and loops through the ideal backwards and performs the divisions. Here is the code:

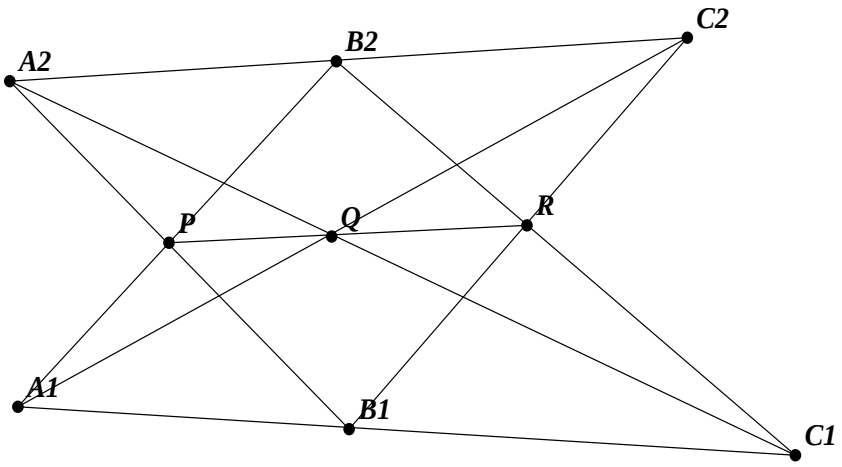
```
proc spdiv (ideal w) {

\\ In order for this procedure to work corectly the ideal
\\ w must be entered in the following way:f1,...,fn,g.


 int a,c;
 int b= size(w);
 ideal R;
 size(R)=b-1;

 R[b-1]= pdiv(w[b],w[b-1],(b-1));

 for(a=b-2; a>1; a=a-1)
 {
  R[a]= pdiv(R[a+1],w[a],a);
 }

 return(R)
}
```

I will now illustrate this on an example with my Singular procedures.

5. An Example of Wu's Method

I will prove Pappus's theorem using the algorithms I have described above. Here is the statement of the theorem:

Theorem 5. *Let the vertices A_1, B_1, C_1, A_2, B_2, C_2 of a hexagon lie on 2 distinct lines. Then the points of intersection of opposite sides B_2C_1 and B_1C_2, C_2A_1 and C_1A_2, A_2B_1 and A_1B_2, are collinear.*



The first objective one must complete before obtaining an equivalent system of equations for a theorem is to determine which parameters in the geometrical construction can be freely chosen and which parameters will then be fixed. If the origin is placed at A_1 then B_1 and C_1 can be freely chosen on the x-axis. Then A_2 and B_2 could be freely chosen above the x-axis. This would leave either the x or the y value of C_2 free, since C_2 is collinear with A_2 and B_2. This will leave the three points of intersection completely determined.

$$\begin{aligned} A_1 : (0,0), B_1 : (u_1,0), C_1 : (u_2,0), \\ A_2 : (u_3,u_4), B_2 : (u_5,u_6), C_2 : (u_7,x_1), \\ P : (x_2,x_3), Q : (x_4,x_5), R : (x_6,x_7) \end{aligned}$$

Now I will formulate 7 equations that are equivalent to the geometric assumptions of the theorem. If this system is consistent then it will imply that PQR are collinear, which is the same as saying the equation that represents PQR being collinear is satisfied.

$$\begin{aligned}
A_2B_2C_2 \text{ are collinear} &: (u_6-u_4)(u_7-u_5)-(u_5-u_3)(x_1-u_6)=0\\
A_2PB_1 \text{ are collinear} &: (x_3-u_4)(x_2-u_1)-x_3(x_2-u_3)=0\\
A_1PB_2 \text{ are collinear} &: x_3(x_2-u_5)-x_2(x_3-u_6)=0\\
A_2QC_1 \text{ are collinear} &: (x_5-u_4)(x_4-u_2)-x_5(x_4-u_3)=0\\
A_1QC_2 \text{ are collinear} &: x_5(x_4-u_7)-x_4(x_5-x_1)=0\\
B_2RC_1 \text{ are collinear} &: (x_7-u_6)(x_6-u_2)-x_7(x_6-u_5)=0\\
B_1RC_2 \text{ are collinear} &: x_7(x_6-u_7)-(x_6-u_1)(x_7-x_1)=0
\end{aligned}$$

The following relation is our conclusion.

$$PQR \text{ are collinear} : (x_3-x_5)(x_6-x_4)-(x_2-x_4)(x_7-x_5)=0$$

Now here is exactly what needs to be entered into Singular to prove the theorem.

```
ring r=0, (x(1..7),u(1..7)), lp;

ideal i= (u(6)-u(4))*(u(7)-u(5)) - (x(1)-u(6))*(u(5)-u(3)),
         (x(3)-u(4))*(x(2)-u(1)) - x(3)*(x(2)-u(3)),
          x(3)*(x(2)-u(5)) - x(2)*(x(3)-u(6)),
         (x(5)-u(4))*(x(4)-u(2)) - x(5)*(x(4)-u(3)),
          x(5)*(x(4)-u(7)) - x(4)*(x(5)-x(1)),
         (x(7)-u(6))*(x(6)-u(2)) - x(7)*(x(6)-u(5)),
          x(7)*(x(6)-u(7)) - (x(6)-u(1))*(x(7)-x(1));


triang(i);

_[1]=x(1)*u(3)-x(1)*u(5)-u(3)*u(6)+u(4)*u(5)-u(4)*u(7)+u(6)*u(7)
_[2]=-x(2)*u(1)*u(6)+x(2)*u(3)*u(6)-x(2)*u(4)*u(5)+u(1)*u(4)*u(5)
_[3]=x(2)*u(6)-x(3)*u(5)
_[4]=-x(1)*x(4)*u(2)+x(1)*x(4)*u(3)-x(4)*u(4)*u(7)+u(2)*u(4)*u(7)
_[5]=x(1)*x(4)-x(5)*u(7)
_[6]=-x(1)*x(6)*u(2)+x(1)*x(6)*u(5)+x(1)*u(1)*u(2)-x(1)*u(1)*u(5)
     +x(6)*u(1)*u(6)-x(6)*u(6)*u(7)-u(1)*u(2)*u(6)+u(2)*u(6)*u(7)
_[7]=x(1)*x(6)-x(1)*u(1)+x(7)*u(1)-x(7)*u(7)


ideal j=triang(i);

ideal c= j+ (x(3)-x(5))*(x(6)-x(4)) - (x(2)-x(4))*(x(7)-x(5));

spdiv(c);
```

```
_[1]=0
_[2]=-x(1)*x(4)*u(2)+x(1)*x(4)*u(3)-x(4)*u(4)*u(7)+u(2)*u(4)*u(7)
_[3]=-x(1)*x(4)*u(2)+x(1)*x(4)*u(3)-x(4)*u(4)*u(7)+u(2)*u(4)*u(7)
_[4]=-x(1)*x(4)*u(2)+x(1)*x(4)*u(3)-x(4)*u(4)*u(7)+u(2)*u(4)*u(7)
_[5]=x(1)^2*x(4)^2*u(2)-x(1)^2*x(4)^2*u(5)-x(1)^2*x(4)*u(1)*u(2)+
     +x(1)^2*x(4)*u(1)*u(5)-x(1)*x(3)*x(4)*u(2)*u(7)+x(1)*x(3)*x(4)*u(5)
     *u(7)+x(1)*x(3)*u(1)*u(2)*u(7)-x(1)*x(3)*u(1)*u(5)*u(7)-x(1)*x(4)^2
     *u(1)*u(6)+x(1)*x(4)^2*u(6)*u(7)+x(1)*x(4)*u(1)*u(2)*u(6)-x(1)*x(4)
     *u(2)*u(6)*u(7)+x(3)*x(4)*u(1)*u(6)*u(7)-x(3)*x(4)*u(6)*u(7)^2-x(3)
     *u(1)*u(2)*u(6)*u(7)+x(3)*u(2)*u(6)*u(7)^2
_[6]=-x(1)*x(3)*x(4)*u(2)+x(1)*x(3)*x(4)*u(5)+x(1)*x(3)*u(1)*u(2)-
     -x(1)*x(3)*u(1)*u(5)+x(1)*x(4)*x(5)*u(2)-x(1)*x(4)*x(5)*u(5)-x(1)*x(5)
     *u(1)*u(2)+x(1)*x(5)*u(1)*u(5)+x(3)*x(4)*u(1)*u(6)-x(3)*x(4)*u(6)*u(7)
     -x(3)*u(1)*u(2)*u(6)+x(3)*u(2)*u(6)*u(7)-x(4)*x(5)*u(1)*u(6)+x(4)*x(5)
     *u(6)*u(7)+x(5)*u(1)*u(2)*u(6)-x(5)*u(2)*u(6)*u(7)
_[7]=-x(3)*x(4)+x(3)*x(6)+x(4)*x(5)-x(5)*x(6)
```

Recall that when I proved Theorem 4 I said that the d_i's corresponded to degenerate configurations. Now that we have an example I will show why these configurations need to be thrown away. Consider the triangular form for the hypotheses of Pappus's Theorem. The leading coefficient of f_1 is $(u_3 - u_5)$. Thus the x-component of A_2 and B_2 would have to be equal. But then the x-component of C_2 is a free choice, so the three points are not collinear for an infinite number of choices for u_7. The configurations that do not satisfy my original hypotheses are not a concern so we disregard where they vanish.