Primitive PRS Algorithm

We describe the algorithm and give Maple code for the primitive fraction-free algorithm that we used to compute a gcd in $L[x]$ for comparison with SparseModGcd algorithm. We think of computing in L as computing modulo the triangular set $M = \{m_1, \ldots, m_r\}$. To avoid fractions, we first set $f_1 := \check{f}_1$, $f_2 := \check{f}_2$ and $M := \{\check{m}_1, \ldots, \check{m}_r\}$. Now suppose we apply the Euclidean algorithm to compute the gcd of f_1 and f_2 modulo M. We would divide f_1 by f_2. In the ordinary division algorithm we would invert the leading coefficient u of the divisor f_2, an algebraic function. The coefficients of the inverse of u would have fractions in $F = \mathbb{Q}(t_1, \ldots, t_k)$.

To avoid fractions here we compute instead v, a quasi-inverse of u, an element of $D[z_1, \ldots, z_r]$ satisfying $vu = c$ for some constant $c \in D = \mathbb{Z}[t_1, \ldots, t_k]$. Now we divide f_1 by vf_2 using pseudo division (mod M). And we make the pseudo remainder "primitive", i.e., we compute and cancel out any common factor in D from the coefficients.

To compute the quasi-inverse v we first apply the (extended) Euclidean algorithm to $\check{m}_r$ and u viewing them as elements of $K[z_r]$ where $K = F[z_1, ..., z_{r-1}]/\langle m_1, ..., m_{r-1}\rangle$. Again, we want to avoid fractions so we use pseudo-division. We perform pseudo-division in $D[z_1, \ldots, z_{r-1}][z_r]$. We obtain s, t, c satisfying

$$sm_r + tu = c \quad \text{where} \quad c \in D[z_1, \ldots, z_{r-1}].$$

Here c does not involve z_r but may involve $z_1, \ldots, z_{r-1}$. Next we recursively compute a quasi-inverse w of c satisfying $wc \in D$ and hence $v = wt$ is a quasi-inverse of u and we reduce wt modulo M using pseudo-division. Here is the algorithm in Maple code.

```
macro( `mod` = MOD );
MOD := proc(f,M,Z) local r,i;
  r := expand(f);
  for i to nops(M) do r := prem(r,M[i],Z[i]) od;
  r;
end:

# This uses the reduced PRS
QuasiInv := proc(x,M,Z)
local u,r0,r1,r2,t0,t1,t2,pq,mu,i,z,beta;
  if M=[] then return 1 fi;
  u := primpart(x,Z);
  r0,r1,t0,t1,beta := M[1],u,0,1,1;
  z := Z[1]; # main variable
  while degree(r1,z)>0 do
    r2 := prem(r0,r1,z,'mu','pq');
    divide(r2,beta,'r2');
    divide(mu*t0 - pq*t1,beta,'t2');
    r0,r1,t0,t1,beta := r1,r2,t1,t2,mu;
  od;
  if r1=0 then error "inverse does not exist" fi;
  if nops(M)>1 then
    r1 := r1 mod (M,Z);
    t1 := t1 mod (M,Z);
    i := QuasiInv(r1,M[2..-1],Z[2..-1]);
    t1 := i*t1 mod (M,Z):
  fi;
  primpart(t1,Z);
end:

PrimitivePRS := proc(f1,f2,x,M,Z)
local xZ,i,r0,r1,r2;
  xZ := [x,op(Z)];
  r0 := primpart(f1,xZ);
  r1 := primpart(f2,xZ);
  while r1 <> 0 do
    i := QuasiInv(lcoeff(r1,x),M,Z);
    r1 := primpart(i*r1 mod (M,Z), xZ);
    r2 := prem(r0,r1,x) mod (M,Z);
    r2 := primpart(r2,xZ);
    r0,r1 := r1,r2;
  od;
  r0;
end:
```