Matemática Discreta - 2011.2

```
Horários do curso em 2011.2:
  4ªs 14-16, sala 5A ou 6A,
  5ªs  9-11, sala 5A ou 6A.
  Veja: http://angg.twu.net/2011.2.html
                  (find-TH "2011.2")
Página do curso de 2011.1:
  http://angg.twu.net/2011.1-MD.html
            (find-TH "2011.1-MD")
Arquivo com todas as folhas manuscritas do curso de 2011.1:
  http://angg.twu.net/MD/MD-2011.1-tudo.pdf
  http://angg.twu.net/MD/MD-2011.1-tudo.djvu


Vamos usar o livro do Scheinerman,
alguns capítulos do livro da Judith Gersting,
e o primeiro capítulo do Hopcroft/Ullman/Motwani.


Versão para impressão:
  http://angg.twu.net/MD/MD-2011.2.pdf
(Pode estar desatualizada!)
```

A monitora é a Gabriela Ávila <gabrielaa@id.uff.br>. Os horários de atendimento dela são nas 2ªs das 9:00 às 13:00 e nas 4ªs das 14:00 às 16:00.

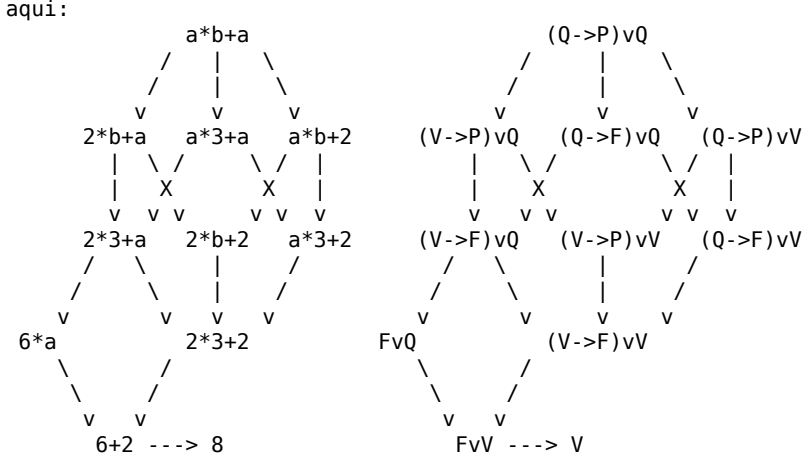
A Ana Isabel está dando o curso dela de modo bem diferente, mas vale muito a pena olhar o material do curso dela (no Moodle):

```
http://softwarelivre.uff.br/esl/
http://softwarelivre.uff.br/esl/course/view.php?id=51  MD Bel 2011.2
http://softwarelivre.uff.br/esl/mod/forum/discuss.php?d=209 Gabriela
```

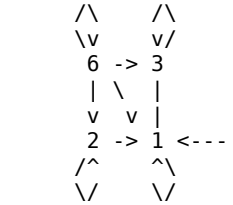
Plano de aulas / resumo do que já aconteceu:

```
1ª aula (10/ago): aula cancelada por ainda não ter sala alocada.
2ª aula (11/ago): dei uma aula para a minha turma e uma para a turma
  da Ana Isabel (que fraturou o tornozelo e está sem poder andar), e
  as duas foram só um pouco diferentes. Discutimos tipos de objetos -
  números, strings, conjuntos, valores de verdade... - e avisei que
  confundir tipos é "erro conceitual gravíssimo" - veja estas regras
  sobre como escrever questões em provas:
    http://angg.twu.net/LATEX/2011-1-GA-regras.pdf
  (depois vou fazer uma versão delas específica para MD).
  Consideramos o exemplo de um programa de computador que é uma
  calculadora na qual o "display" dela funciona também como uma barra
  de texto na qual podemos digitar expressões, e quando apertamos o
  botão "CALC" ela calcula o valor da expressão. Por exemplo,
    "2+3"        -->  "5"
    "2*3+4*5"    -->  "26"
    "26"         -->  "26"
    "2<3"        -->  "V"
    "4>=5"       -->  "F"
    "1/0"        -->  "ERRO"
    "Alexandre"  -->  "ERRO"
    "ERRO"       -->  "ERRO"
  Às vezes vamos considerar que essa calculadora suporta váriaveis,
  como em linguagens de programação... então quando a=2 temos
    "a"          --> "2"
```

```
  "a+3"        --> "5"
```
e quando a=10 temos:
```
  "a"          --> "10"
  "a+3"        --> "13"
```
Aí nós deixamos de lado temporariamente a idéia dessa calculadora que mostra direto o resultado final e começamos a ver como calcular expressões passo a passo. Vimos como funcionam o "∧", o "∨" e o "->" (definimos eles por tabelas por enquanto), e, supondo que a=2, b=3, P=F, Q=V, tentamos calcular "a*b+a" e "(Q->P)vQ". Mostrei que tipo de diagramas estávamos procurando, avisei que os diagramas pra estas expressões teriam 11 nós e 16 setas, e os alunos chegaram a isto aqui:



Discutimos também vários tipos de setas erradas - por exemplo:
```
  2*3+4 --> 2*7
```
Não queremos incluir setas erradas nos nossos diagramas.
Pra minha turma falei do grafo direcionado do "é divisível por" em {1,...,6}:



e lembrei a todos que ele é representado "em matematiquês" como o par ordenado
```
  ( {1,2,3,4,5},
    {                                       (6,6)
                                      (5,5),
                             (4,4),
                    (3,3),                  (6,3),
            (2,2),          (4,2),          (6,2),
      (1,1), (2,1), (3,1), (4,1), (5,1), (6,1) } )
```
e que o grafo direcionado do cálculo passo a passo do "a*b+a" é algo parecido, mas com strings:
```
  ( {"a*b+a", "2*b+a", ..., "8"},
    { ("a*b+a", "2*b+a"), ..., ("6+2", "8") } )
```
Pra turma da Bel fizemos a tabela do x>3->x^2>3 para alguns valores de x,

x	x>3	x^2	x^2>3	x>3->x^2>3
0	F	0	F	V
1	F	1	F	V
2	F	4	V	V
3	F	9	V	V
4	V	16	V	V
5	V	25	V	V

e perguntei: será que x>3->x^2>3 é sempre verdade? E disse que na aula seguinte nós veríamos expressões que formalizam a idéia de "x>3->x^2>3 é sempre verdade" (e que os detalhes seriam complicados e que num primeiro momento as pessoas arrancariam os cabelos).

3ª aula (17/ago): cancelada.
4ª aula (18/ago): Voltando à idéia da calculadora, se ela usa a sintaxe do C, em que o "=" quer dizer atribuição e comparação é "==", então:

```
  "b = (a = 22) + 3"    -->   25
       \------/
          22
       \----------/
            25
    \--------------/
          25
```

e depois disto as variáveis "a" e "b" passam a ter os valores 22 e 33 respectivamente, e:

```
  "a"      -->  "22"
  "b"      -->  "25"
  "a+b"    -->  "47"
```

em MD nós vamos SEMPRE usar ":=" para atribuição, mas essas atribuições vão ser TEMPORÁRIAS (!!!!!!!!!!!!!!!)... Em Matemática, ao contrário de em programação as variáveis nunca mudam de valor - vamos entender isto aos poucos.
Se a calculadora suporta atribuição com ":=", então:

```
  "x := 2"    -->  "2"
  "x^2 < 10"  -->  "V"
  "x := 3"    -->  "3"
  "x^2 < 10"  -->  "V"
  "x := 4"    -->  "4"
  "x^2 < 10"  -->  "F"
```

A calculadora que a Gabriela (monitora) está implementando vai saber calcular coisas como:

```
  "∀x∈{2,3,4}.x^2<10"   -->  "F"
```

Mas precisamos entender como ela (a calculadora) faz isto...
Como podemos calcular expressões com "∀" e "∃" passo a passo?
A sintaxe das expressões com "∀" é SEMPRE assim:

```
  ∀ (variável) ∈ (conjunto) . (expressão)
```

então, por exemplo,

```
  "∀x∈{2,3,4} x^2<10"   -->  "ERRO"
```

porque falta o "." (!!!!!!!!!!) e

```
  "∀2∈{2,3,4}.x^2<10"   -->  "ERRO"
```

porque 2 não é uma variável!!!!!!!!!! Em Matemática Discreta, como em calculadoras e em linguagens de programação, QUALQUER erro de sintaxe TEM que ser tratado como erro - se a gente começa a ser tolerante com erros de sintaxe mais tarde a gente vai ser obrigado a decidir o significado de expressões ambíguas, e a gente vai se ferrar; o único modo da gente se proteger contra ambiguidades é a gente se restringir a expressões com a sintaxe correta.
  A regra pra gente fazer o "passo" que nos livra de um "∀" é a seguinte: "∀ (variável) ∈ (conjunto) . (expressão)" vira várias "cópias" da "expressão", uma para cada elemento do "conjunto", mas em cada "cópia" substituimos cada ocorrência da "variável" pelo elemento correspondente do conjunto, e depois conectamos as "cópias" com "∧"s...
Passei estes exercícios, e passamos muito tempo discutindo eles:

```
  (1) ∀a∈{-1,0,1}.a^2<a
  (2) ∀P∈{V,F}.(P->P)
  (3) ∀P∈{V,F}.∀Q∈{V,F}.PvQ
  (4) ∀a∈{0,1}.a<2v(∀a∈{1,2}.a>1)
```

Por exemplo:

```
     ∀a∈{-1,0,1}.a^2<a
                |
                v
```

```
   ((-1)^2<(-1))∧(0^2<0)∧(1^2<1)
                :
                v
              F∧V∧V
 e:
     ∀P∈{V,F}.∀Q.{V,F}.PvQ -----> ∀P∈{V,F}.((PvV)∧(FvF))
                |                          :
                v                          :
  (∀Q∈{V,F}.VvQ)∧(∀Q∈{V,F}.FvQ)            :
                :                          :
                v                          :
   ((VvV)∧(VvF))∧((FvV)∧(FvF)) <- - - - - - /
```

A diferença entre "∀" e "∃" é que no "∀" a gente usa "∧" e no "∃" a gente usa "v". Comentei que "∀" e "∃" são parecidos com somatórios - por exemplo,

$$\sum_{k=3}^{5} (k^2 + 1) = (3^2 + 1) + (4^2 + 1) + (5^2 + 1)$$

repare que em somatórios a gente liga as "cópias" com sinais de "+". Fiquei de pôr aqui na página do curso mais um monte de exercícios de fixação sobre quantificadores. Os primeiros exercícios são:

(5) ∀P∈{F,V}.Pv¬P
(6) ∀P∈{F,V}.∀Q∈{F,V}.P->(P∧Q)
(7) ∀P∈{F,V}.∀Q∈{F,V}.(P∧Q)->P
(8) ∀P∈{F,V}.∀Q∈{F,V}.(PvQ)->P
(9) ∀P∈{F,V}.∀Q∈{F,V}.P->(PvQ)
(10) ∃a∈{0,1,2,3}.2*a=3
(11) ∃a∈{0,1,2,3,4}.2*a=4
(12) ∃a∈{0,1,2,...,100}.2*a=100

nos próximos exercícios onde estiver escrito "a|b" (pronúncia: "a divide b") você vai ter que substituir o "a|b" por "∃k∈{0,...,b}.a*k=b" - ou seja, a _definição_ de "a|b" vai ser "∃k∈{0,...,b}.a*k=b" (vamos ver mais sobre definições em breve).

(13) ∃a∈{2,3,4}.a|5
(14) ∃a∈{2,...,5}.a|6

5ª aula (24/ago): pus os problemas (5) a (14) da aula passada no quadro, e mais estes:

(15) -2|4
(16) 2|-4

Está na hora de todo mundo aprender a fazer as próprias definições (pus no quadro avisos de que os alunos vão passar o curso todo de Ciência da Computação fazendo definições, que eles vão ter que começar a treinar logo, caveira × 10, etc), e a passei estes exercícios:

(A) Dê uma definição para "a é par" e teste-a.
(B) Dê uma definição para "a é ímpar" e teste-a.
(C) Isto vale? ∀a∈Z.(a é par)v(a é ímpar)
(D) Isto vale? 2/3 é ímpar
(E) Dê uma definição de ímpar na qual 2/3 seja ímpar (e teste-a).

Acabamos ficando só no (A). Apareceram várias definições erradas, além da certa, e vi que o mais interessante seria testarmos cada uma delas.

(A1) Def: a é par_1 se e só se ∀b∈Z.∀a∈R.a/2=b
(A2) Def: a é par_2 se e só se ∃k∈Z.(ak=b∧(a/2∈Z))
(A3) Def: a é par_3 se e só se ∃b∈Z.∀a∈R.a/2=b
(A4) Def: a é par_4 se e só se ∃x∈Z.x/2∈Z
(A5) Def: a é par_5 se e só se ∃a∈Z.a/2∈Z

Vimos que os quantificadores criam "variáveis locais", como em linguagens de programação....

6ª aula (25/ago):

Vimos duas sintaxes para definições:

Def: f(P,Q) = (P->(P∧Q))
Def: f(P,Q) se e só se P->(P∧Q)

E discutimos como usar definições pra calcular mais rapidamente os itens (5) a (16) da 4ª aula.

7ª aula (31/ago): na aula anterior só 5 pessoas vieram...
Voltamos aos problemas (15) e (16). Ninguém sabia o que deveria ser {-4,...,0}, então trabalhamos em cima deste problema (que chamamos de "4", por motivos que não importam aqui):

(4) Encontre uma definição para {a,...,b} e teste-a. Sua definição deve dar o resultado "óbvio" pelo menos nos itens 4a e 4b abaixo:
- (4a) Calcule {0, ..., 2}.
- (4b) Calcule {1, ..., 4}.
- (4c) Calcule {1.1, ..., 4.1}.
- (4d) Calcule {1.1, ..., 4}.
- (4e) Calcule {1.1, ..., 1.1}.
- (4f) Calcule {0, ..., -4}.

Apareceram as seguintes definições, e começamos a testá-las:

{a,...,b} se e só se ∃k∈{a,...,b}.ak=b∧b=a+n∧n∈R	(Gustavo)
{a,...,b} = ∀a<b.∃k∈Z.{{a,...,b} | a<k<b}	(Thiago)
{a,...,b} = (∃k∈{a,...,b}.4+k>b)	(Rodrigo)
{a,...,b} = (∃!n∈R.a+n_1+n_2+...+n_x=b)	(Natan)
{a,...,b} = ∀k∈{0,...,2}.k+1<=2∧n∈N	(Luis Fernando)
{a,...,b} se e só se ∃k∈R.a<k<b	(Luis Felipe)
{a,...,b} se e só se ∃k∈{0,...,2}.a<=2k	(Dângelo)

8ª aula (01/set): "Workshop de definições".
Vimos alguns critérios rápidos pra testar se definições - em particular as definições para {a,...,b} da aula passada - estão erradas:
1) Sintaxe
2) Tipos
3) Variáveis indefinidas
4) Recursão

e vimos que cada uma das definições propostas na aula passada tinha pelo menos um desses erros.
Pra casa: encontre duas definições diferentes para {a,...,b}, que dêem os resultados "esperados" quando a,b∈Z e a<=b, mas:
- {4.1, ..., 6.1}_I = {5, 6}
- {4.1, ..., 6.1}_R = {4.1, 5.1, 6.1}

9ª aula (07/set): feriado.
10ª aula (08/set):
Mostrei duas regras pra lidar com "_∈{_|_}": a "regra nova" e a "regra horrível". A "regra nova" nos permite lidar com expressões da forma "_ ∈ {gerador | filtro}". Exemplo:

```
  2.5 ∈ {k∈R | a+k<=b}
            |
            v
    2.5∈R ∧ a+2.5<=b
```

A "regra horrível" nos permite traduzir expressões da forma "{expressão | gerador}" para "{gerador | filtro}". Exemplo (com dois geradores):

```
  {a+b | a∈{10,20}, b∈{3,4}}
              |
              v
  {x∈R | ∃a∈{10,20}.∃b∈{3,4}.a+b=x}
```

Repare que "x∈R" é um gerador difícil de usar (porque gera valores demais) e "∃a∈{10,20}.∃b∈{3,4}.a+b=x" também pode ser um filtro difícil de usar... mas num caso como este, que eu passei como exercício na aula,

```
  14 ∈ {a+b | a∈{10,20}, b∈{3,4}}
```

podemos usar a "regra horrível", depois a "regra nova", depois expandir os dois "∃"s, e chegar ao resultado.
Os alunos passaram a aula quase toda propondo definições para {a,...,b}_R e testando-as (eu avisei que é muito melhor a gente escrever definições possivelmente erradas e tentar testá-las e

consertá-las do que não escrever nada). As definições que foram postas no quadro foram estas:

```
{a,...,b}_RNatan1    = {k∈Z | a<=k<=b | k∈[a,b]}
{a,...,b}_RNatan2    = {k∈Z | a<=k<=b ∧ k∈[a,b]}
{a,...,b}_RNatan3    = {k∈R | a<=k<=b ∧ k∈[a,b]}
{a,...,b}_RThaynara  = {k∈R | a+k<=b}
{a,...,b}_RThaynara2 = {k∈R | a<=k<=b}
{a,...,b}_RThiago    = {k∈N | a<=a+k<=b}
{a,...,b}_RRodrigo   = {∃k∈N | 0<=k<=b ∧ k+a<=b}
{a,...,b}_RNatan4    = {k∈N | a+k∈(a,b)}
{a,...,b}_RNatan5    = {k∈N | a+k∈[a,b]}
{a,...,b}_RRodrigo2  = {a+k | k∈{0,1,2}}
{a,...,b}_RNatan'    = {k∈R | x<b ∧ a+x=k}
```

Vimos que este programa em C,

```
int main(void) {
  float a=2.1, b=4.1, k;
  for (k=0; a+k<=b; k++)
    printf("%f\n", a+k);
}
```

meio que corresponde a esta expressão em matematiquês:

```
{a+k | k∈Z, a+k<=b}
```

11ª aula (14/set): Definições indutivas e notação para definições por casos. Exerçícios:

```
(F_0, F_1, F_2, ...) = (0, 1, 1, 2, 3, 5, ...)   (Fibonacci)
(a_0, a_1, a_2, ...) = (0, 1, 1+2, 1+2+3, ...)
                           1  1+2  1+2+3     )
(b_0, b_1, b_2, ...) = (0, -, ---, -----, ...)
                           1   2     3
```

a%d = resto da divisão de a por d

12ª aula (15/set): Introdução a gramáticas e BNF.

Discutimos uma gramática parecida com esta ("G1"),

```
digit  : '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'   /* 1 */
digits : digit                      /* 2 */
       | digit digits               /* 3 */
number | digits                     /* 4 */
exp    : number                     /* 5 */
       | exp '+' exp                /* 6 */
       | exp '-' exp                /* 7 */
       | exp '*' exp                /* 8 */
       | exp '/' exp                /* 9 */
       | '-' exp                    /* 10 */
       | exp '^' exp                /* 11 */
       | '(' exp ')'                /* 12 */
```

e como o conjunto dos strings que são "exp"s pode ser expresso como uma construção indutiva; vimos como provar que "2*3+(-45*6)" é uma "exp". Obs: esta gramática é adaptada da que aparece aqui:

(find-node "(bison)Infix Calc")

13ª aula (21/set): A gramática G1, acima, tem "ambiguidades". Vimos que há vários modos de parsear a expressão "2*3+(-45*6)", e que podemos usar uma relação Val_G1 pra definir precisamente o que quer dizer "s é uma expressão válida que pode ter o valor v". Por exemplo:

```
    Val_G1("2*3",6)  Val_G1("(-45*6)",-270)
    ---------------------------------------
           Val_G1("2*3+(-45*6)",-264)
```

e:

```
    Val_G1("2",2)  Val_G1("3+(-45*6)",-267)
    --------------------------------------
           Val_G1("2*3+(-45*6)",-534)
```

um modo de consertá-la é este. A gramática G2 é:

```
digit  : '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'  /* 13 */
digits : digit                      /* 14 */
       | digit digits               /* 15 */
```

```
  number | digits                        /* 16 */
  pexp   : number                        /* 17 */
         | '(' aexp ')'                  /* 18 */
  mexp   : pexp                          /* 19 */
         | mexp '*' pexp                 /* 20 */
         | mexp '/' pexp                 /* 21 */
  aexp   : mexp                          /* 22 */
         | '-' mexp                      /* 23 */
         | aexp '+' mexp                 /* 24 */
         | aexp '-' mexp                 /* 25 */
```

Esta gramática nos dá a noção certa de "subexpressão" - quando parseamos "2*3+(-45*6)" as subexpressões são os substrings que aparecem como "pexp"s, "mexp"s ou "aexps".

[[Falta uma parte]]

14ª aula (22/set): variáveis ligadas, substituição, regras de dedução.

```
      /------------y------------\
      |               /---x---\ |
      |               |       | |
  P(x,y) := ∀x∈{2,3}.∀x∈{3,4}.x<y
```

é equivalente a:

```
      /------------y------------\
      |               /---z---\ |
      |               |       | |
  P(x,y) := ∀x∈{2,3}.∀z∈{3,4}.z<y
```

e daí

```
  P(5,6) = ∀x∈{2,3}.∀z∈{3,4}.z<6
```

[[Acabei não transcrevendo muita coisa! Usei alguns exemplos bastante bons, preciso pegá-los do caderno de algum aluno...]]

15ª aula (28/set): Introdução a provas formais "em Fitch".

[[Preciso pôr aqui os exemplos e exercícios da aula...]]

16ª aula (29/set): Que regras são válidas numa prova formal (em Fitch)? Nós já vimos várias regras válidas ("*" = caveirinha),

```
  | P
  | Q      | P∧Q     | P∧Q
  +----    +----     +----
  | P∧Q    | P       | P
  (conj)   (simp1)   (simp2)
                                        | | P
  | P      | P->Q                       | +---
  | P->Q   | ¬Q      | P      | ¬¬P      | | :
  +-----   +-----    +----    +----      | | :
  | Q      | P       | ¬¬P    | P        | | Q
  (MP)     (MT)      (neg)    (neg)      | P->Q
                                        (-> Intro (*))

  | P      | Q
  +----    +----
  | PvQ    | PvQ
  (vI1)    (vI2)
```

```
    Links:
    (find-books "__logic/__logic.el" "barwise-etchemendy")
    (find-barwiseetchemendypage (+ 10 148) "6.2 Disjunction rules")
    (find-barwiseetchemendypage (+ 10 206) "8.2" "-> Intro")
```

Mas estas não são todas, e nenhuma pessoa normal decora todas.

A qualquer momento numa prova em Fitch podemos colocar uma "conclusão" que seja algo que já sabíamos - por exemplo os axiomas do apêndice C do Scheinerman - ou uma tautologia, ou algo que provamos "por contas"... Lembre que cada novo passo tem que ter uma "justificativa", e ela tem que ser válida.

Como é que tautologias viram regras de dedução?

Exemplo: ∀P,Q,R∈{F,V}.((P->R)∧(Q->R)->(PvQ->R)). Calcular por redução ("-~->") que isto é verdade dá um trabalhão (tente em casa - procure modos de reduzir as contas, depois tente explicá-los em português de forma que todo mundo entenda); verificamos isto com uma tabela improvisada:

```
   P  Q  R   (P-->R)∧(Q-->R)-->(P∨Q-->R)
   --------+---------------------------
   F  F  F |                 V  FFF V
   F  F  V |                 V      V V
   F  V  F |        F V F F  V
   F  V  V |                 V      V V
   V  F  F |  V F F F        V
   V  F  V |                 V      V V
   V  V  F |        F V F F  V
   V  V  V |                 V      V V
```

[[completar]]

(find-QUADROfile "" "2011-09-29-MD")
(find-QUADRO "2011-09-29-MD-1.jpg")
(find-QUADRO "2011-09-29-MD-2.jpg")
(find-QUADRO "2011-09-29-MD-3.jpg")

Em exercício (grande & difícil; pedi pra tentarem improvisar antes de ver as regras de verdade). Digamos que sabemos que:

  Def: a é par <-> ∃x∈Z.2x=a
  Def: a é ímpar <-> ∃x∈Z.2x+1=a
  e que ∀x∈Z.(x é par)∨(x é ímpar).

Prove que ∀x∈Z.x^2+x é par.

Um exemplo de uma dedução em Fitch com regras ∃-intro e ∃-elim - a regra ∃-elim é uma das regras difíceis (com caveira!) que envolvem hipóteses temporárias que são descartadas.

```
   1 | a é par
     +--------
   2 | ∃x∈Z.2x=a               (por 1 e pela def de par)
   3 | | x∈Z                  (HIPÓTESE TEMPORÁRIA!)
   4 | | 2x=a                 (HIPÓTESE TEMPORÁRIA!)
     | +-----
   5 | | a^2+a = (2x)^2+(2x)  (por 4 e álgebra)
   6 | | a^2+a = 2(2x^2+x)    (por 5 e álgebra)
   7 | | 2x^2+x∈Z             (por 3 e álgebra)
   8 | | ∃y∈Z.a^2+a=2y        (por 6,7, ∃-intro)
   9 | | a^2+a é par          (por 8 e pela def de par)
  10 | a^2+a é par            (por 2-9, ∃-elim)
```

17ª aula (05/out): Hoje: conjuntos!

(Dica: pra seguir esta parte do curso é melhor usar o livro da Judith, principalmente o cap.3).

Conjuntos são definidos pelos seus elementos: se A e B são conjuntos, então

  A=B <-> ∀x.(x∈A<->x∈B)

Note que aqui apareceu um "∀_._" - estávamos usando só "∀_∈_._"!

Além disso:

  y∈{x∈A|P(x)} <-> y∈A∧P(y)

Note que os "<->"s acima podem ser utilizados como _definições_ para igualdade de conjuntos e para "_∈{_|_}".

  Def: [a,b] = {x∈R|a<=x<=b}
  Def: a<=b<=c <-> a<=b∧b<=c

Vamos ver como provar formalmente coisas como

  ∀x∈[3,4].2<x   e
  An(BnC)=(AnB)nC,

e no caminho vamos entender mais coisas sobre as regras do "∀".

[*CAV*] O livro da Judith - e o do Scheinerman - usam principalmente "∀_._" e "∃_._" ao invés de "∀_∈_._" e "∃_∈_._" - isso é horrível para _calcular_ coisas, mas facilita _demonstrações_.

Duas definições possíveis (equivalentes) para "∩":

  Def: A∩B = {a∈A|a∈B}   (*)
  Def: x∈A∩B <-> x∈A∩x∈B   (**)

Não estamos acostumados a usar definições como a (**)...

Obs: def: P<->Q <-> (P->Q)∧(Q->P)

Exercícios: prove que

```
  | x∈A∩B    | x∈(A∩B)∩C   |
  +------    +----------   |
```

```
  |  :       |  :          | :
  | x∈B∩A    | x∈A∩(B∩C)   | x∈(A∩B)∩C<->x∈A∩(B∩C)
```
Alguns modos de resolvê-los:
```
  1 | x∈A∩B
    +------
  2 | x∈A∧x∈B  (por (**))
  3 | x∈A
  4 | x∈B
  5 | x∈B∧x∈A
  6 | x∈B∩A    (por (**))
```
e
```
   1 | x∈(A∩B)∩C
     +----------
   2 | x∈(A∩B)∧x∈C  (por 1, (**))
   3 | x∈A∩B
   4 | x∈A∧x∈B      (por 3, (**))
   5 | x∈A
   6 | x∈B
   7 | x∈C
   8 | x∈B∧x∈C
   9 | x∈B∩C        (por 8, (**))
  10 | x∈A∧x∈B∩C
  11 | x∈A∩(B∩C)   (por 10, (**))
```
e
```
  100 | | x∈A∩(B∩C)
      | +----------
  101 | | x∈A∧x∈B∩C
      | | :
  148 | | x∈(A∩B)∩C
  149 | x∈A∩(B∩C)->x∈(A∩B)∩C    (por 101-148, ->Intro)
  150 | | x∈(A∩B)∩C
      | +----------
      | | :
  197 | | x∈A∩(B∩C)
  198 | x∈(A∩B)∩C->x∈A∩(B∩C)    (por 150-197, ->Intro)
  199 | x∈(A∩B)∩C<-x∈A∩(B∩C)    (por 149, reescrevendo P->Q como Q<-P)
  200 | x∈(A∩B)∩C<->x∈A∩(B∩C)   (por 198, 199, def do "<->" (conj))
```
Note que a passagem 197,198->200 poderia ser feita "mais
honestamente" assim:
```
  198   | P->Q
  199   | P<-Q
  199.5 | (P->Q)∧(P<-Q)
  200   | P<->Q             (por 199.5, def do "<->")
```
Regras novas ([*CAV*] - são difíceis de formalizar):
```
  | P(x)
  +-----
  |  :
  | Q(x,y)
  | ∀y.Q(x,y)   (∀-Intro)
```
a regra "∀-Intro", que generaliza a conclusão de Q(x,y) para
∀y.Q(x,y), _só pode ser utilizada quando não há nenhuma hipótese
adicional sobre y_ - ou seja, quando o y que estávamos utilizando
"podia ser qualquer y" (a hipótese P(x) acima está lá só pra sugerir
que "nenhuma das hipóteses envolve condições sobre o y).
Variação:
```
  | P(x)
  +-----
  | | y∈A
  | +----
  | | :
  | | Q(x,y)
  | ∀y∈A.Q(x,y)   (∀-Intro)
```
Com estas regras podemos fazer:
```
  200 | x∈(A∩B)∩C<->x∈A∩(B∩C)
  201 | ∀x.(x∈(A∩B)∩C<->x∈A∩(B∩C))
```

    202 | (A∩B)∩C=A∩(B∩C)
18ª aula (06/out):
    (find-QUADROfile "" "2011-10-05-MD")
    (find-QUADRO   "2011-10-05-MD-1.jpg")
    (find-QUADRO   "2011-10-05-MD-2.jpg")
    (find-QUADRO   "2011-10-05-MD-3.jpg")

19ª aula (12/out): feriado (dia das crianças e dia nacional da luta
  contra a corrupção).
20ª aula (13/out):
    (find-QUADROfile "" "2011-10-13-MD")
    (find-QUADRO   "2011-10-05-MD-1.jpg")
    (find-QUADRO   "2011-10-05-MD-2.jpg")
    (find-QUADRO   "2011-10-05-MD-3.jpg")
    (find-QUADRO   "2011-10-05-MD-4.jpg")

21ª aula (19/out): semana acadêmica
22ª aula (20/out): semana acadêmica

23ª aula (26/out): revisão pra prova
    (find-QUADROfile "" "2011-10-26-MD")
24ª aula (27/out): Pegamos uma demonstração em Português que apareceu
  na P2 do semestre passado, que era:
    (A) Suponha que a é um inteiro que é par e ímpar ao mesmo tempo.
        Então existem inteiros x e y tais que a=2x e a=2y+1, e temos
        2x=2y+1 e 1=2x-2y=2(x-y); portanto x-y = 1/2, que não pertence
        a Z, mas como x,y∈Z temos x-y∈Z, uma contradição.
  E tentei ver que dificuldades os alunos teriam para traduzí-la para
  Fitch. Dei as seguintes dicas:
    (1) nomeie as sub-sentenças (P, Q, P_1, ...)
        e subprovas (P->¬Q, ...);
    (2) divida o problema maior em vários subproblemas;
    (3) anote quais são as variáveis livres em cada sentença e as
        hipóteses que foram usadas pra chegar a cada conclusão;
    (4) **teste os subproblemas**;
    (5) liste as definições;
    (6) anote que regra de dedução você usou para chegar a cada
        conclusão;
    (7) ponha "?"s nos passos dos quais você não tem certeza ou que
        você acha que merecem mais detalhes;
    (8) traduza entre português e matematiquês, usando vários passos
        se necessário.
  Exercícios (pequenos):
    (I) Quais são as hipóteses da prova A e qual a conclusão?
    (II) Expresse A como "H_1∧H_2∧...∧H_n->C" ou como "∀_∈_._".
    (III) Liste as definições usadas em A e expresse-as em
          matematiquês formal.
  Não fomos muito longe 8-(.
  Algumas dificuldades que surgiram: às vezes as pessoas confundem
  definir "___ é par" e definir o conjunto dos números pares; e
    Def: par = {a | ∀x∈R | a=2x}
  tem _montes_ de problemas!...

25ª aula (02/nov): feriado - finados
  *** vou pôr aqui as idéias que surgiram em e-mails ***
  Um exemplo simples de como usar o ∃-elim:
        | .
        | :
    100 | ∃a∈A.a<20
    101 | | a∈A   (hipótese temporária)
    102 | | a<20   (hipótese temporária)
        | +-----
    103 | | a<30   (por 102 e álgebra)
    104 | | ∃a∈A.a<30 (por 101 e 103)
    105 | | ∃b∈A.b<30 (por 104, renomeando uma variável local)

```
  106 | ∃b∈A.b<30    (por ∃-elim - repare que tínhamos concluído
      | .             ∃b∈A.b<30 usando ∃a∈A.a<20, a∈A e a<20, mas
      | :             a regra ∃-elim nos permite descartar as hipóteses
                      a∈A e a<20 e manter só a hipótese ∃a∈A.a<20)
```

Uma dica importante:

Se uma dedução estiver certa então cada linha dela pode ser reescrita "com todas as suas hipóteses explícitas", como no exemplo abaixo:

```
  100 | ∃a∈A.a<20         (∃a∈A.a<20)             ->(∃a∈A.a<20)
  101 | | a∈A             (∃a∈A.a<20)∧(a∈A)       ->(a∈A)
  102 | | a<20            (∃a∈A.a<20)∧(a∈A)∧(a<20)->(a<20)
      | +-----
  103 | | a<30            (∃a∈A.a<20)∧(a∈A)∧(a<20)->(a<30)
  104 | | ∃a∈A.a<30       (∃a∈A.a<20)∧(a∈A)∧(a<20)->(∃a∈A.a<30)
  105 | | ∃b∈A.b<30       (∃a∈A.a<20)∧(a∈A)∧(a<20)->(∃b∈A.b<30)
  106 | ∃b∈A.b<30         (∃a∈A.a<20)             ->(∃b∈A.b<30)
```

e todas estas "proposições com hipóteses explícitas" têm que ser verdadeiras - e sem hipóteses adicionais!

Repare que neste exemplo o valor de cada proposição que apareceu na coluna da direita "depende" do valor de algumas variáveis. Mas precisamente: só sabemos CALCULAR - pelas regras de redução - o valor da expressão da linha 100,

```
  100                                 (∃a∈A.a<20)->(∃a∈A.a<20)
```

se A for conhecido, e só sabemos CALCULAR o valor de

```
  105                    (∃a∈A.a<20)∧(a∈A)∧(a<20)->(∃b∈A.b<30)
```

se conhecemos A e a...

O problema dos "inteiros pares e ímpares ao mesmo tempo"

A tradução para Fitch dele é:

```
   1 | a∈Z
   2 | a é par e ímpar
     +-------------------------
   3 | (∃x∈Z.2x=a)∧(∃y∈Z.2y+1=a)
   4 | ∃x∈Z.2x=a
   5 | ∃y∈Z.2y+1=a
   6 | | x in Z
   7 | | 2x=a
     | +---------
   8 | | | y in Z
   9 | | | 2y+1=a
     | | +--------
  10 | | | 2x=2y+1
  11 | | | 2x-2y=1
  12 | | | 2(x-y)=1
  13 | | | x-y=1/2
  14 | | | ¬(x-y∈Z)
  15 | | | x-y∈Z
  16 | | | F
  17 | | F
  18 | F
```

Exercícios:

1) Explique porque - e em que sentido - P∧Q->F é equivalente a P->¬Q.
2) Escreva à direita de cada proposição da prova acima a sua "versão com hipóteses explícitas", e as variáveis das quais ela depende.
3) Explique esta dedução:

```
  1 | ¬(x-y∈Z)
  2 | x-y∈Z
    +------
  3 | F
```

4) Complete a dedução acima (a de 18 linhas) dizendo o nome da regra utilizada em cada passo. Obs: este exercício é uma

desculpa pra fazer você se familiarizar com as regras do Fitch.

5) Explique (pode ser em Português) como é que a demonstração acima prova que não existe inteiros que são pares e ímpares ao mesmo tempo.
6) (Difícil.) Releia a seção 1.4 - "Lógica de Predicados" - do livro da Judith Gersting, e compare o "axioma 6" dela com a regra "∃-elim" do Fitch. Obs: as "várias livres" nas deduções funcionam de modos completamente diferentes no Fitch e no sistema da Judith...
7) Verifique que "∃a∈A.P(a)" é equivalente a "∃a.(a∈A∧P(a))" (se você não achar isto óbvio), e explique porque é que a nossa regra pro "∃" introduz _duas_ hipóteses temporárias ao invés de uma só; compare com a do Barwise/Etchemendy, que introduz uma hipótese temporária só.
8) Traduza para Fitch a demonstração que aparece na questão 6 da P2 do semestra passado:
   http://angg.twu.net/MD/MD_P2_2011jun22.pdf
9) Traduza para Fitch mais demonstrações que aparecem no capítulo 2 do livro da Judith.

Obs: há um scan de uma parte do livro da Judith em:
http://angg.twu.net/MD/judith_p59_a_110.pdf
e um scan inteiro dele (e do Barwise/Etchemendy também) no bcclivros.

26ª aula (03/nov): P1 - matéria: construções indutivas e um pouco de demonstrações - basicamente tudo o que vimos até agora!

27ª aula (09/nov):
28ª aula (10/nov):

29ª aula (16/nov):
30ª aula (17/nov):

31ª aula (23/nov):
32ª aula (24/nov):

33ª aula (30/nov):
34ª aula (01/dez):

35ª aula (07/dez):
36ª aula (08/dez):