PLLIC - Provador para as Lógicas Linear, Intuicionista e Clássica

Elaine Gouvêa Pimentel

25 de janeiro de 2007

Resumo

O presente texto visa apresentar o provador automático de teoremas PLLIC, que verifica quando seqüentes do tipo $\Gamma \vdash_L \Delta$ são prováveis, onde Γ, Δ são conjuntos de fórmulas e L pode ser uma das seguintes lógicas: linear LL, intuicionista LJ ou clássica LK. Apesar de ser possível expandir o programa para o uso de quantificadores (universal e existencial), PLLIC decide sobre a provabilidade do fragmento *proposicional* das lógicas acima citadas, uma vez que a nossa intenção é que o programa sempre pare, ou seja, que a pergunta: *o seqüente* $\Gamma \vdash_L \Delta$ *é provável?* tenha sempre uma resposta: *sim* ou *não*. Além disso, se um seqüente é provável, PLLIC exibe a sua prova em cálculo de seqüentes.

Sumário

1 Introdução

2 Lógica de Primeira Ordem

Nesta seção, descreveremos a Lógica de Primeira Ordem, que é um sistema de argumentação simbólica em que cada sentença ou afirmativa é composta de *sujeitos* e *predicados*. São exemplos de predicados: "*todo homem é mortal*", "*p ou não p*" e "*para todo n, a soma de n e zero é n*".

Os predicados a serem utilizados na prática dependem do problema específico no qual estamos trabalhando. Dos exemplos de predicado acima, o primeiro é o clássico exemplo estudado por Aristóteles; o segundo é a fórmula lógica que aparece no famoso princípio do terceiro excluído, sobre o qual falaremos mais tarde; e o último... bem, depende de quem é n! Caso n seja um número natural, o predicado define o caso base da soma de números naturais. Já se n é um número real, estamos simplesmente dizendo que a soma de zero com um número qualquer n é o próprio n. Desta forma, para cada predicado devemos descrever o seu *tipo*.

Começaremos portanto por descrever a noção de tipos e assinaturas, que são conjuntos finitos de constantes, cada uma com seu tipo definido. Passaremos então a definir a sintaxe da lógica de primeira ordem, isto é, como podemos construir *fórmulas* a partir de predicados atômicos e alguns conectivos. Por fim, detalharemos um *sistema de provas* baseado em *cálculo de seqüentes* para a lógica de primeira ordem.

2.1 Tipos e Assinaturas

Tipos estão presentes tanto em matemática (e portanto em lógica matemática) quanto em computação. Na teoria de conjuntos tradicional, o agrupamento de elementos em um conjunto independe da natureza desses elementos. Quando passamos a trabalhar em aplicações específicas, precisamos classificar os objetos em categorias, de acordo com o seu uso ou aplicação.

A noção de tipos origina-se dessa classificação: um tipo é uma coleção de objetos ou valores que possuem alguma propriedade em comum.

Seja S um conjunto fixo, finito de tipos primitivos. O conjunto de tipos primitivos que tomamos depende do tipo de problema que queremos resolver, ou do tipo de aplicação que estamos interessados.

Por exemplo, em linguagens de programação tipadas existem alguns tipos pré-definidos, implícitos. Alguns tipos comuns em linguagens de programação são, por exemplo, o tipo primitivo `int`, o tipo primitivo `real` e o tipo primitivo `string`. Na sintaxe do *λ-Prolog* [22], escrevemos

```
kind   int      type.
kind   real     type.
kind   string   type.
```

para representar que `int`, `real` e `string` são tipos.

No caso de linguagens de programação *lógicas* como Prolog ou λ-Prolog, existe um tipo primitivo relativo a predicados, o tipo `o` (veja Church [6]):

```
kind   o        type.
```

No presente texto, assumiremos que `o` é sempre membro de S.

O *conjunto dos tipos* T é o menor conjunto de expressões que contenha os tipos primitivos S e seja fechado com relação à construção de tipos funcionais, construídos com o símbolo binário infixado $\rightarrow$. Ou seja, tipos de T são dados pela sintaxe abaixo:

Tipos	$\tau ::=$	$\tau,\ \tau \in S$	tipo primitivo
		$\tau_1 \rightarrow \tau_2$	tipo funcional

Figura 1: Sintaxe de tipos

Utilizaremos letras gregas τ e σ para representar variáveis sintáticas para tipos. Como de maneira usual, o construtor de tipos $\rightarrow$ associa-se à direita: lê-se $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$ como $\tau_1 \rightarrow (\tau_2 \rightarrow \tau_3)$.

Seja τ o tipo $\tau_1 \rightarrow \tau_2 \rightarrow \ldots \rightarrow \tau_n \rightarrow \tau_0$ onde τ_0 é um tipo primitivo (ou seja, $\tau_0 \in S$) e $n \geq 0$. Os tipos $\tau_1, \ldots, \tau_n$ são ditos *tipos argumento* de τ enquanto que o tipo τ_0 é chamado *tipo alvo* de τ.

Exemplo 1 *Suponha que estejamos interessados em saber se um determinado aluno teve um desempenho ruim durante o semestre, como descrito abaixo:*

```
kind pessoa type.

type a                              pessoa.
type professor, naoestudou          pessoa -> o.
type desempenhoruim                 pessoa -> o.
type aluno, naopassou               pessoa -> pessoa -> o.

desempenhoruim X :- pi y\(professor y =>aluno X y => naopassou X y).
naopassou A B :- professor B, aluno A B, naoestudou A.

naoestudou a.

% Pergunta-se: ?- desempenhoruim a.
```

O código acima retrata a situação em que um aluno possui um desempenho ruim se para toda[1] *disciplina que ele cursa ele não passa. E um estudante não passa se ele não estuda.*

O único tipo primitivo para este exemplo é `pessoa`*, que pertence ao conjunto* S *juntamente com o tipo* `o`*.*

Os predicados `professor, naoestudou, naopassou, desempenhoruim` *e* `aluno` *possuem tipos funcionais.*

Um conceito muito importante é o de *ordem* de um tipo.

Definição 1 *A ordem de um tipo* τ *(representado por* $ord(\tau)$*) é definida da seguinte forma:*

[1] Em λ-Prolog, representamos *para todo* x como `pi x\`.

$$\begin{array}{lcl} ord(\tau) & = & 0 \text{ se } \tau \in S \\ ord(\tau_1 \to \tau_2) & = & max\{ord(\tau_1)+1, ord(\tau_2)\} \end{array}$$

No Exemplo 1, o tipo primitivo `pessoa` possui ordem 0 pois pertence ao conjunto S. Se chamamos $\tau = pessoa \to \mathtt{o}$, então

$$\begin{array}{rcl} ord(\tau) & = & max\{ord(pessoa)+1, ord(\mathtt{o})\} \\ & = & max\{0+1, 0\} \\ & = & 1. \end{array}$$

Por sua vez, se $\tau' = pessoa \to pessoa \to \mathtt{o}$, então

$$\begin{array}{rcl} ord(\tau') & = & max\{ord(pessoa)+1, ord(\tau)\} \\ & = & max\{0+1, 1\} \\ & = & 1. \end{array}$$

Fica claro então por indução que se $\tau = \tau_1 \to \ldots \to \tau_n \to \tau_0$ onde $\tau_0, \tau_1, \ldots, \tau_n \in S$, então $ord(\tau) = 1$.

Definição 2 *Os tipos τ de ordem 0 ou 1 tais que nenhum tipo argumento de τ é* `o` *são chamados de* tipos de primeira ordem.

Desta forma, todos os predicados definidos no Exemplo 1 possuem tipo de primeira ordem.

Como de maneira usual, assumimos que para cada tipo τ existam inúmeras constantes e variáveis deste tipo. Também constantes e variáveis não se sobrepõem, e se duas constantes ou variáveis têm tipos diferentes, são constantes ou variáveis diferentes.

Uma assinatura Σ sobre S é um conjunto finito de constantes. Representaremos Σ listando seus membros como pares da forma $a : \tau$, onde a é uma constante de tipo τ. Uma assinatura é de primeira ordem se todas suas constantes possuem tipos de primeira ordem.

2.2 Termos e fórmulas de primeira ordem

Agora podemos definir a sintaxe da lógica de primeira ordem $\mathcal{F}$. As constantes lógicas de $\mathcal{F}$ são os símbolos: $\wedge$ (conjunção), $\vee$ (disjunção), $\supset$ (implicação), $true$ (verdadeiro), $false$ (absurdo), $\forall_\tau$ (quantificação universal) e $\exists_\tau$ (quantificação existencial) onde $\tau \in S - \{\mathtt{o}\}$. Por questões didáticas, incluiremos também a negação $\neg$ como constante lógica. Como veremos mais adiante, $\neg B$ é equivalente a $B \supset false$.

Exemplo 2 *Utilizando a sintaxe de λ-Prolog, a assinatura Σ de $\mathcal{F}$ pode ser especificada como:*

```
kind  form          type.
kind  i             type.

type  true, false   form.
type  neg           form -> form.
type  and, or, imp  form -> form -> form.
type  forall        (i -> form) -> form.
type  exists        (i -> form) -> form.
```

Passaremos agora a descrever algumas noções que aparecem bastante confusas na maioria dos textos existentes sobre lógica de primeira ordem. Seguiremos as definições apresentadas em [19].

Definição 3 *Seja τ um tipo da forma $\tau_1 \to \tau_2 \to \ldots \to \tau_n \to \tau_0$ onde τ_0 é um tipo primitivo e $n \geq 0$.*

i. Se τ_0 for `o`*, uma constante do tipo τ é uma* constante de predicado *de aridade n.*

ii. Se τ_0 não for `o`*, então uma constante do tipo τ é ou uma* constante individual *se $n = 0$, ou é uma* constante funcional *de aridade n se $n \geq 1$.*

No Exemplo 1, `a` é uma constante individual, `professor, naoestudou` e `desempenhoruim` são constantes de predicado de aridade 1 enquanto que `aluno` e `naopassou` são constantes de predicado de aridade 2.

No Exemplo 2, `true` e `false` são constantes individuais, `neg` é uma constante funcional de aridade 1, enquanto `and, or` e `imp` são constantes funcionais de aridade 2.

Similarmente, podemos definir variável de predicado de aridade n, variável individual e variável funcional de aridade n.

Definição 4 *Seja τ um tipo primitivo diferente de* `o`*. Um* termo de primeira ordem *de tipo τ pode ser: ou uma constante c, ou uma variável X de tipo τ, ou da forma $f\ t_1 \ldots t_n$ onde f é uma constante funcional de tipo $\tau_1 \to \cdots \to \tau_n \to \tau$. Neste caso, dizemos que f é a* cabeça *e $t_1 \ldots t_n$ são os* argumentos *do termo.*

Em outras palavras, termos de primeira ordem são dados pela gramática:

$$t \quad ::= \quad c \mid X \mid f\ t_1 \ldots t_n$$

Definição 5 *Uma* fórmula de primeira ordem *pode ser atômica ou não-atômica.*

i. Uma fórmula atômica é da forma $p\ t_1 \ldots t_n$ onde $n \geq 0$, p é uma constante de predicado de tipo $\tau_1 \to \cdots \to \tau_n \to o$, e $t_1, \ldots, t_n$ são termos de primeira ordem de tipos $\tau_1, \ldots, \tau_n$, respectivamente. A constante de predicado p é a cabeça desta fórmula atômica.

ii. Fórmulas não-atômicas de $\mathcal{F}$ são dadas pela gramática

$$F \quad ::= \quad (F \wedge F) \mid (F \vee F) \mid (F \supset F) \mid false \mid true \mid \forall_\tau x.F \mid \exists_\tau x.F$$

onde τ é um tipo primitivo diferente de `o`*.*

Exemplo 3 *Seguindo o trabalho de especificação da lógica de primeira ordem, podemos descrever fórmulas atômicas e não atômicas:*

```
nao_atomico (A imp B).
nao_atomico (A and B).
nao_atomico (A or B).
nao_atomico false.
nao_atomico true.
```

```
nao_atomico (neg A).
nao_atomico (forall B).
nao_atomico (exists B).

atomico B :- not (nao_atomico B).
```

Observe que `nao_atomico` *e* `atomico` *são predicados de tipo*

```
form -> o
```

e que `not` *é a negação do λ-Prolog.*

2.3 Cálculo de Seqüentes

Um sistema lógico formal é composto, além da sintaxe (ou notação, como a que acabamos de apresentar), de uma especificação cuidadosa de regras de argumentação (regras de inferência) e de alguma noção de como interpretar e dar um significado a sentenças (ou proposições) da linguagem adotada (semântica).[2]

Na presente seção, apresentaremos um sistema de provas baseado em cálculo de seqüentes para a lógica de primeira ordem $\mathcal{F}$. Chamaremos tal sistema de **G**, seguindo a notação de [39], e em homenagem a Gentzen.

Foi o alemão Gerhard Gentzen que introduziu o cálculo de seqüentes nos anos 1930[3] [9], cálculo este que permite lidar com verdades lógicas considerando a forma da dedução.

Um seqüente de $\mathcal{F}$ é uma tripla:

$$\Sigma : \Gamma \vdash \Delta$$

onde Σ é uma assinatura de primeira ordem sobre S, Γ e Δ são multi-conjuntos finitos (possivelmente vazios) de fórmulas e $\vdash$ é um meta-símbolo de validade. Chamamos Γ de *antecedente* e Δ de *suscedente* do seqüente. Como de costume, denotaremos a união $\Gamma \cup \{B\}$ por Γ, B. Em geral, também omitiremos a assinatura, escrevendo um sequente simplesmente como: $\Gamma \vdash \Delta$.

Uma prova para o seqüente $\Gamma \vdash \Delta$ é uma *árvore finita* cuja *raiz* é o próprio seqüente, os *galhos* são construídos usando as regras de inferência, e as *folhas* são instâncias do único axioma do cálculo, o axioma inicial (Figura 2).

As regras de inferência de **G**, por sua vez, podem ser separadas em três grupos: o único axioma do sistema (Figura 2), as regras de introdução de conectivos lógicos (Figura 3) as regras estruturais (Figura 4), entre elas a regra *cut* sobre a qual falaremos mais tarde.

Vamos descrever agora alguns exemplos de provas utilizando cálculo de sequentes. Começaremos por mostrar que, na lógica de primeira ordem, a negação de uma fórmula B é equivalente a $B \supset false$,

Exemplo 4 *Para demonstrar a equivalência* $\neg B \equiv B \supset false$*, devemos provar que* $\neg B \vdash B \supset false$ *e também que* $B \supset false \vdash \neg B$*:*

[2] No presente texto não trataremos de semãntica. Em [28] é apresentada a semântica formal, via álgebras de Boole, da lógica clássica proposicional e a semântica da lógica intuicionista, via álgebras de Heyting (pseudo-álgebras de Boole). Para a semântica da lógica linear, veja [10] e [38].

[3] Gentzen que também introduziu o sistema de *dedução natural* – para uma introdução ao sistema de dedução natural, veja [30] ou [28].

$$\frac{}{A \vdash A}\ \mathit{Inicial}$$

Figura 2: Axioma inicial de **G**

$$\frac{}{\Gamma \vdash \mathit{true}, \Delta}\ \mathit{trueR} \qquad \frac{\Gamma_1 \vdash A, \Delta_1 \quad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash A \wedge B, \Delta_1, \Delta_2}\ \wedge R$$

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \supset B, \Delta}\ \supset R \qquad \frac{\Gamma, A \vdash \mathit{false}, \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R1 \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R2$$

$$\frac{\Gamma \vdash A[x/y], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall R \qquad \frac{\Gamma \vdash A[x/t], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists R$$

$$\frac{}{\Gamma, \mathit{false} \vdash \Delta}\ \mathit{falseL} \qquad \frac{\Gamma_1 \vdash A, \Delta_1}{\Gamma_1, \Gamma_2, \neg A \vdash \Delta_1, \Delta_2}\ \neg L$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L1 \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L2$$

$$\frac{\Gamma_1, A \vdash \Delta_1 \quad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \vee B \vdash \Delta_1, \Delta_2}\ \vee L \qquad \frac{\Gamma_1 \vdash A, \Delta_1 \quad \Gamma_1, B \vdash \Delta_1}{\Gamma_1, \Gamma_2, A \supset B \vdash \Delta_1, \Delta_2}\ \supset L$$

$$\frac{\Gamma, A[x/t] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall L \qquad \frac{\Gamma, A[x/y] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists L$$

Figura 3: Regras de introdução de **G**

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ \mathit{weak\ L} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}\ \mathit{weak\ R}$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ \mathit{cont\ L} \qquad \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}\ \mathit{cont\ R}$$

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \mathit{Cut}$$

Figura 4: Regras estruturais de **G**

- $\neg B \vdash B \supset false$:

$$\dfrac{\dfrac{\dfrac{}{B \vdash B}\ Inicial}{\neg B, B \vdash false}\ \neg L}{\neg B \vdash B \supset false}\ \supset R$$

- $B \supset false \vdash \neg B$:

$$\dfrac{\dfrac{\dfrac{}{B \vdash B}\ Inicial \quad \dfrac{}{false \vdash false}\ falseL}{B \supset false, B \vdash false}\ \supset L}{B \supset false \vdash \neg B}\ \neg R$$

Outro exemplo importante é o princípio do terceiro excluído.

Exemplo 5 *Na lógica cássica de primeira ordem (veja Seção 3) vale o tão comentado* princípio do terceiro excluído. *Ou seja, a proposição*

$$p \vee \neg p$$

é sempre válida. Isso significa que uma fórmula é sempre ou verdadeira, ou falsa.

Essa afirmação é extremamente não construtiva, uma vez que nada se pode dizer sobre qual das opções é valida. A prova em cálculo de seqüentes no sistema **G** *é dada abaixo:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash p}\ Inicial}{p \vdash false, p}\ weakR}{\vdash p, \neg p}\ \neg R}{\vdash p, p \vee \neg p}\ \vee R2}{\vdash p \vee \neg p, p \vee \neg p}\ \vee R1}{\vdash p \vee \neg p}\ contR$$

Observe que esta prova só é possível porque podemos utilizar regras estruturais à direita do seqüente. Desta forma, evitamos escolher entre provar p ou $\neg p$, simplesmente através da duplicação do suscedente. Como veremos na Seção 4, a proposição $p \vee \neg p$ não pode ser provada na lógica intuicionista.

O cálculo de seqüentes possui uma série de características interessantes:

- o cálculo de seqüentes possui apenas regras de introdução para conectivos, isto é, regras que, quando lidas de cima para baixo (*top-down*), introduzem um conectivo lógico;
- premissas e conclusões são tratadas da mesma forma e são construídas simultaneamente;
- é tecnicamente bem simples: quando lidas de baixo pra cima (*bottom up*), fica claro que as regras no cálculo de seqüentes simplificam o processo de construção de provas.

Desta forma, é fácil especificar todas as regras de introdução descritas nas Figuras 2 e 3. O axioma inicial também é facilmente especificado. Acontece que a presença de regras estruturais pode trazer alguns problemas de implementação.

Por exemplo, a implementação de *contraction* é inviável, uma vez que geraria loops.

Não existe um sistema único de provas sem regras estruturais que represente ambas as lógicas clássica e intuicionista. Desta forma, a partir deste ponto vamos dividir a nossa discussão de acordo com a lógica em questão, com excessão da regra *cut*, que será analisada na Seção 5.

3 Lógica Clássica

Podemos substituir o sistema **G** dado pelas regras nas Figuras 2, 3 e 4 pelo sistema dado na Figura 5, que chamaremos de **G3c**, ainda seguindo [39].

Começamos por observar que podemos simular a regra $\vee R$ da Figura 5 utilizando as regras $\vee R1$ e $\vee R2$ da Figura 3, na presença da regra *contraction* (Figura 4). De fato,

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A, A \vee B}\ \vee R2}{\Gamma \vdash \Delta, A \vee B, A \vee B}\ \vee R1}{\Gamma \vdash \Delta, A \vee B}\ contR$$

Por outro lado, não há um jeito de simular as regras $\vee R1$ e $\vee R2$ em **G3c** uma vez que não podemos utilizar *weakening* explicitamente. Mas a ocorrência desta regra implícita no axioma inicial faz com que, para cada prova de um certo seqüente em **G** tal que as regras $\vee R1$ e/ou $\vee R2$ sejam aplicadas, exista uma outra prova do mesmo seqüente em **G3c** tal que a regra $\vee R$ seja aplicada.

Não vamos demonstrar essa afirmação, mas ilustrá-la com um exemplo.

Exemplo 6 *Considere a seguinte prova em* **G***:*

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\ Inicial}{A \vdash B \vee A}\ \vee R2 \qquad \dfrac{\dfrac{}{B \vdash B}\ Inicial}{B \vdash B \vee A}\ \vee R1}{A \vee B \vdash B \vee A}\ \vee L$$

Podemos construir uma prova similar em **G3c***:*

$$\dfrac{\dfrac{\dfrac{}{A \vdash B, A}\ Inicial}{A \vdash B \vee A}\ \vee R \qquad \dfrac{\dfrac{}{B \vdash B, A}\ Inicial}{B \vdash B \vee A}\ \vee R}{A \vee B \vdash B \vee A}\ \vee L$$

Observe que, para terminar a prova, basta que a mesma fórmula esteja tanto no antecedente quanto no sucedente do seqüente – não há mais a necessidade de que eles sejam iguais, entretanto.

De maneira dual, temos que a regra $\wedge L$ em **G3c** é equivalente[4] a $\wedge L1 + \wedge L2 + weakL + contL$.

Além disso, como os contextos são todos copiados em **G3c**, o seguinte resultado pode ser facilmente provado:

Proposição 1 *O sistema* **G** *sem a regra CUT é equivalente ao sistema* **G3c**.

[4]No sentido de que **G3c** com a regra $\wedge R$ prova exatamente as mesmas fórmulas lógicas de **G** com as regras $\wedge R1$ e $\wedge R2$.

O sistema **G3c** é computacionalmente mais interessante que **G** uma vez que, como as regras estruturais *weak* e *cont* são *implícitas*, ele pode ser facilmente implementado.

$$\frac{}{\Gamma, A \vdash \Delta, A}\ Inicial$$

$$\frac{}{\Gamma \vdash true, \Delta}\ trueR \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge R$$

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \supset B, \Delta}\ \supset R \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R$$

$$\frac{\Gamma \vdash A, B\Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee R$$

$$\frac{\Gamma \vdash A[x/y], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall R \qquad \frac{\Gamma \vdash \exists x A, A[x/t], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists R$$

$$\frac{}{\Gamma, false \vdash \Delta}\ falseL \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ \neg L$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge L$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee L \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \supset B \vdash \Delta}\ \supset L$$

$$\frac{\Gamma, A[x/t], \forall x A \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall L \qquad \frac{\Gamma, A[x/y] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists L$$

Figura 5: Sistema **G3c** para a lógica clássica de primeira ordem

3.1 Especificando a lógica clássica

Para especificar um seqüente, devemos primeiro descrever um predicado que diz respeito à *provabilidade* de um seqüente:

```
type prova     list form -> list form -> o.
```

Ou seja, `prova` é um predicado que toma listas de fórmulas Γ e Δ e traz como resultado se Γ segue de Δ ou não.

Depois, seguimos especificando o axioma inicial:

```
prova G H :- membro A G, membro A H.
```

Ou seja, se `A` aparece em ambos os lados do seqüente, a prova acaba. Aqui `membro` é um predicado definido em um outro módulo do programa que trata de listas (veja Apêndice A). Intuitivamente, `membro A G` retorna verdadeiro se o elemento `A` pertence à lista `G`.

Podemos então seguir especificando as outras regras do sistema:

```
prova G H :- membro true H.
prova G H :- memb_and_rest (A and B) H H1, prova G (A::H1),
             prova G (B::H1).
```

```
prova G H :- memb_and_rest (A imp B) H H1, prova (A::G) (B::H1).
prova G H :- memb_and_rest (neg A) H H1, prova (A::G) H1.
prova G H :- memb_and_rest (A or B) H H1, prova G (A::(B::H1)).
prova G H :- memb_and_rest (forall A) H H1,
             pi x\(prova G ((A x)::H1)).
prova G H :- memb (exists A) H, prova G ((A T)::H).

prova G H :- membro false G.
prova G H :- memb_and_rest (neg A) G G1, prova G1 (A::H).
prova G H :- memb_and_rest (A and B) G G1, prova (A ::(B :: G1)) H.
prova G H :- memb_and_rest (A or B) G G1, prova (A :: G1) H,
             prova (B :: G1) H.
prova G H :- memb_and_rest (A imp B) G G1, prova (B::G1) H,
             prova G1 (A::H).
prova G H :- memb (forall A) G, prova ((A T)::G) H.
prova G H :- memb_and_rest (exists A) G G1,
             pi x\(prova ((A x)::G1) H).
```

A especificação do predicado `memb_and_rest` é apresentada no Apêndice A. Intuitivamente, `memb_and_rest A G G1` retorna verdadeiro se `A` aparece em `G` e `G1` é `G-A`.

Apesar de termos apresentado a especificação de toda lógica clássica de primeira ordem, PLLIC lida apenas com a *parte proposicional* das lógicas implementadas, uma vez que é um provador de teoremas *completamente automático* e *decidível*. Mas é importante notar que pode-se estender PLLIC de modo que este torne-se indecidível mas suporte quantificação, ou mesmo semi-automático suportando, entre outras coisas, indução (veja por exemplo [27]).

4 Lógica intuicionista

Como descrito na Seção 3, na lógica clássica afirmativas são ou falsas ou verdadeiras, uma vez que vale o princípio do terceiro excuído.

A lógica intuicionista abandona a idéia de verdade absoluta, e afirmativas são consideradas válidas se e somente se existe uma prova *construtiva* da mesma. Ou seja, o princípio do terceiro excluído não é mais uma *tautologia* uma vez que não existe um método construtivo único que prove *qualquer* proposição ou a sua negação.

No Exemplo 5, vimos que a fórmula $p \vee \neg p$ só pôde ser provada porque duplicamos o sucedente do seqüente. Desta forma, evitamos ter que dizer de imediato quem vale: p ou $\neg p$.

Para evitar que tenhamos esse tipo de escolha não construtiva em um sistema intuicionista, devemos restringir os seqüentes válidos àqueles que possuam *exatamente* uma fórmula como suscedente.[5] Desta forma, fica claro, por exemplo que no sistema **G** as regras de weakening and contraction não são válidas à direita. As regras de negação devem também ser ajustadas. A Figura 6 apresenta o sistema **Gi**, derivado do sistema **G** para a lógica intuicionista.

[5]Existem sistemas intuicionistas em que sequentes podem ter mais de uma fórmula no suscedente (veja [25]). Mas analisar esse tipo de cálculo está fora do nosso objetivo.

$$\frac{}{A \vdash A}\ Inicial$$

$$\frac{}{\Gamma \vdash true}\ trueR \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}\ \wedge R$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset R \qquad \frac{\Gamma, A \vdash false}{\Gamma \vdash \neg A}\ \neg R$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee R1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee R2$$

$$\frac{\Gamma \vdash A[x/y]}{\Gamma \vdash \forall x A}\ \forall R \qquad \frac{\Gamma \vdash A[x/t]}{\Gamma \vdash \exists x A}\ \exists R$$

$$\frac{}{\Gamma, false \vdash C}\ falseL \qquad \frac{\Gamma_1 \vdash A}{\Gamma_1, \Gamma_2, \neg A \vdash C}\ \neg L$$

$$\frac{\Gamma, A \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge L1 \qquad \frac{\Gamma, B \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge L2$$

$$\frac{\Gamma_1, A \vdash C \quad \Gamma_2, B \vdash C}{\Gamma_1, \Gamma_2, A \vee B \vdash C}\ \vee L \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_1, B \vdash C}{\Gamma_1, \Gamma_2, A \supset B \vdash C}\ \supset L$$

$$\frac{\Gamma, A[x/t] \vdash C}{\Gamma, \forall x A \vdash C}\ \forall L \qquad \frac{\Gamma, A[x/y] \vdash C}{\Gamma, \exists x A \vdash C}\ \exists L$$

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ weak\ L \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ cont\ L$$

$$\frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

Figura 6: Regras do sistema **Gi**

Como conseqüência da restrição sobre a forma do seqüente na lógica intuicionista, temos que os seguintes seqüentes clássicos não são prováveis em **Gi**:

$\vdash A \vee \neg A$	princípio do terceiro excluído
$\vdash \neg A \vee \neg\neg A$	princípio fraco do terceiro excluído
$\vdash \neg\neg A \vdash A$	lei de dupla negação
$\vdash (A \supset B) \vee (B \supset A)$	lei de Dummett
$\vdash ((A \supset B) \supset A) \supset A$	lei de Pierce
$\neg(\neg A \wedge \neg B) \vdash A \vee B$	

Essa última fórmula indica que as famosas *relações de dualidade de "de Morgan"* não são válidas.

4.1 Especificando a lógica intuicionista

Implementar a lógica intuicionista é um pouquinho mais trabalhoso que a lógica clássica, por causa da regra de implicação à esquerda, como veremos a seguir.

Começamos por adaptar o sistema **Gi** para o caso de regras estruturais *weakening* e *contraction* implícitas, obtendo o sistema **G3i** da Figura 7.

Observe que a regra $\supset L$ explicitamente copia a fórmula $A \supset B$ da conclusão

$$\frac{}{\Gamma, A \vdash A}\ Inicial$$

$$\frac{}{\Gamma \vdash true}\ trueR \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge R$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset R \qquad \frac{\Gamma, A \vdash false}{\Gamma \vdash \neg A}\ \neg R$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee R1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee R2$$

$$\frac{\Gamma \vdash A[x/y]}{\Gamma \vdash \forall x A}\ \forall R \qquad \frac{\Gamma \vdash A[x/t]}{\Gamma \vdash \exists x A}\ \exists R$$

$$\frac{}{\Gamma, false \vdash C}\ falseL \qquad \frac{\Gamma \vdash A}{\Gamma, \neg A \vdash C}\ \neg L$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}\ \wedge L$$

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}\ \vee L \qquad \frac{\Gamma, A \supset B \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \supset B \vdash C}\ \supset L$$

$$\frac{\Gamma, A[x/t], \forall x A \vdash C}{\Gamma, \forall x A \vdash C}\ \forall L \qquad \frac{\Gamma, A[x/y] \vdash C}{\Gamma, \exists x A \vdash C}\ \exists L$$

Figura 7: Sistema **G3i** para a lógica intuicionista de primeira ordem

para a premissa da esquerda. Isso causa problemas seríssimos de implementação, uma vez que o interpretador poderá *sempre* escolher utilizar essa regra, entrando em *loop*.[6]

Desta forma, substituiremos a regra $\supset L$ de **G3i** pelas quatro regras abaixo (veja [7]), obtendo assim o sistema **G3i'**:

$$\frac{P, B, \Gamma \vdash E}{P, P \supset B, \Gamma \vdash E}\ L0 \supset \qquad \frac{C \supset D \supset B, \Gamma \vdash E}{(C \wedge D) \supset B, \Gamma \vdash E}\ L\wedge \supset$$

$$\frac{C \supset B, D \supset B, \Gamma \vdash E}{(C \vee D) \supset B, \Gamma \vdash E}\ L\vee \supset \qquad \frac{C, D \supset B, \Gamma \vdash D \quad B, \Gamma \vdash E}{(C \supset D) \supset B, \Gamma \vdash E}\ L \supset\supset$$

onde P é atômico.

Podemos então facilmente especificar a lógica intuicionista:

```
prova G A :- membro A G.
prova G C :- membro false G.
prova G true.

prova G (A and B) :- prova G A, prova G B.
prova G (A imp B) :- prova (A::G) B.
```

[6] Observe que a mesma discussão serve para a regra $\forall L$ de **G3** e **G3i** e $\exists R$ de **G3**. Mas como PLLIC implementa apenas a parte proposicional das lógicas em questão, nos limitaremos a discutir o problema da implicação.

```
prova G (A or B) :- prova G A; prova G B.
prova G (neg A) :- prova (A::G) false.
prova G (forall A) :- pi x\(prova G (A x)).
prova G (exists A) :- prova G (A T).

prova G C :- memb_and_rest (A and B) G G1, prova (A ::(B :: G1)) C.
prova G C :- memb_and_rest (A or B) G G1, prova (A :: G1) C,
             prova (B :: G1) C.
prova G C :- memb_and_rest (neg A) G G1, prova G1 A.
prova G C :- memb (forall A) G, prova ((A T)::G) C.
prova G C :- memb_and_rest (exists A) G G1,
             pi x\(prova ((A x)::G1) C).

prova G E :- memb_and_rest (P imp B) G G1, atomico P, membro P G1,
             prova (B::G1) E.
prova G E :- memb_and_rest ((C and D) imp B) G G1,
             prova ((C imp (D imp B))::G1) E.
prova G E :- memb_and_rest ((C or D) imp B) G G1,
             prova ((C imp B)::(D imp B)::G1) E.
prova G E :- memb_and_rest ((C imp D) imp B) G G1,
             prova (C::(D imp B)::G1) D, prova (B::G1) E.
```

5 Cut elimination

Talvez a regra lógica mais conhecida em cálculo de seqüentes seja a regra *Cut*:

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ (Cut)$$

Basicamente, essa regra formaliza o conceito de provas matemáticas utilizando lemas auxiliares. Ou seja, se podemos provar um lema A (e outros resultados Δ_1) a partir de um conjunto de hipóteses Γ_1 e, a partir de A (e possivelmente algumas outras hipóteses Γ_2) é possível provar outro conjunto de resultados (Δ_2), então podemos provar Δ_1, Δ_2 diretamente a partir de Γ_1, Γ_2.

Essa idéia é utilizada sempre em Matemática mas é também muito interessante sob o ponto de vista computacional, uma vez que a implementação de tal regra é feita "*bottom-up*". Ou seja, para tentar provar Δ_1, Δ_2 a partir de Γ_1, Γ_2, primeiro tentamos provar uma fórmula A (para uma certa fórmula desconhecida A), e a partir de A tentamos provar Δ_2. Isto significa que a fórmula lógica A deve ser "adivinhada" pelo programa de computador e isso representa um problema muito sério, uma vez que computadores não tem a "criatividade" necessária para adivinhar fórmulas.

Portanto, é muito importante dentro da área de programação lógica a possibilidade de se verificar se um sistema lógico possui a propriedade de *cut-elimination*, ou seja, checar se a regra *Cut* é, de fato, redundante e portanto pode ser eliminada.

Enquanto para Ciência da Computação a importância da propriedade de *cut-elimination* está relacionada com a viabilidade de implementações, para os matemáticos essa propriedade reforça o fato de que lemas são ferramentas úteis para organizar uma prova, mas completamente dispensáveis. Ou seja, toda

prova que utiliza a regra Cut pode ser substituída por uma onde Cut não está presente.

Checar se um sistema lógico possui a propriedade de *cut-elimination* é, em geral, um problema não trivial (veja, por exemplo, [26, 20, 21, 27]). Daremos apenas uma idéia da prova para o sistema **G**.

A prova de que a regra Cut pode ser eliminada do sistema **G** é feita através do método de exaustão aliado à indução estrutural. De maneira mais clara, tomamos como base a prova Π do seqüente $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$:

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \Gamma_1 \vdash \Delta_1, A & A, \Gamma_2 \vdash \Delta_2\end{array}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ Cut$$

tal que esta seja a aplicação da regra Cut mais perto das folhas[7]. Desta forma, Π_1 e Π_2 não possuem aplicações da regra Cut – são *livres de cut*. Olhamos agora para a última regra aplicada em Π_1 e Π_2:

i. se a última regra de Π_1 ou Π_2 não agir sobre A (dizemos que A não é *principal*), então podemos mover a regra Cut mais para cima na prova. Por exemplo:

$$\dfrac{\begin{array}{cc}\Pi_1 & \dfrac{\begin{array}{c}\Pi_2' \\ A, \Gamma_2, B \vdash C\end{array}}{A, \Gamma_2 \vdash C \supset B}\supset R \\ \Gamma_1 \vdash A & \end{array}}{\Gamma_1, \Gamma_2 \vdash B \supset C}\ Cut$$

pode ser substituída por

$$\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2' \\ \Gamma_1 \vdash A & A, \Gamma_2, B \vdash C\end{array}}{\Gamma_1, \Gamma_2, B \vdash C}\ Cut}{\Gamma_1, \Gamma_2 \vdash B \supset C}\supset R$$

Observe que agora a última aplicação da regra Cut está ainda mais próxima das folhas.

Suponhamos então que A é principal.

ii. Se a última regra de Π_1 ou Π_2 for o axioma $Inicial$ ou uma das regras $falseL$ ou $trueR$, o corte pode ser eliminado. De fato, considere o exemplo:

$$\dfrac{\begin{array}{cc}\dfrac{}{\Gamma_1 \vdash \Delta_1, A}\ Inicial & \begin{array}{c}\Pi_2 \\ A, \Gamma_2 \vdash \Delta_2\end{array}\end{array}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ Cut$$

Como terminamos com um axioma e A é principal, $A \in \Gamma_1$ e portanto a dedução anterior pode ser substituída por:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_2 \\ A, \Gamma_2 \vdash \Delta_2\end{array}}{A, \Gamma_2 \vdash \Delta_1, \Delta_2}\ weak\ R}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ weak\ L$$

[7] Existe um jeito formal de definir o que é *mais perto* através o conceito de altura de provas.

iii. Caso A seja principal e a última regra de Π_1 ou Π_2 não for uma das listadas em (ii.), analisa-se *todas* as possibilidades de aplicação de regras em Π_1 e Π_2. Daremos aqui apenas um exemplo quando $A = B \supset C$. A derivação:

$$\dfrac{\dfrac{\begin{matrix}\Pi_1' \\ \Gamma_1, B \vdash \Delta_1, C\end{matrix}}{\Gamma_1 \vdash \Delta_1, B \supset C} \supset R \quad \dfrac{\begin{matrix}\Pi_2' & \Pi_2'' \\ C, \Gamma_2' \vdash \Delta_2' & \Gamma_2'' \vdash B, \Delta_2''\end{matrix}}{B \supset C, \Gamma_2 \vdash \Delta_2} \supset L}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} \, Cut$$

onde $\Gamma_2' \cup \Gamma_2'' = \Gamma_2$ e $\Delta_2' \cup \Delta_2'' = \Delta_2$ pode ser substituída por

$$\dfrac{\dfrac{\begin{matrix}\Pi_2'' & \Pi_1' \\ \Gamma_2'' \vdash \Delta_2'', B & B, \Gamma_1 \vdash \Delta_1, C\end{matrix}}{\Gamma_1, \Gamma_2'' \vdash \Delta_1, \Delta_2'', C} \, Cut \quad \begin{matrix}\Pi_2' \\ C, \Gamma_2' \vdash \Delta_2'\end{matrix}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} \, Cut$$

Desta forma, aumentamos o número de cortes, mas diminuímos o *tamanho* dos mesmos. Ou seja, os cortes são sobre estruturas mais simples.

Desta forma, podemos eliminar os cortes que aparecem nas folhas da prova e substituímos cortes internos por cortes mais simples ou mais perto das folhas. Ou seja, sobram eventualmente apenas cortes sobre átomos, que são novamente movidos na direção das folhas e então eliminados.

Vale observar que, como uma primeira conseqüência imediata da propriedade de *cut-elimination*, as lógicas clássica e intuicionista de primeira ordem são **consistentes**. De fato, é fácil ver que não há prova possível para $false$: as únicas regras de inferência que poderiam ser usadas para provar o seqüente (que não a regra Cut)

$$\vdash false$$

seriam *contr* R e *weak* R e essas regras, isoladamente, não levam a uma prova.

Uma segunda conseqüencia interessante é que todos os seqüentes que ocorrem na prova de um sequente $\Gamma \rightarrow \Delta$ onde a regra *Cut* não é utilizada contêm fórmulas que são sub-fórmulas de uma fórmula em Γ ou em Δ. Isso é chamado de propriedade da *sub-fórmula*.

Finalmente, apesar de que os cortes podem ser eliminados de provas, existem poucos seqüentes matemáticos interesantes que possuam provas livres do uso da regra *Cut* que podem ser escritos ou armazenados na memória de um computador.

6 Lógica linear

Como visto na Seção 3, matemáticos começam de um conjunto de axiomas, provam alguns lemas e então os utilizam para provar teoremas. Algumas das provas utilizadas não são construtivas, e o uso da estratégia conhecida como *redução ao absurdo* é muito comum.

Uma vez que um lema é provado, ele pode ser usado de novo para provar outras proposições ou teoremas: um lema provado verdadeiro será verdadeiro para sempre. Portanto, matemáticos trabalham com a lógica clássica, a lógica da *verdade estável* e de *recursos e conclusões infinitos*.

Já a lógica intuicionista (Seção 4) joga fora essa noção de verdade absoluta e a veracidade de uma afirmativa depende da existência de uma prova (ou construção) da afirmativa.

Mas ainda, a lógica intuicionista é uma lógica de *infinitos recursos* – mas não infinitas conclusões, uma vez que permitir a prova de todos os resultados possíveis implica em permitir o princípio do terceiro excluído.

Agora, se imaginarmos a situação real de descrever uma máquina de vender refrigerantes, não é adequado usar uma lógica de recursos infinitos. Ou seja, se uma latinha de guaraná custa um real e tenho um real na minha carteira, posso comprar apenas uma latinha e, no fim do processo, vou estar sem dinheiro.

A Lógica linear (desenvolvida por Girard [10]) lida com situações como essa: é uma lógica de *recursos conscientes*. Em Lógica linear, afirmativas não podem ser livremente copiadas (*Contraction*) ou descartadas (*Weakening*), apenas em situações especiais, onde aparece um tipo muito particular de conectivos: os exponenciais "?" e "!". Intuitivamente, $!B$ significa que o recurso B pode ser usado tantas vezes quanto necessárias. De maneira dual, $?B$ indica a possibilidade de produção de uma quantidade infinita da conclusão B.

A implicação linear é representada pelo símbolo "$\multimap$" e o significado de $A \multimap B$ é:

consome-se A dando origem a B

Isto significa que, a partir do ponto em que B é produzido, o predicado A deixa de ser válido. A implicação intuicionista "$\Rightarrow$" então significa:

$$A \Rightarrow B \equiv !A \multimap B$$

ou seja, um predicado A implica B intuicionisticamente se e somente se existe uma quantidade infinita de A que linearmente implica B.

A ausência de *Contraction* e *Weakening* muda a natureza dos conectivos lógicos. De fato, a conjunção intuicionista (assim como a disjunção) é separada em dois conectivos diferentes. Portanto, existem duas maneiras distintas de formular a conjunção, correspondendo a dois conectivos distintos em Lógica Linear: o conectivo multiplicativo "$\otimes$" ($A \otimes B$ significa ambos A e B) e o aditivo "&" ($A \& B$ = escolha entre A e B). O mesmo para a disjunção: multiplicativo "$\parr$" ($A \parr B$ é igual a A paralelo a B) e aditivo "$\oplus$" ($A \oplus B$ significa ou A ou B).

Lógica Linear utiliza ainda os seguintes conectivos: $\bot$, e 1 para a versão multiplicativa de falso e verdadeiro respectivamente; 0, $\top$ para a versão aditiva desses conectivos; e $\forall$ e $\exists$ para quantificações universal e existencial. Veja a Figura 8 para o sistema de seqüentes da lógica linear.

6.1 Forum

Os conectivos da lógica linear podem ser classificados em *síncronos* e *assíncronos* [1], dependendo se a regra de introdução à direita para aquele conectivo depende ou não do seu contexto.

O *de Morgan dual* de um conectivo em uma dessas classes é um conectivo na outra classe.

Dada essa divisão de conectivos, Miller propôs em [18] a presentação *Forum* de lógica linear na qual fórmulas são construídas utilizando apenas os conectivos assíncronos, a saber: ?, $\parr$, $\bot$, &, $\top$, $\multimap$, e $\forall$, junto com a versão intuicionista da

Axioma e regras Cut

$$\frac{}{A \vdash A}\ initial \qquad \frac{\Gamma_1 \vdash \Delta_1, A \quad A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ Cut$$

$$\frac{\Gamma_1 \vdash \Delta_1, !\,A \quad (!\,A)^n, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ Cut! \qquad \frac{\Gamma_1 \vdash \Delta_1, (?\,A)^n \quad ?\,A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\ Cut?$$

Regras à direita

$$\frac{}{\Gamma \vdash \top, \Delta}\ \top R \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \bot\, R \qquad \frac{}{\vdash 1}\ 1R$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \,\&\, B, \Delta}\ \&R \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta}\ \parr R$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ \oplus R_1 \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ \oplus R_2$$

$$\frac{\Gamma_1 \vdash A, \Delta_1 \quad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash A \otimes B, \Delta_1, \Delta_2}\ \otimes R \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}\ \multimap R$$

$$\frac{\Gamma \vdash A[x/y], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall R \qquad \frac{\Gamma \vdash A[x/t], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists R$$

Regras à esquerda

$$\frac{}{0, \Gamma \vdash \Delta}\ 0L \qquad \frac{}{\bot \vdash}\ \bot\, L \qquad \frac{\Gamma \vdash \Delta}{1, \Gamma \vdash \Delta}\ 1L$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta}\ \&L_1 \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta}\ \&L_2$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta}\ \oplus L \qquad \frac{\Gamma_1, A \vdash \Delta_1 \quad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \parr B \vdash \Delta_1, \Delta_2}\ \parr L$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}\ \otimes L \qquad \frac{\Gamma_1 \vdash A, \Delta_1 \quad \Gamma_2, B \vdash \Delta_2}{\Gamma_1, \Gamma_2, A \multimap B \vdash \Delta_1, \Delta_2}\ \multimap L$$

$$\frac{\Gamma, A[x/t] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall L \qquad \frac{\Gamma, A[x/y] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists L$$

Exponenciais

$$\frac{\Gamma \vdash \Delta}{\Gamma, !\,A \vdash \Delta}\ !W \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, !\,A \vdash \Delta}\ !D \qquad \frac{\Gamma, !\,A, !\,A \vdash \Delta}{\Gamma, !\,A \vdash \Delta}\ !C \qquad \frac{!\,\Gamma \vdash A, ?\,\Delta}{!\,\Gamma \vdash !\,A, ?\,\Delta}\ !R$$

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash ?\,A, \Delta}\ ?W \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash ?\,A, \Delta}\ ?D \qquad \frac{\Gamma \vdash ?\,A, ?\,A, \Delta}{\Gamma \vdash ?\,A, \Delta}\ ?C \qquad \frac{!\,\Gamma, A \vdash ?\,\Delta}{!\,\Gamma, ?\,A \vdash ?\,\Delta}\ ?L$$

Figura 8: Cálculo LL para a lógica linear

implicação $B \Rightarrow C$ ($B \Rightarrow C$ denota $!\,B \multimap C$).[8] Os conectivos síncronos da lógica linear estão disponíveis implicitamente, uma vez que conectivos que aparecem no lado esquerdo do seqüente comportam-se de modo síncrono.

O objetivo de apresentar *Forum* é que este possui um algoritmo de prova de seqüentes: a busca por provas em seqüentes com conectivos assíncronos no sucedente corresponde à busca *goal directed*, enquanto que conectivos assíncronos no antecedente correspondem ao procedimento de *backchaining* sobre cláusulas de programas [17].

Ao mesmo tempo, Forum captura toda a lógica linear, uma vez que os conectivos que faltam podem ser definidos utilizando as seguintes equivalências lógicas:

$$B^{\perp} \equiv B \multimap \perp \qquad 0 \equiv \top \multimap \perp \qquad 1 \equiv \perp \multimap \perp \qquad \exists x.B \equiv (\forall x.B^{\perp})^{\perp}$$
$$!\,B \equiv (B \Rightarrow \perp) \multimap \perp \qquad B \oplus C \equiv (B^{\perp} \,\&\, C^{\perp})^{\perp} \qquad B \otimes C \equiv (B^{\perp} \parr C^{\perp})^{\perp}$$

O sistema de provas de Forum é dada na Figura 9.

Seqüentes em Forum possuem uma das formas

$$\Sigma\colon \Psi; \Delta \longrightarrow \Gamma; \Upsilon \quad \text{e} \quad \Sigma\colon \Psi; \Delta \xrightarrow{B} \Gamma; \Upsilon,$$

onde Σ é uma assinatura, Δ e Γ são multiconjuntos de fórmulas, Ψ e Υ são conjuntos de fórmulas, e B é uma fórmula. Todas as fórmulas nos seqüentes são compostas dos conectivos assíncronos listados anteriormente (juntamente com $\Rightarrow$). Os significados de tais seqüentes em lógica linear são

$$!\,\Psi, \Delta \vdash \Gamma, ?\,\Upsilon \quad \text{e} \quad !\,\Psi, \Delta, B \vdash \Gamma, ?\,\Upsilon,$$

respectivamente.

No sistema de provas da Figure 9, as regras à direita atuam apenas sobre seqüentes da forma $\Sigma\colon \Psi; \Delta \longrightarrow \Gamma; \Upsilon$. A variável sintática $\mathcal{A}$ na Figura 9 denota um multiconjunto de fórmulas atômicas. Regras à esquerda são aplicadas apenas à fórmula B, que é o label de $\Sigma\colon \Psi; \Delta \xrightarrow{B} \mathcal{A}; \Upsilon$.

6.2 Especificando a lógica linear

Epecificamos, na verdade, Forum. Como a busca por provas é *uniforme*, podemos sempre começar pelo sucedente, passando ao antecedente apenas quando temos apenas átomos do lado direido do seqüente.

```
rproof Psi Delta Atomos (top::Gama) Upsilon.

rproof Psi Delta Atomos ((B land C)::Gama) Upsilon :-
    rproof Psi Delta Atomos (B::Gama) Upsilon,
    rproof Psi Delta Atomos (C::Gama) Upsilon.
rproof Psi Delta Atomos ((B cimp C)::Gama) Upsilon :-
    rproof (B::Psi) Delta Atomos (C::Gama) Upsilon.
rproof Psi Delta Atomos (falso::Gama) Upsilon :-
    rproof Psi Delta Atomos Gama Upsilon.
rproof Psi Delta Atomos ((B lpar C)::Gama) Upsilon :-
```

[8] Utilizamos aqui o símbolo $\Rightarrow$ ao invés de $\supset$ por uma questão de didática: queremos separar completamente as lógicas dadas por **G** da lógica *LL*.

$$\frac{}{\Sigma\colon\Psi;\Delta\longrightarrow\top,\Gamma;\Upsilon}\ \top R$$

$$\frac{\Sigma\colon\Psi;\Delta\longrightarrow B,\Gamma;\Upsilon\quad\Sigma\colon\Psi;\Delta\longrightarrow C,\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow B\,\&\,C,\Gamma;\Upsilon}\ \&\,R$$

$$\frac{\Sigma\colon\Psi;\Delta\longrightarrow\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow\bot,\Gamma;\Upsilon}\ \bot R\qquad\frac{\Sigma\colon\Psi;\Delta\longrightarrow B,C,\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow B\otimes C,\Gamma;\Upsilon}\ \otimes R$$

$$\frac{\Sigma\colon\Psi;B,\Delta\longrightarrow C,\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow B\multimap C,\Gamma;\Upsilon}\ \multimap R\qquad\frac{\Sigma\colon B,\Psi;\Delta\longrightarrow C,\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow B\Rightarrow C,\Gamma;\Upsilon}\ \Rightarrow R$$

$$\frac{y\colon\tau,\Sigma\colon\Psi;\Delta\longrightarrow B[y/x],\Gamma;\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow\forall_\tau x.B,\Gamma;\Upsilon}\ \forall R\qquad\frac{\Sigma\colon\Psi;\Delta\longrightarrow\Gamma;B,\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow ?\,B,\Gamma;\Upsilon}\ ?\,R$$

$$\frac{\Sigma\colon B,\Psi;\Delta\xrightarrow{B}\mathcal{A};\Upsilon}{\Sigma\colon B,\Psi;\Delta\longrightarrow\mathcal{A};\Upsilon}\ decide\,!\qquad\frac{\Sigma\colon\Psi;\Delta\longrightarrow\mathcal{A},B;B,\Upsilon}{\Sigma\colon\Psi;\Delta\longrightarrow\mathcal{A};B,\Upsilon}\ decide\,?$$

$$\frac{\Sigma\colon\Psi;\Delta\xrightarrow{B}\mathcal{A};\Upsilon}{\Sigma\colon\Psi;B,\Delta\longrightarrow\mathcal{A};\Upsilon}\ decide$$

$$\frac{}{\Sigma\colon\Psi;\cdot\xrightarrow{A}A;\Upsilon}\ initial\qquad\frac{}{\Sigma\colon\Psi;\cdot\xrightarrow{A}\cdot;A,\Upsilon}\ initial?$$

$$\frac{}{\Sigma\colon\Psi;\cdot\xrightarrow{\bot}\cdot;\Upsilon}\ \bot L\qquad\frac{\Sigma\colon\Psi;\Delta\xrightarrow{B_i}\mathcal{A};\Upsilon}{\Sigma\colon\Psi;\Delta\xrightarrow{B_1\&B_2}\mathcal{A};\Upsilon}\ \&\,L_i\qquad\frac{\Sigma\colon\Psi;B\longrightarrow\cdot;\Upsilon}{\Sigma\colon\Psi;\cdot\xrightarrow{?\,B}\cdot;\Upsilon}\ ?\,\mathrm{L}$$

$$\frac{\Sigma\colon\Psi;\Delta_1\xrightarrow{B}\mathcal{A}_1;\Upsilon\quad\Sigma\colon\Psi;\Delta_2\xrightarrow{C}\mathcal{A}_2;\Upsilon}{\Sigma\colon\Psi;\Delta_1,\Delta_2\xrightarrow{B\otimes C}\mathcal{A}_1,\mathcal{A}_2;\Upsilon}\ \otimes L\qquad\frac{\Sigma\colon\Psi;\Delta\xrightarrow{B[t/x]}\mathcal{A};\Upsilon}{\Sigma\colon\Psi;\Delta\xrightarrow{\forall_\tau x.B}\mathcal{A};\Upsilon}\ \forall L$$

$$\frac{\Sigma\colon\Psi;\Delta_1\longrightarrow\mathcal{A}_1,B;\Upsilon\quad\Sigma\colon\Psi;\Delta_2\xrightarrow{C}\mathcal{A}_2;\Upsilon}{\Sigma\colon\Psi;\Delta_1,\Delta_2\xrightarrow{B\multimap C}\mathcal{A}_1,\mathcal{A}_2;\Upsilon}\ \multimap L$$

$$\frac{\Sigma\colon\Psi;\cdot\longrightarrow B;\Upsilon\quad\Sigma\colon\Psi;\Delta\xrightarrow{C}\mathcal{A};\Upsilon}{\Sigma\colon\Psi;\Delta\xrightarrow{B\Rightarrow C}\mathcal{A};\Upsilon}\ \Rightarrow L$$

Figura 9: Sistema de provas de Forum

```
    rproof Psi Delta Atomos (B::C::Gama) Upsilon.
rproof Psi Delta Atomos ((B limp C)::Gama) Upsilon :-
    rproof Psi (B::Delta) Atomos (C::Gama) Upsilon.
rproof Psi Delta Atomos ((? B)::Gama) Upsilon :-
    rproof Psi Delta Atomos Gama (B::Upsilon).


rproof Psi Delta Atomos (A::Gama) Upsilon:-
    atomic A, rproof Psi Delta (A::Atomos) Gama Upsilon.
rproof Psi Delta Atomos nil Upsilon:- memb_and_rest B Delta Delta1,
    lproof Psi Delta1 B Atomos Upsilon; memb B Psi,
    lproof Psi Delta B Atomos Upsilon; memb B Upsilon,
    rproof Psi Delta Atomos (B::nil) Upsilon.
```

```
lproof Psi nil A nil (A::Upsilon).
lproof Psi nil A (A::nil) Upsilon.
lproof Psi nil falso nil Upsilon.

lproof Psi Delta (B land C) Atomos Upsilon :-
    lproof Psi Delta B Atomos Upsilon; lproof Psi Delta C Atomos Upsilon.
lproof Psi Delta (B cimp C) Atomos Upsilon :-
    rproof Psi nil nil (B::nil) Upsilon, lproof Psi Delta C Atomos Upsilon.
lproof Psi nil (? B) nil Upsilon :-
    rproof Psi (B::nil) nil nil Upsilon.
lproof Psi Delta (B lpar C) Atomos Upsilon:-
    split Delta Delta1 Delta2, split Atomos Atomos1 Atomos2,
    lproof Psi Delta1 B Atomos1 Upsilon,
    lproof Psi Delta2 C Atomos2 Upsilon.
lproof Psi Delta (B limp C) Atomos Upsilon:-
    split Delta Delta1 Delta2, split Atomos Atomos1 Atomos2,
    rproof Psi Delta1 (B::Atomos1) Upsilon,
    lproof Psi Delta2 C Atomos2 Upsilon.

prova Psi Delta Gama Upsilon :- rproof Psi Delta nil Gama Upsilon.
```

7 Aplicações: problemas lógicos

Podemos utilizar PLLIC para especificar probleminhas lógicos, como os que usualmente aparecem na literatura.

Exemplo 7 *Consideremos o seguinte exemplo, retirado de [36]:*

– Que tal perparar-nos umas tortas saborosas? – perguntou o Rei de Copas à Rainha de Copas.

– Como posso fazer tortas sem farinha? – perguntou a Rainha.

– Você está querendo dizer que a farinha foi roubada? – gritou o Rei.

– Foi! – respondeu a Rainha. Encontre o patife e corte-lhe a cabeça!

– Ora, ora – disse o Rei – não vamos ser precipitados!

Mesmo assim, era preciso encontrar a farinha. E, com efeito, ela foi encontrada na casa da Lebre de Março, do Chapeleiro Louco e do Leirão, de modo que os três foram prontamente detidos e julgados.

No julgamento, a Lebre de Março declarou que o Chapeleiro a havia roubado. O Chapeleiro e o Leirão também fizeram declarações mas, por alguma razão, essas declarações não foram anotadas. Se constatou que apenas um dos três havia roubado a farinha, e era o único dos três que dizia a verdade. Quem roubou a farinha?

Podemos especificar o problema acima da seguinte forma:

```
kind nome             type.
type lm,cl,lr         pessoa.
type roubou,verdade   pessoa -> form.
type exemplo          nome -> form -> o.
type entail           form -> form -> o.
type roubo_tortas     nome.
```

```
exemplo roubo_tortas (((roubou lm) or (roubou cl) or (roubou lr)) and
    ((roubou lm) imp ((neg (roubou cl)) and (neg roubou lr))) and
    ((roubou cl) imp ((neg (roubou lm)) and (neg roubou lr))) and
    ((roubou lr) imp ((neg (roubou cl)) and (neg roubou lm))) and
    ((verdade lm) imp (roubou lm)) and
    ((verdade cl) imp (roubou cl)) and
    ((verdade lr) imp (roubou lr)) and
    ((verdade lm) imp (roubou cl))).

entail X Y :- exemplo roubo_tortas X, prova (X::nil) (Y::nil).

%Pergunta-se: entail X (roubou lr).
```

A Programas utilizados no PLLIC em λ-Prolog

```
module conectivos.

nao_atomico (A imp B).
nao_atomico (A and B).
nao_atomico (A or B).
nao_atomico false.
nao_atomico true.
nao_atomico (neg A).
nao_atomico one.
nao_atomico zero.
nao_atomico (A land B).
nao_atomico (A cimp B).
nao_atomico (A limp B).
nao_atomico (A lpar B).
nao_atomico (? A).
nao_atomico (bang A).
nao_atomico (A multand B).
nao_atomico (A addor B).

atomico B :- not (nao_atomico B).

expand zero (neg true).
expand one (neg false).
expand (bang A) (neg (A cimp false)).
expand (A addor B) (neg ((neg A) land (neg B))).
expand (A multand B) (neg ((neg A) lpar (neg B))).
expand (A limp B)  (A1 limp B1) :- expand A A1, expand B B1.
expand (A land B)  (A1 land B1) :- expand A A1, expand B B1.
expand (A lpar B)  (A1 lpar B1) :- expand A A1, expand B B1.
expand (? A)  (? A1) :- expand A A1.
expand (neg A)  (neg A1) :- expand A A1.
expand A A.
```

Figura 10: Módulo conectivos

```
sig conectivos.

kind form type.

type imp              form -> form -> form.    % implicacao intuicionista/classica
type and              form -> form -> form.    % conjuncao intuicionista/classica
type or               form -> form -> form.    % disjuncao intuicionista/classica
type false            form.                    % falso intuicionista/classico/ll
type true             form.                    % verdadeiro intuicionista/classico/ll
type neg              form -> form.            % negacao intuicionista/classica/linear
type limp             form -> form -> form.    % implicacao linear
type cimp             form -> form -> form.    % implicacao intuitionistica
type land             form -> form -> form.    % conjuncao linear aditiva
type lpar             form -> form -> form.    % disjuncao linear multiplicativa
type ?                form -> form.            % modal ?
type bang        form -> form. % modal !
type multand        form -> form -> form.    % conjuncao linear multiplicativa
type addor        form -> form -> form.    % disjuncao linear aditiva
type one              form.                    % verdadeiro linear aditivo
type zero             form.                    % falso linear aditivo

infixr  and   120.
infixr  or    120.
infixr  imp   110.
infixr land 2.
infixr multand 2.
infixr lpar 3.
infixr addor 3.
infixr limp 1.
infixr cimp 4.

type nao_atomico       form -> o.
type atomico           form -> o.
type expand        form -> form -> o.
```

Figura 11: Assinatura conectivos

```
module listas.

id nil nil.
id (X::L) (X::K) :- id L K.

membro X (X::L).
membro X (Y::L) :- membro X L.

append nil K K.
append (X::L) K (X::M) :- append L K M.

memb_and_rest X (X::L) L.
memb_and_rest X (Y::K) (Y::L) :- memb_and_rest X K L.

split nil nil nil.
split (X::L) (X::K) M :- split L K M.
split (X::L) K (X::M) :- split L K M.
```

Figura 12: Modulo listas

```
sig listas.

type id                list A -> list A -> o.
type membro            A -> list A -> o.
type append            list A -> list A -> list A -> o.
type memb_and_rest     A -> list A -> list A -> o.
type split        list A -> list A -> list A -> o.
```

Figura 13: Assinatura listas

Referências

[1] Andreoli, JM., *Logic programming with focusing proofs in linear logic*, Journal of Logic and Computation, vol. 2, no. 3, pp. 297–347 (1992).

[2] Appel, K., Haken, W. and Koch, J., *Every Planar map is Four Colorable*, Illinois: Journal of Mathematics, vol.21, pp. 439–567 (1977).

[3] Armstrong, E., et al, *The J2EE 1.4 Tutorial*, Sun Microsystems (2005).

[4] Barendregt, H.P., *The Lambda Calculus: its syntax and semantics*, N.103 in Studies in Logic and the Foundations of Mathematics (revised edition), North-Holland, Amsterdam (1994).

[5] Coq proof assistant, http://coq.inria.fr/.

[6] Church, A., *A formulation of the simple theory of types*, Journal of Symbolic Logic 5, pp. 56–68 (1940).

[7] Dyckhoff, R. *Contraction-free sequent calculi for intuitionistic logic*, The Journal of Symbolic Logic, vol. 7, pp. 795–807 (1992).

[8] A. Felty and D. Miller Specifying theorem provers in a higher-order logic programming language, *Ninth International Conference on Automated Deduction*, 1988.

[9] Gentzen, G., *Investigations into logical deductions*, The Collected Papers of Gerhard Gentzen, North-Holland Publishing Co., Amsterdam (1969).

[10] Girard, J-Y., *Linear Logic*, Theoretical Computer Science, vol 50, pp. 1–102 (1987).

[11] Girard, J-Y., *Proofs and types*.

[12] Gödel, K. *On Undecidable Propositions of Formal Mathematical Systems*, New York (1934).

[13] Hatcher, W. S. *Foundations of Mathematics* (1968).

[14] van Heijenoort, J., *From Frege to Gödel*, Harvard College, (1999).

[15] Hilbert, D., Ackermann, W., *Grundzüge der Theoretischen Logik*, Berlin, Springer (1928).

[16] Isabelle, http://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html.

[17] Miller, D., Nadathur, G., Pfenning, F., and Scedrov, A., *Uniform proofs as a foundation for logic programming*, Annals of Pure and Applied Logic, vol.51 (1991).

[18] Miller, D., *Forum: A multiple-conclusion specification language*, Theoretical Computer Science, vol.165, pp. 201–232 (1996).

[19] Miller, D. *Sequent Calculus and the Specification of Computation*, School Marktoberdorf on Logic of Computation in 1997: An Advanced Study Institute of the NATO Science Committee and the Technical University of Munich, Germany (1997).

[20] Miller, D., Pimentel, E. *Using linear logic to reason about sequent systems*, Lecture Notes in Computer Science, v. 2381, pp. 2–23 (2002).

[21] Miller, D., Pimentel, E. *Linear logic as a framework for specifying sequent calculus*, Lecture Notes in Logic 17, pp. 111–135 (2004).

[22] G. Nadathur and D. Miller. An Overview of λ-Prolog. In *Fifth International Logic Programming Conference*, pp. 810–827, August 1988. MIT Press.

[23] Nadathur, G., *Teyjus: Language Manual*, http://teyjus.cs.umn.edu/language/teyjus_toc.html.

[24] Objective Calm, http://caml.inria.fr/ocaml/.

[25] Negri, S. and von Plato, J., *Structural proof theory*, Cambridge University Press (2001).

[26] Pfenning, F. *Logical frameworks*. In Alan Robinson and Andrei Voronkov, editors, Handbook of Automated Reasoning, chapter 17, pp. 1063–1147 (2001).

[27] Pimentel, E., Miller, D., *On the specification of sequent systems*, Lecture Notes in Computer Science, v. 3835, pp. 352 – 366 (2005).

[28] Pimentel, E., *Fundamentos de Matemática*, disponível em http://www.mat.ufmg.br/~elaine/fundmat.pdf (2006).

[29] Pimentel, E., *PLLIC - fundamentação teórica*, disponível em http://www.mat.ufmg.br/~elaine/PLLIC_teoria.pdf (2007).

[30] Prawitz, D., *Natural Deduction: a proof-theoretical study*, Dover Publications (2006).

[31] PVS Specification and Verification System, http://pvs.csl.sri.com.

[32] Ronchi Della Rocca S., Paolini L., *The Parametric λ-calculus: a metamodel for computation*, Computer Science-Monograph, Springer Verlag (2004).

[33] Robertson, N., Sanders, D., Seymour, P., and Thomas, R., *Efficiently four-coloring planar graphs*, ACM Press (1996).

[34] Russell, B. and Whitehead, A. N., *Principia Mathematica*, New York, Cambridge University Press (1927).

[35] Smullyan, R. M., *Gödel's incompletness theorems*, New York, Oxford University Press (1992).

[36] Smullyan, R. M., *Alice no país dos enigmas*, Rio de Janeiro, Jorge Zahar Editor (2000).

[37] Sørensen, M. H. and Urzyczyn, P. *Lectures on the Curry-Howard isomorphism* (1998).

[38] Troelstra, A. S., *Lectures on Linear Logic*, CSLI (1992).

[39] Troelstra, A. S. and Schiwichtenberg, *Basic Proof Theory*, Cambridge University Press (2000).

[40] Turing, A.M. *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, series 2, 42 pp , 230-265 (1936-37).