Lógica Computacional 117366
Descrição do Projeto
Formalização de Algoritmos para computação do MDC

17 de dezembro de 2010

Prof. Mauricio Ayala-Rincón

O aluno Thiago Mendonça Ferreira Ramos (`thiagomendoncaferreiraramos@yahoo.com.br`) dará suporte aos alunos no desenvolvimento do projeto. Laboratórios do LINFO (6) tem instalado o software necessário (PVS com as livrarias PVS da NASA).

1 Introdução

O Máximo Divisor Comum (MDC) entre dois números inteiros, diferentes de zero, é uma noção básica em computação e matemática, definida como

$$\forall i, j \in \mathbb{Z}, i \neq 0 \vee j \neq 0, \quad MDC(i,j) := \text{máximo}\{k \mid k \mid i \wedge k \mid j\}$$

onde "$m \mid l$" denota que m divide a l.

Algoritmos para computar MDC estão disponíveis na maioria das linguagens de programação assim como em ambientes computacionais utilizados para computação algébrica e numérica.

Diversas implementações para computar MDC são inspiradas no primeiro algoritmo conhecido, o algortimo de Euclides, mas outras abordagens são possíveis.

A correção e completude de qualquer método computacional, $\mathcal{A}$, para computar MCD pode ser expressa da seguinte maneira:
Correção: para todo par de inteiros i e j, tais que algum deles é diferente de zero, $\mathcal{A}$ computa para as entradas i e j o valor $\mathcal{A}(i,j)$, tal que $\mathcal{A}(i,j) \mid i$ e $\mathcal{A}(i,j) \mid j$ e para qualquer $k \in \mathbb{Z}$, tal que $k \mid i$ e $k \mid j$, $k \leq \mathcal{A}(i,j)$.

O objetivo do projeto da disciplina é formalizar a correção de diferentes abordagens para computar MDC no assistente de demonstração PVS.

2 Descrição do Projeto

Com base na *teoria* PVS `gcd` disponível na página da disciplina, especificada na linguagem do assistente de demonstração PVS (`pvs.csl.sri.com`) executável em plataformas Unix/Linux os alunos deverão formalizar a corrção de diferentes abordagens para computar MDC expressas nas especificações das funções `gcd_SW`, `gcd_DT` e `gcd_jp`. A base para esse trabalho é a formalização da função `gcd`, que corretamente computa o MDC entre dois inteiros não simultaneamente nulos, presente na *teoria* `gcd`:

```
gcd(n, m) : RECURSIVE nat =
                  IF abs(n) = abs(m) THEN abs(n)
                  ELSE IF (n = 0 OR m = 0) THEN abs(n+m)
                       ELSE IF (abs(n) > abs(m)) THEN
                                       gcd(abs(n)-abs(m), abs(m))
                                  ELSE gcd(abs(m)-abs(n),abs(n))
                            ENDIF
```

```
                    ENDIF
                ENDIF
MEASURE abs(n)+abs(m)
```

A formalização da correção desta função consiste de:

- Prova de que a função está bem definida; i.e., para cada par de argumentos inteiros `n` e `m`, `gcd` retorna um valor `natural`. O que está expresso na demonstração de *Type Correctness Conditions - TCCs* geradas pelo sistema.
- Prova de que a função é logicamente correta; i.e., prova de que o `gcd` computado fornece um divisor dos argumentos e da sua maximalidade, o que está expresso no corolário:

```
gcd_is_correct : COROLLARY
            (m /= 0 OR n /=0) => divides(gcd(m,n),m) &
                                 divides(gcd(m,n),n) &
                                 FORALL (k) : (divides(k,m) &
                                               divides(k,n) =>
                                               k <= gcd(m,n))
```

Mais especificamente, os alunos deverão formalizar a correção das funções `gcd_SW`, `gcd_DT`, especificadas para computar o MDC entre inteiros, e `gcd_jp`, especificada para computar o MDC entre inteiros positivos, embaixo

```
gcd_SW(n, m) : RECURSIVE nat =
                IF abs(n) = abs(m) THEN abs(n)
                ELSE IF (n /= 0 AND m /= 0) THEN
                        IF (abs(n) > abs(m)) THEN
                             gcd_SW(abs(n)-abs(m), m)
                        ELSE gcd_SW(abs(m)-abs(n), n)
                        ENDIF
                     ELSE  abs(n+m)
                     ENDIF
                ENDIF
 MEASURE abs(n)+abs(m)

gcd_DT(n, (m | m /= 0 OR n/= 0) )  : RECURSIVE nat =
                IF abs(n) = abs(m) THEN abs(n)
                ELSE IF (n = 0 OR m = 0) THEN abs(n+m)
                     ELSE IF (abs(n) > abs(m)) THEN
                                    gcd_DT(abs(n)-abs(m), abs(m))
                                ELSE gcd_DT(abs(m)-abs(n),abs(n))
                          ENDIF
                     ENDIF
                ENDIF
 MEASURE abs(n)+abs(m)

 gcd_jp_aux(i, j,(k | (k >= 2 AND k <= i + 1 AND k <= j + 1)) ) : RECURSIVE nat =
                IF (i = 1 OR j = 1) THEN 1
```

```
                ELSE
                 IF (k > i OR k > j) THEN 1
                 ELSE
                  IF (divides(k, i) AND divides(k, j)) THEN
                    k * gcd_jp_aux(ndiv(i,k), ndiv(j,k), 2)
                  ELSE gcd_jp_aux(i, j, k+1) ENDIF
                 ENDIF
                ENDIF
 MEASURE i + j - k
```

demonstrando mecanicamente, as seguintes conjecturas:

```
%%% Question 01  %%%
gcd_SW_same_gcd : LEMMA FORALL (m,n) :
    gcd_SW(m,n) = gcd(m,n)

gcd_SW_is_correct : CONJECTURE
            (m /= 0 OR n /=0) => divides(gcd_SW(m,n),m) &
                                 divides(gcd_SW(m,n),n) &
                                 FORALL (k) : (divides(k,m) &
                                              divides(k,n) =>
                                              k <= gcd_SW(m,n))
%%% Question 02  %%%
gcd_DT_same_gcd : CONJECTURE FORALL (m,n) :
     (m /= 0 OR n /=0) => gcd_DT(m,n) = gcd(m,n)

gcd_DT_is_correct : CONJECTURE
            (m /= 0 OR n /=0) => divides(gcd_DT(m,n),m) &
                                 divides(gcd_DT(m,n),n) &
                                 FORALL (k) : (divides(k,m) &
                                              divides(k,n) =>
                                              k <= gcd_DT(m,n))
%%% Question 03  %%%
gcd_jp_same_gcd : CONJECTURE
          gcd_jp(i,j) = gcd(i,j)

gcd_jp_is_correct : CONJECTURE
         divides(gcd_jp(i,j),i) &
         divides(gcd_jp(i,j),j) &
         FORALL (k) : (divides(k,i) & divides(k,j) =>
                                        k <= gcd_jp(i,j))
```

Observa-se que as duas primeiras formalizações são consequências simples das conjecturas de coincidência entre estas funções assim especificadas (`gcd_SW` e `gcd_DT`) e a função `gcd`, mas o sucesso na última formalização dependerá de descobrir e provar alguns resultados algébricos a ser determinados durante a pesquisa para desenvolvimento desta formalização.

3 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **quatro** membros. O projeto será dividido em duas etapas como segue:

- A primeira etapa do projeto e a de Verificação das Formalizações. Os grupos deverão ter prontas as suas formalizações na linguagem do assistente de demonstração PVS o dia **31.01.2011**. Na semana de 31.01-02.02.2011 realizar-se-á a verificação do trabalho para a qual os grupos deverão, em acordo com o monitor e professor, determinar um horário (de uma hora) no qual todos membros do grupo deverão comparecer.
  Avaliação (peso 6.0):
  - Um dos membros, selecionado por sorteio, explicará os detalhes da formalização em máximo 20 minutos.
  - Os quatro membros do grupo poderão complementar a explicação inicial em máximo 10 minutos.
  - A formalização será testada nos seguintes 30 minutos.
- A segunda etapa do projeto consiste da apresentação dos resultados finais e conclusões do estudo do problema.
  Avaliação (peso 4.0): Cada grupo de trabalho devera entregar um Relatório Final inédito, editado em Latex, limitado a oito páginas (12 pts, A4, espaçamento simples) do projeto até o dia **03.02.2011** com o seguinte conteúdo:
  - Introdução e contextualização do problema.
  - Explicação da soluções.
  - Especificação do problema e explicação do método de solução.
  - Descrição da formalização.
  - Conclusões.
  - Referências.