

Sebenta Prática de Elementos Lógicos da Programação I

(Guiões das sessões laboratoriais no sistema *Isabelle*)

Maria João Frade

mjf@di.uminho.pt

Departamento de Informática
Universidade do Minho
1997/98

Índice

1	Introdução	1
2	Aula 1 (Laboratorial) - O Sistema Hm	2
2.1	Sistema Dedutivo de Hilbert para a Lógica Proposicional Mínima	2
2.2	Provas por Propagação em Retrocesso	3
2.3	Provas com Hipóteses	4
3	Aula 2 (Laboratorial) - O Sistema Nm	5
3.1	Sistema de Dedução Natural para a Lógica Proposicional Mínima	5
3.2	Provas por Propagação em Retrocesso	5
3.3	Provas com Hipóteses	7
4	Aula 3 (Laboratorial) - O Sistema Hc	8
4.1	Sistema de Dedutivo de Hilbert para a Lógica Proposicional Clássica	8
4.2	Provas por Propagação em Retrocesso	8
4.3	Questões de Avaliação Prática	11
5	Aula 4 (Laboratorial) - Os Sistemas Ni e Nc para a Lógica Proposicional (duas versões)	12
5.1	Sistemas de Dedução Natural para a Lógica Proposicional (Intuicionista e Clássica)	12
5.2	Provas por Propagação em Retrocesso	13
5.3	Questões de Avaliação Prática	15
6	Aula 5 (Laboratorial) - O Sistema Nc	16
6.1	Sistema de Dedução Natural para a Lógica de Primeira Ordem Clássica	16
6.2	Provas por Propagação em Retrocesso	17
6.3	Questões de Avaliação Prática	19
7	Aula 6 (Laboratorial) - O Sistema G1	20
7.1	Sistema Dedutivo de Gentzen para a Lógica de Primeira Ordem Clássica	20
7.2	Provas por Propagação em Retrocesso	22
7.3	Questões de Avaliação Prática	23
8	Aulas 7 e 8 - Relação entre G1 e Nc e Eliminação do Corte	24
8.1	Regra do Corte. Relação de G1 com Nc. Eliminação do Corte.	24
8.1.1	Exercícios	24
9	Aula 9 (Laboratorial) - Os Sistemas G1 e GS	25
9.1	O Sistema Dedutivo G1	25
9.2	Questões de Avaliação Prática	26
9.3	O Sistema GS de <i>Gentzen-Schutte</i> (ou <i>Gentzen Simétrico</i>)	27
9.4	Provas por Propagação em Retrocesso	27
10	Aula 10 - Resolução e Unificação	29
10.1	Resolução e Unificação	29
10.1.1	Exercícios	30

11 Aula 11 (Laboratorial) - O Sistema S4	31
11.1 O Sistema Dedutivo de Gentzen para a Lógica Modal	31
11.2 Provas por Propagação em Retrocesso	32
11.3 Questões de Avaliação Prática	34
12 Aula 12 (Laboratorial) - O Sistema S4	35
12.1 O Sistema Dedutivo S4	35
12.2 Questões de Avaliação Prática	36
A Exame Prático Laboratorial	38

1 Introdução

Este documento é o resultado da compilação dos guiões das aulas práticas da disciplina de Elementos Lógicos da Programação I, leccionada ao 2^o ano da Licenciatura em Matemática e Ciências da Computação, no ano lectivo de 1997/98, e serve de complemento à sebenta teórica da disciplina [1].

Os assuntos tratados nesta cadeira incidem essencialmente nos aspectos dedutivos do estudo da lógica. As aulas práticas foram na sua maioria laboratoriais (em sistema Unix), usando um sistema de prova assistida - o sistema *Isabelle* - que nos permitiu descrever todas os sistemas dedutivos estudados.

As sessões laboratoriais têm uma duração de duas horas, e foram acompanhadas por guiões, com a seguinte estrutura: apresentação do sistema dedutivo a usar, exercícios de aplicação e, periodicamente, questões de avaliação prática.

O *Isabelle* [2, 3, 4, 5] é um sistema de prova assistida generalista, que permite descrever e raciocinar sobre vários sistemas lógicos. Está escrito em Standard ML e o utilizador interage directamente com o interpretador. Este sistema é muito vasto, mas o utilizador inexperiente pode trabalhar, sem dificuldades, com um pequeno repertório de comandos e noções básicas.

Ao longo das várias sessões laboratoriais vai sendo feita uma apresentação gradual dos comandos e características do sistema *Isabelle*, com base em exemplos e casos de estudo. No entanto, o objectivo não é o de fazer uma apresentação exaustiva do sistema. A sua utilização é limitada às necessidades desta disciplina.

2 Aula 1 (Laboratorial) - O Sistema Hm

Nesta primeira sessão laboratorial será feita uma primeira apresentação do sistema Isabelle¹, usando como caso de estudo o *Sistema Dedutivo de Hilbert para a Lógica Proposicional Mínima (Hm)*. O Isabelle utiliza por base a linguagem ML, não sendo contudo imprescindível o conhecimento da linguagem para a utilização do sistema.

2.1 Sistema Dedutivo de Hilbert para a Lógica Proposicional Mínima

Vamos usar os ficheiros **Hm.thy** e **Hm.ML** que pode copiar de `~mjf/ELP1/` para a sua área de trabalho.

Em **Hm.thy** está definida a teoria para o sistema **Hm**. Dos vários aspectos desta definição vamos, por agora, realçar:

```
Hm = Pure +
[...]
consts
[...]
"-->"          :: [o, o] => o          (infixr 25)
rules
  ax1  "A --> (B --> A)"
  ax2  "(A -->B -->C) --> (A-->B) --> A --> C"
  mp   "[| A-->B ; A |] ==> B"
end
```

chamando atenção para o facto do símbolo `-->` estar a ser usado para a implicação da lógica objecto em vez de \supset , sendo associativo à direita.

Comecemos por entrar no sistema Isabelle. Por agora vamos chamá-lo na sua forma mais simples (*Pure*), isto é, desprovido de qualquer lógica objecto: apenas equipado com a meta-lógica.

```
isabelle Pure
```

Vamos então agora carregar a teoria **Hm.thy**.

```
> use_thy "Hm";
```

Se tudo correr bem, após carregar a teoria **Hm**, o sistema tenta ler o ficheiro **Hm.ML** se ele existir. Este ficheiro começa tipicamente com a declaração ML `open Hm`; e poderá conter o código ML que se quiser.

Neste momento tem à sua disposição o sistema dedutivo **Hm**.

Experimente agora alguns comandos que lhe permitem ver as regras lógicas:

```
> prth ax1;
> prth mp;
> prths [ax1, ax2, mp];
```

¹Certifique-se que `/usr2/Isabelle94-8/bin` e `/usr/local/sml/bin` se encontram na sua `PATH`. Caso tal não se verifique inclua o comando `export PATH=${PATH}:/usr2/Isabelle94-8/bin:/usr/local/sml/bin` no fim do seu ficheiro `~/.profile` ou `~/.bashrc`.

Com certeza reparou que os símbolos proposicionais aparecem precedidos por ponto de interrogação. Essas entidades são variáveis livres a que o sistema Isabelle chama “*unknowns*” ou “*schematic variables*” e que estão envolvidas no processo de unificação. Na realidade, os axiomas e regras definidas são apenas esquemas.

2.2 Provas por Propagação em Retrocesso

Vamos então provar que $A \supset A$. Para lançar a prova utiliza-se o comando `goal`.

```
> goal Hm.thy "A --> A";
```

Repare que o Isabelle numera as premissas que faltam provar.

Para avançar com a prova em retrocesso pode usar-se o comando `by` aplicado a uma tática de prova. A tática aqui a usar é a resolução da regra `mp` com a premissa 1 (para já a única): (`resolve_tac [mp] 1`). Faça então:

```
> by (resolve_tac [mp] 1);
```

Existe uma abreviatura deste comando que é `br mp 1`; . Vamos então desfazer este passo de prova e usar a abreviatura

```
> undo();  
> br mp 1;
```

Digite então a seguinte sequência de comandos que corresponde à estratégia de prova usada para a resolução deste problema, feita já na aula teórica. Analise com cuidado cada passo da prova.

```
> br mp 1;  
> br ax2 1;  
> br ax1 1;  
> br ax1 1;
```

No final o sistema deve responder `No subgoals!` indicando que já terminou a prova.

O comando `result()` retorna o teorema que se acabou de provar.

```
> prth(result());
```

Se quisermos guardar este resultado, podemos associá-lo a um nome (por exemplo, `teo1`) fazendo

```
> val teo1 = result();
```

Daqui para a frente podemos utilizar este resultado usando `teo1`.

Lancemos então outra prova. Acompanhe o seu desenvolvimento escrevendo no papel a sua árvore de dedução.

```
> goal Hm.thy "B --> P --> P";  
  
> br mp 1;  
> br ax1 1;  
> br teo1 1;
```

Repare que se utilizou o resultado já provado `teo1`.

Exercício:

Faça agora, por si, a prova de $Q \supset (P \supset Q \supset P)$.

Com certeza já reparou que cada estado da prova é etiquetado por `Level n`. Se quiser recuar directamente para um nível anterior da prova pode usar o comando `choplev` com o nível para onde pretende ir.

2.3 Provas com Hipóteses

Ao lançar uma prova com hipóteses, convém atribuir a estas um nome. Vamos então lançar a prova $\{A \supset B, B \supset C\} \vdash A \supset C$

```
> val [h1, h2] = goal Hm.thy "[| A --> B; B --> C |] ==> A --> C";
```

Repare que as hipóteses são colocadas entre `[| ... |]` e separadas por `;`, denotando `==>` a meta-implicação. Através de `val [h1, h2] = ...` atribuímos um nome a cada uma das hipóteses.

A prova processa-se normalmente, agora com a característica adicional de se poder utilizar a resolução com `h1` e `h2` como táticas de prova.

```
> br mp 1;
> br h1 2;
> br mp 1;
> br ax2 1;
> br mp 1;
> br ax1 1;
> br h2 1;
```

Exercício:

Atente na seguinte árvore de dedução para $\{A, B \supset A \supset C\} \vdash B \supset C$

$$\frac{\frac{\frac{(B \supset A \supset C) \supset (B \supset A) \supset (B \supset C)}{ax2} \quad \frac{B \supset A \supset C}{mp} \quad \frac{A \supset (B \supset A)}{ax1} \quad \frac{A}{h1}}{(B \supset A)}}{mp} \quad B \supset C$$

Desenvolva em Isabelle esta prova.

Para sair do sistema faça

```
> quit();
```

3 Aula 2 (Laboratorial) - O Sistema Nm

Nesta sessão laboratorial usaremos o Isabelle para trabalhar no *Sistema de Dedução Natural para a Lógica Proposicional Mínima (Nm)*.

3.1 Sistema de Dedução Natural para a Lógica Proposicional Mínima

Vamos usar os ficheiros `Nm.thy` e `Nm.ML` que pode copiar de `~mjf/ELP1/` para a sua área de trabalho. Em `Nm.thy` está definida a teoria para o sistema `Nm`. Dos vários aspectos desta definição só nos interessa, por agora, realçar:

```
Nm = Pure +
[...]
consts
[...]
"-->"      :: [o, o] => o                (infixr 25)
rules
  impI      "[| A ==> B |] ==> A-->B"
  impE      "[| A-->B ; A |] ==> B"
end
```

Sob o ponto de vista da nomenclatura, todas as regras de um sistema de dedução natural assumem uma de duas designações: *regras de introdução* - se introduz na conclusão uma conectiva; e *regras de eliminação* - se elimina da conclusão uma conectiva. Em `Nm`, temos apenas a conectiva `-->`, com as regras que lhe estão associadas.

Começemos por entrar no sistema e carregar a esta teoria.

```
isabelle Pure
> use_thy "Nm";
```

3.2 Provas por Propagação em Retrocesso

Apesar de não ser particularmente vocacionada para o efeito, a dedução natural pode construir provas por propagação em retrocesso.

Vamos começar por provar que $A \supset (B \supset A)$, fazendo a prova por propagação em retrocesso.

```
> goal Nm.thy "A --> (B --> A)";
```

A conclusão da árvore de dedução não pode ter hipóteses abertas. Por isso, neste ponto da prova o conjunto de hipóteses abertas é vazio. No entanto, nos nodos anteriores da construção da prova o conjunto de hipóteses abertas pode não ser vazio. É o que acontece ao aplicarmos a regra de introdução de implicação:

```
> br impI 1;
> br impI 1;
```

Repare que as fórmulas `A` e `B` surgem como hipóteses abertas.

Do ponto de vista da árvore de dedução, a utilização desta regra (`impI`) nestes dois pontos da prova correspondem aos momentos em que se faz o cancelamento das hipóteses; neste caso de `A` e `B`.

Finalmente, o último passo é feito pela regra da assunção que introduz uma árvore de dedução com conclusão `A`, partindo de um conjunto de hipóteses abertas que contém `A`. Para isso, faça


```
> by (assume_tac 1);
```

Existe uma abreviatura deste comando que é `ba 1`; Para o experimentar pode fazer

```
> undo();
> ba 1;
```

Lancemos então outra prova. Acompanhe o seu desenvolvimento escrevendo no papel a sua árvore de dedução.

```
> goal Nm.thy "(A-->B-->C) --> B --> (A-->C)";
> br impI 1;
> br impI 1;
> br impI 1;
> br impE 1;
> br impE 1;
> ba 1;
> ba 1;
> ba 1;
```

Repare que a sequência de construção da árvore de dedução não tem de ser necessariamente esta. Voltemos ao nível 5 da árvore para aplicar outra estratégia.

```
> choplev 5;
> ba 2;
```

Ao aplicar a assunção o sistema tenta usar as hipóteses em aberto pela ordem em que elas surgem na lista de hipóteses. Assim, ao aplicar a assunção na premissa 2 o sistema unifica a variável livre que aparece na conclusão com a hipótese $A \rightarrow B \rightarrow C$, o que não é útil para a evolução da prova, neste caso.

Para tentar outra unificação e, conseqüentemente, uma aplicação alternativa da assunção, faça

```
> back();
```

Repare que o nível da prova não se altera! `back()` apenas dá como resultado o estado da prova que resulta de aplicar o método anterior de uma outra forma, caso isso seja possível.

Encontre agora uma estratégia para terminar a prova.

Exercício:

Faça agora, por si, a prova de $(A \rightarrow B) \rightarrow (B \rightarrow C) \rightarrow (A \rightarrow C)$

Mais dois comandos uteis: `pr()`; permite ver o estado actual da prova; e `prlev n`; dá a possibilidade de ver o estado da prova no nível `n` sem alterar o estado actual da prova.

3.3 Provas com Hipóteses

Ao lançar uma prova com hipóteses, convém atribuir a estas um nome. Vamos então lançar a prova $\{A \supset B, B \supset C\} \vdash A \supset C$

```
> val [h1, h2] = goal Nm.thy "[| A --> B ; B --> C |] ==> A --> C";
```

Repare que as hipóteses são colocadas entre `[| ... |]` e separadas por `;`, denotando `==>` a meta-implicação. Através de `val [h1, h2] = ...` atribuímos um nome a cada uma das hipóteses.

Analise agora com cuidado a seguinte “árvore de dedução”

```
> br impI 1;
> br impE 1;
> br h2 1;
> br impE 1;
> br h1 1;
> ba 1;
```

Se ao lançar uma prova com hipóteses se esquecer de lhes atribuir um nome, pode ainda fazê-lo no decorrer da prova. Por exemplo, faça

```
> goal Nm.thy "[| A --> B ; B --> C |] ==> A --> C";
> br impI 1;
> br impE 1;
```

O comando `premises()`; retorna a lista de hipóteses associada à prova corrente; experimente-o. Não se esqueça que para ver uma lista de fórmulas deve usar `prths`.

Para atribuir um nome a cada hipótese da lista, pode usar as facilidades do ML fazendo, por exemplo

```
> val [h1, h2] = premises();
```

Confirme a atribuição de nomes que foi feita. Agora pode concluir a prova.

Exercício:

Construa em `Nm` a prova de $\{A, B \supset A \supset C\} \vdash B \supset C$.

E para sair ...

```
> quit();
```

4 Aula 3 (Laboratorial) - O Sistema Hc

Nesta sessão laboratorial usaremos o Isabelle para trabalhar no *Sistema de Dedutivo de Hilbert para a Lógica Proposicional Clássica (Hc)*.

4.1 Sistema de Dedutivo de Hilbert para a Lógica Proposicional Clássica

Vamos usar os ficheiros **Hc.thy** e **Hc.ML** que pode copiar de `~mjf/ELP1/` para a sua área de trabalho. Em **Hc.thy** está definida a teoria para o sistema **Hc**. Dos vários aspectos desta definição só nos interessa, por agora, realçar:

```
Hc = Pure +
[...]
consts
[...]
  Not           :: o => o                ("~ _" [40] 40)
  "&"           :: [o, o] => o           (infixr 35)
  "|"           :: [o, o] => o           (infixr 30)
  "-->"        :: [o, o] => o           (infixr 25)
rules
  H1    "A --> (B --> A)"
  H2    "(A --> B --> C) --> (A --> B) --> A --> C"
  H3    "(~B --> ~A) --> A --> B"
  H4    "(B --> ~A) --> A --> ~B"
  mp    "[| A --> B ; A |] ==> B"
(* Definitions *)
  and_def    "A & B == ~(A --> ~B)"
  or_def     "A | B == ~A --> B"
end
```

Os símbolos \sim , $-->$, $\&$, e $|$, estão a ser usados para representar as conectivas \neg , \supset , \wedge e \vee , respetivamente. Repare que $\&$ e $|$ são definidos à custa de $-->$ e \sim , como abreviaturas. Isto é, $A \wedge B \doteq \neg(A \supset \neg B)$ e $A \vee B \doteq \neg A \supset B$. Repare ainda na prioridade atribuída às várias conectivas, o que evita a utilização excessiva de parêntesis.

Entre agora no sistema e carregue a teoria **Hc**. Pode ver as abreviaturas definidas para \wedge e \vee fazendo

```
> prths [and_def, or_def];
```

4.2 Provas por Propagação em Retrocesso

Começemos por demonstrar que $\vdash P \supset \neg\neg P$.

```
> goal Hc.thy "P --> ~~P";
```

A prova pode desenvolver-se através da seguinte sequência de comandos:

```
> br mp 1;
> br H4 1;
```

```
> br mp 1;
> br mp 1;
> br H2 1;
> br H1 1;
> br H1 1;
```

O teorema que acabamos de provar é um resultado importante que merece ser guardado para futuras utilizações (vamos dar-lhe o nome de `teo1`). Como já tínhamos visto, isto pode ser feito fazendo: `val teo1 = result()`; mas o Isabelle tem o comando `qed` que permite fazer a mesma coisa de um modo mais elegante. Faça então

```
> qed "teo1";
```

Repare, de novo, no nível 2 da árvore de dedução.

```
> choplev 2;
```

Com certeza já reparou que a premissa que está em aberto não é mais do que uma instância de um resultado mais genérico ($A \supset A$) que já foi demonstrado na primeira sessão laboratorial. Se quisermos assumir o teorema $A \supset A$ como válido (uma vez que já foi demonstrado), de modo a podermos utilizar esse resultado para facilitar esta prova, podemos fazer

```
> val teo2 = assume_ax Hc.thy "A --> A";
```

Ao fazer isto, a fórmula $A \supset A$ é assumida como sendo uma hipótese que pode ser usada. Note que ao acrescentar $A \supset A$ como hipótese não distorcemos o sentido da prova, uma vez que $A \supset A$ já era demonstrável a partir da teoria inicial.

Agora pode fechar a prova fazendo

```
> br teo2 1;
```

Vamos tentar guardar o este resultado fazendo

```
> qed "t";
```

Como deve ter reparado, o sistema retorna uma mensagem de erro, alertando para o facto do teorema ter sido provado com a ajuda de uma hipótese adicional. Se quisermos que o sistema não se preocupe com esse facto, e que portanto omita essas verificações, devemos usar o comando `uresult()`. Faça então

```
> bind_thm("t", uresult());
```

para associar este resultado ao teorema de nome `t`. Veja o que está guardado em `teo1` e em `t` e compare as diferenças.

Vamos agora demonstrar que a *Lei do Terceiro Excluído* é válida em **Hc** (ou seja, $\vdash \neg A \vee A$)

```
> goal Hc.thy "~A | A";
```

A disjunção aparece nesta teoria como uma abreviatura e não temos nenhuma regra a ela associada. Por isso, o que devemos fazer é reescrever a fórmula usando a definição da disjunção. Vamos então aplicar a seguinte tática

```
> by (rewrite_tac [or_def]);
```

Como é costume, existe uma abreviatura para este comando, que é `bw`. Para o experimentar pode fazer

```
> undo();  
> bw or_def;
```

Repare, pelo resultado obtido, como a lei que estamos a demonstrar é equivalente à *Lei da Dupla Negação*. Usando o resultado da prova anterior, esta prova faz-se muito facilmente.

```
> br mp 1;  
> br H3 1;  
> br teol 1;
```

Para guardar o resultado faça

```
> qed "lte";
```

O Isabelle dispõe de um comando que permite lançar uma prova e, simultaneamente, reescreve-la de acordo com a lista de definições fornecida. Esse comando é `goalw`. Experimente

```
> goalw Hc.thy [or_def] "~A | A";
```

e confronte o resultado com o que tinha obtido anteriormente, quando utilizou o comando `goal`.

Exercício:

Diga quais as fórmulas correspondentes (usando apenas as conectivas básicas \supset e \neg) às seguintes fórmulas:

(i) $(B \vee \neg A) \wedge (C \supset B \wedge \neg A)$

(ii) $(P \supset Q) \vee (\neg Q \wedge P)$

O Isabelle fornece também uma tática que permite reescrever uma fórmula, usando as definições em sentido contrário.

```
> goal Hc.thy "~~A -->A";  
> by (fold_tac [or_def, and_def]);
```

Exercício:

Veja como poderia reescrever as seguinte fórmulas usando conjunções e disjunções:

(i) $\neg(\neg(\neg A \supset B) \supset \neg C)$

(ii) $(\neg P \supset \neg Q) \supset \neg(\neg\neg P \supset \neg\neg Q)$

Exercício:

Demonstre em **Hc** que $\vdash B \supset A \vee B$.

4.3 Questões de Avaliação Prática

1. Demonstre em **Hc**:

(i) $\neg A \vdash A \supset B$

(ii) $\neg\neg P \supset \neg\neg Q \vdash P \supset Q$

5 Aula 4 (Laboratorial) - Os Sistemas Ni e Nc para a Lógica Proposicional (duas versões)

Nesta sessão laboratorial iremos trabalhar com duas versões dos *Sistemas de Dedução Natural para a Lógica Proposicional (Intuicionista e Clássica)*

Numa primeira versão, \neg e \supset são consideradas como conectivas primitivas, e \wedge e \vee como abreviaturas. Na segunda versão, \neg , \supset , \wedge e \vee , são consideradas conectivas primitivas.

5.1 Sistemas de Dedução Natural para a Lógica Proposicional (Intuicionista e Clássica)

Vamos usar os ficheiros `Ni_v1.thy`, `Nc_v1.thy`, `Ni_v2.thy`, `Nc_v2.thy` e `Ni_v1.ML`, `Nc_v1.ML`, `Ni_v2.ML`, `Nc_v2.ML` que deve copiar de `~mjf/ELP1/` para a sua área de trabalho.

Em `Ni_v1.thy` está definida a teoria para o sistema de dedução natural intuicionista, que considera \neg e \supset como conectivas primitivas e \wedge e \vee como abreviaturas.

Em `Nc_v1.thy` a teoria `Ni_v1` é enriquecida com a regra `classica`, obtendo-se assim um sistema de dedução natural para a Lógica Proposicional Clássica.

```
Ni_v1 = Pure +
[...]
consts
[...]
  Not           :: o => o                ("~ _" [40] 40)
  "&"           :: [o, o] => o            (infixr 35)
  "|"           :: [o, o] => o            (infixr 30)
  "-->"        :: [o, o] => o            (infixr 25)
rules
  impI          "(A ==> B) ==> A --> B"
  impE          "[| A --> B ; A |] ==> B"
  negI          "(!! B. A ==> B) ==> ~A"
  negE          "[| ~A ; A |] ==> B"
(* Definitions *)
  and_def       "A & B == ~(A --> ~B)"
  or_def        "A | B == ~A --> B"
end

Nc_v1 = Ni_v1 +
rules
  classica      "(~A ==> A) ==> A"
end
```

Repare na regra de introdução da negação, `negI`. O símbolo `!!` é a notação do Isabelle para \wedge que representa a quantificação universal ao meta-nível. Esta regra tem uma forma que se baseia numa técnica a que se chama *Skolemização* (*cf.* pag. 63 da sebenta teórica).

Entre agora no sistema e carregue a teoria `Ni_v1`. Para facilitar a escrita, pode associar lista de definições a um identificador (`defs`, neste caso) fazendo

```
> val defs = [and_def, or_def];
```

5.2 Provas por Propagação em Retrocesso

Começemos por demonstrar que $\vdash A \supset A \vee B$.

```
> goalw Ni_v1.thy defs "A --> A | B";
```

A prova pode desenvolver-se através da sequência de comandos abaixo apresentada. Construa no papel o quadro da prova.

```
> br impI 1;
> br impI 1;
> br negE 1;
> ba 1;
> ba 1;
```

Vamos agora provar em $Ni_v1 \vdash A \supset B \supset A \wedge B$. Faça o quadro da prova.

```
> goalw Ni_v1.thy defs "A --> B --> A & B";
> br impI 1;
> br impI 1;
> br negI 1;
```

Repare no estado actual da prova. A premissa em aberto, é uma representação finita de um número infinito de premissas. Faça agora

```
> br negE 1;
> ba 2;
> back();
```

Como deve ter reparado, ao fazer a resolução com `negE` surgem variáveis livres da forma $?A3(Ba)$, indicando que $?A3$ é uma variável livre que pode conter Ba como subfórmula. Esse facto, resulta de dois aspectos:

- quando é feita a unificação da premissa 1, Ba , com a regra `negE` a meta-variável A não é instanciada;
- a resolução é feita no âmbito da quantificação universal de Ba .

A prova termina agora facilmente, fazendo

```
> br impE 1;
> ba 1;
> ba 1;
```

Repare, de novo, no nível 3 da árvore de dedução.

```
> choplev 3;
```

Se, ao fazer a resolução com `negE`, quisermos forçar algumas instanciações, podemos fazê-lo através da tática `res_inst_tac` indicando a lista de instanciações. Cada instanciação é representada por um par com o nome da meta-variável a instanciar e a respectiva instanciação.

No nosso caso, queremos que a meta-variável A da regra `negE` seja instanciada com a fórmula B . Por isso, devemos fazer

```
> by (res_inst_tac [("A","B")] negE 1);
```


Agora já deve saber terminar a prova ...

Carregue agora, para o sistema, a versão clássica desta teoria. Esta teoria só difere da anterior por termos também à nossa disposição a regra `classica`. É claro que tudo o que é demonstrável em `Ni_v1` também o é em `Nc_v1`.

```
> use_thy "Nc_v1";
```

Demonstremos agora, nesta teoria, que $\vdash (\neg A \supset C) \supset (A \supset C) \supset C$. siga o desenrolar da demonstração, escrevendo o quadro da prova.

```
> goal Nc_v1.thy "(~A --> C) --> (A --> C) --> C";
> br impI 1;
> br impI 1;
> br classica 1;
> br impE 1;
> ba 1;
> br negI 1;
> by (res_inst_tac [("A","C")] negE 1);
> by (res_inst_tac [("A","A")] impE 2);
> ba 1; ba 1; ba 1;
```

Passemos agora a analisar uma segunda versão do sistema de dedução natural. Em `Ni_v2.thy` está definida a teoria para o sistema de dedução natural intuicionista, na versão que considera \neg , \supset , \wedge e \vee conectivas primitivas. Repare que temos regras de introdução e de eliminação para cada uma destas conectivas. A versão clássica desta teoria está em `Nc_v2.thy` e construída acrescentando a `Ni_v2.thy` a regra `classica`.

```
Ni_v2 = Pure +
[...]
consts
[...]
  Not      :: o => o           ("~_" [40] 40)
  "&"      :: [o, o] => o      (infixr 35)
  "|"      :: [o, o] => o      (infixr 30)
  "-->"    :: [o, o] => o      (infixr 25)
rules
  impI     "(A ==> B) ==> A-->B"
  impE     "[| A-->B ; A |] ==> B"
  negI     "(!!B. A ==> B) ==> ~A"
  negE     "[| A ; ~A |] ==> B"
  conjI    "[| A ; B |] ==> A&B"
  conjE1   "A&B ==> A"
  conjE2   "A&B ==> B"
  disjI1   "A ==> A|B"
  disjI2   "B ==> A|B"
  disjE    "[| A|B ; A ==> C ; B ==> C |] ==> C"
end
```

```

Nc_v2 = Ni_v2 +
rules
  classica      "(~A ==> A) ==> A"
end

```

Carregue a teoria Ni_v2.thy.

```
> use_thy "Ni_v2";
```

Atente na demonstraco de $\vdash A \wedge B \supset \neg(A \supset \neg B)$ que se segue, e construa o quadro da prova.

```

> goal Ni_v2.thy "A & B --> ~(A --> ~B)";
> br impI 1;
> br negI 1;
> br negE 1;
> br impE 2;
> ba 2;
> br conjE2 1;
> ba 1;
> br conjE1 1;
> ba 1;

```

Exerccio:

Carregue para o sistema a teoria Nc_v2.thy e prove, nesta teoria o seguinte resultado clssico:
 $\vdash \neg\neg A \supset A$

5.3 Questes de Avaliao Prtica

1. Use a teoria Ni_v2 para demonstrar:

(i) $\vdash (A \supset B) \supset (\neg B \supset \neg A)$

(ii) $\vdash A \vee B \supset (\neg A \supset B)$

6 Aula 5 (Laboratorial) - O Sistema Nc

Nesta sessão laboratorial iremos trabalhar com um *Sistema de Dedução Natural para a Lógica de Primeira Ordem Clássica (Nc)*.

6.1 Sistema de Dedução Natural para a Lógica de Primeira Ordem Clássica

Vamos usar os ficheiros `Nc.thy` e `Nc.ML` que deve copiar de `~mjf/ELP1/` para a sua área de trabalho.

```
Nc = Pure +
[...]
consts
[...]
  False  :: o
(* Connectives *)
  Not    :: o => o           ("~_" [40] 40)
  "&"    :: [o, o] => o      (infixr 35)
  "|"    :: [o, o] => o      (infixr 30)
  "-->"  :: [o, o] => o      (infixr 25)
(* Quantifiers *)
  All    :: ('a => o) => o    (binder "ALL " 10)
  Ex     :: ('a => o) => o    (binder "EX " 10)
[...]
rules
(* Propositional logic *)
  impI   "(A ==> B) ==> A-->B"
  impE   "[| A-->B ; A |] ==> B"
  conjI  "[| A ; B |] ==> A&B"
  conjE1 "A&B ==> A"
  conjE2 "A&B ==> B"
  disjI1 "A ==> A|B"
  disjI2 "B ==> A|B"
  disjE  "[| A|B ; A ==> C ; B ==> C |] ==> C"
  exFalsum "False ==> B"
  classica "(A --> B ==> A) ==> A"          (* Lema de Peirce *)
(* Definitions *)
  not_def "~A == A-->False"
(* Quantifiers *)
  allI   "(!!x. A(x)) ==> (ALL x.A(x))"
  allE   "(ALL x.A(x)) ==> A(x)"
  exI    "A(x) ==> (EX x.A(x))"
  exE    "[| EX x.A(x) ; !!x. A(x) ==> C |] ==> C"
end
```

Repare que \neg não é aqui considerada como uma conectiva primitiva, mas como uma abreviatura $\neg A \doteq A \supset \perp$. `False` é a representação, nesta teoria de \perp .

Nas linguagens de primeira ordem existem duas classes de entidades: os *predicados* e os *indivíduos*.

O quantificadores \forall e \exists representam-se aqui por **ALL** e **EX**, respectivamente. Confronte as regras aqui associadas aos quantificadores com as que foram definidas na parte teórica da disciplina.

!! representa, em Isabelle, a quantificação universal ao meta-nível e \implies a meta-implicação. Portanto, !! e \implies são conectivas da meta-lógica, que no nosso caso é a teoria **Pure**.

Entre agora no sistema e carregue esta teoria.

```
isabelle Pure
> use_thy "Nc";
```

6.2 Provas por Propagação em Retrocesso

Para vermos como funcionam as demonstrações envolvendo quantificadores, vamos analisar detalhadamente duas pequenas provas. Ambas implicam o “descartar” de quantificadores do objectivo a provar. Vamos ver que devido à ordem pela qual os quantificadores aparecem no objectivo da prova, um dos objectivos é demonstrável, mas o outro não.

Comecemos pelo caso com sucesso. Lance então aprova de $\vdash \forall x.\exists y.P(x) \supset P(y)$

```
> goal Nc.thy "ALL x. EX y. P(x) --> P(y)";
```

podemos aplicar a regra **allI** para retirarmos o quantificador \forall

A regra **allI** pode ser lida como “se $A(x)$ é verdade para todo o x , então $\forall x.A(x)$ é verdade”.

```
> br allI 1;
Level 1
ALL x. EX y. P(x) --> P(y)
  1. !!x. EX y. P(x) --> P(y)
```

Ao aplicar esta regra na prova em retrocesso, cria-se um sub-objectivo quantificado universalmente ao meta-nível (**!!x.**). A variável **x**, ligada a esta quantificação chama-se *parametro*.

Para agora removermos o quantificador existencial podemos aplicar a regra **exI**.

```
> br exI 1;
Level 2
ALL x. EX y. P(x) --> P(y)
  1. !!x. P(x) --> P(?y1(x))
```

Uma vez que a resolução de **exI** com **EX y. P(x) --> P(y)** é feita no âmbito da quantificação **!!x**, a variável livre (*unknown*) introduzida pela regra **exI** passa a poder ser dependente de **x**, e o sistema representa-a por **(?y1(x))**. **?y1** é uma *função* desconhecida. Informalmente, podemos olhar para **?y1(x)** como representando qualquer termo, incluindo os que contêm **x**.

Faça agora

```
> br impI 1;
Level 3
ALL x. EX y. P(x) --> P(y)
  1. !!x. P(x) ==> P(?y1(x))
```

$?y1(x)$ pode ser substituído por qualquer termo que contenha x . O termo que nos interessa para $?y1(x)$, de modo a fechar a prova por assunção, é x ; o que faz com que $?y1$ seja a função identidade.

Como $?y1(x)$ pode ser substituído por x , pelos motivos acima expostos, podemos aplicar a assunção para terminar a prova com sucesso.

```
> ba 1;
Level 4
ALL x. EX y. P(x) --> P(y)
No subgoals!
```

Lancemos agora $\vdash \exists y. \forall x. P(x) \supset P(y)$ como objectivo a provar

```
> goal Nc.thy "EX y. ALL x. P(x) --> P(y)";
```

Como, relativamente à demonstração anterior, a ordem dos quantificadores está ao contrario, vamos começar por retirar o quantificador existencial

```
> br exI 1;
Level 1
EX y. ALL x. P(x) --> P(y)
  1. ALL x. P(x) --> P(?y)
```

A variável livre introduzida por esta resolução é apenas $?y$. $?y$ não pode ser substituída por termos que contenham x , uma vez que x é uma variável ligada.

Ao remover o quantificador universal

```
> br allI 1;
Level 2
EX y. ALL x. P(x) --> P(y)
  1. !!x. P(x) --> P(?y)
```

x está ainda ligada; agora por $!!$ em vez de \forall . Faça então

```
> br impI 1;
Level 3
EX y. ALL x. P(x) --> P(y)
  1. !!x. P(x) ==> P(?y)
```

Ao tentar agora aplicar a assunção para fechar a prova, a unificação falha!

```
> ba 1;
*** by: tactic failed
```

Apesar de $?y$ ser um termo ainda não instanciado, representa **um** termo constante (a escolher). Ora $!!x. P(x) ==> P(?y)$ é uma representação finita de uma infinidade de premissas (uma premissa para cada x do domínio). Ao aplicar a assunção apenas conseguimos fechar uma das premissas (para um dos elementos do domínio), por isso a tática falha.

Quando, na prova de cima, tínhamos $!!x. P(x) ==> P(?y1(x))$, como $?y1(x)$ representa um termo que pode conter x , tivemos a possibilidade de fechar esta infinidade de premissas por assunção, fazendo $?y1(x) = x$.

Exercício:

Faça agora, por si, a prova de $\vdash (\forall x.A(x)) \supset (\exists x.A(x))$. Construa o quadro da prova.

Acompanhe o desenrolar da prova de $\vdash (\exists x.\exists y.A(x,y)) \supset (\exists y.\exists x.A(x,y))$ que a seguir se apresenta, construindo no papel o quadro da prova.

```
> goal Nc.thy "(EX x. EX y. A(x,y)) --> (EX y. EX x. A(x,y))";
> br impI 1;
> br exE 1;
> ba 1;
> br exE 1;
> br exI 2;
> br exI 2;
> ba 2;
> ba 1;
```

Exercício:

Construa, em Nc, a prova de $\vdash (\exists x.A(x) \supset B(x)) \supset ((\forall x.A(x)) \supset (\exists x.B(x)))$

6.3 Questões de Avaliação Prática

1. Use a teoria Nc para demonstrar:

(i) $\vdash \neg(\exists x.A(x)) \supset (\forall x.\neg A(x))$

(ii) $\vdash \neg(\exists x.\neg A(x)) \supset (\forall x.A(x))$

7 Aula 6 (Laboratorial) - O Sistema G1

Nesta sessão laboratorial iremos trabalhar com um *Sistema Dedutivo de Gentzen* para a Lógica de Primeira Ordem Clássica.

7.1 Sistema Dedutivo de Gentzen para a Lógica de Primeira Ordem Clássica

Vamos usar os ficheiros `G1.thy` e `G1.ML` que deve copiar de `~mjf/ELP1/` para a sua área de trabalho.

```
G1 = Sequents +
consts
[...]
  True,False  :: o
  "="         :: ['a,'a] => o      (infixl 50)
  Not         :: o => o           ("~_" [40] 40)
  "&"         :: [o,o] => o       (infixr 35)
  "|"         :: [o,o] => o       (infixr 30)
  "-->","<->" :: [o,o] => o       (infixr 25)
  All         :: ('a => o) => o    (binder "ALL " 10)
  Ex          :: ('a => o) => o    (binder "EX " 10)
rules
(* Structural rules *)
  basic      "${H, P, $G ; $E, P, $F}"
  weakR      "${H ; $E, $F} ==> {H ; $E, P, $F}"
  weakL      "${H, $G ; $E} ==> {H, P, $G ; $E}"
  conR       "${E ; $F, A, $G, A} ==> {E ; $F, A, $G}"
  conL       "${F, A, $G, A ; $E} ==> {F, A, $G ; $E}"
  cut        "${H ; $E, P} ==> {H, P ; $E} ==> {H ; $E}"
(* Propositional rules *)
  conjR      "${H ; $E, P, $F} ==> {H ; $E, Q, $F} ==> {H ; $E, P&Q, $F}"
  conjL      "${H, P, Q, $G ; $E} ==> {H, P & Q, $G ; $E}"
  disjR      "${H ; $E, P, Q, $F} ==> {H ; $E, P|Q, $F}"
  disjL      "${H, P, $G ; $E} ==> {H, Q, $G ; $E} ==> {H, P|Q, $G ; $E}"
  impR       "${H, P ; $E, Q, $F} ==> {H ; $E, P-->Q, $F}"
  impL       "${H,$G ; $E,P} ==> {H, Q, $G ; $E} ==> {H, P-->Q, $G ; $E}"
  FalseL     "${H, False, $G ; $E}"
(* Definitions *)
  iff_def    "P<->Q == (P-->Q) & (Q-->P)"
  not_def    "~P == P --> False"
  True_def   "True == False-->False"
(* Quantifiers *)
  allR       "(!!x. {H ; $E, P(x), $F}) ==> {H ; $E, ALL x.P(x), $F}"
  allwL      "${H, P(x), $G ; $E} ==> {H, ALL x.P(x), $G ; $E}"
  exwR       "${H ; $E, P(x), $F} ==> {H ; $E, EX x.P(x), $F}"
  exL        "(!!x. {H, P(x), $G ; $E}) ==> {H, EX x.P(x), $G ; $E}"
end
[...]
```

Como sabe, a definição de um *sistema de Gentzen* pode resumir-se nos seguintes pontos:

1. Definição do espaço de asserções.
2. Definição do subconjunto das asserções válidas.

Relativamente ao ponto 1, a linguagem lógica usada é a linguagem de primeira ordem gerada pelas conectivas primitivas $\wedge, \vee, \supset, \perp$ (aqui representadas por `&, |, -->, False`) e pelos quantificadores \forall e \exists (aqui, `ALL` e `EX`). A negação (aqui, `~`) é definida pela abreviatura $\neg A \doteq A \supset \perp$. Como pode verificar, esta linguagem foi ainda enriquecida com a definição do predicado primitivo `=` e de duas abreviaturas `True` \doteq `False` `-->` `False` e `P` \leftrightarrow `Q` \doteq `(P --> Q) & (Q --> P)`, para a verdade e a equivalência.

As asserções usadas nos sistemas de Gentzen chamam-se *sequentes* e definem-se como pares de multiconjuntos de fórmulas escritas na forma $\Gamma; \Delta$. A Γ dá-se o nome de *antecedente* e a Δ de *consequente*.

A intuição por detrás do conceito de sequente pode ser expressa do seguinte modo: “ $\Gamma; \Delta$ é um sequente válido se e só se, sendo válidas todas as fórmulas do antecedente Γ , é válida pelo menos uma fórmula do consequente Δ .”

A notação usada nesta teoria Isabelle para sequente é $\{\Gamma; \Delta\}$. Cada multiconjunto é uma sequência de fórmulas, separadas por vírgulas; e um identificador iniciado por `$` representa um multiconjunto qualquer.

Relativamente ao ponto 2, o sistema `G1`, que está a ser definido, é o menor conjunto de sequentes que verifica todas as regras que estão definidas na teoria.

Se confrontar o sistema aqui apresentado com os sistemas de Gentzen definidos na parte teórica desta disciplina, e tendo em conta que em `G1.ML` já estão demonstrados (e podem ser utilizados) os seguintes teoremas

```
[...]
(* LK Rules *)
"notR"   "{ $H, P; $E, $F } ==> { $H ; $E, ~P, $F }"
"notL"   "{ $H, $G ; $E, P } ==> { $H, ~P, $G ; $E }"
"allL"   "{ $H, P(x), $G, ALL x.P(x) ; $E } ==> { $H, ALL x.P(x), $G ; $E }"
"exR"    "{ $H ; $E, P(x), $F, EX x.P(x) } ==> { $H ; $E, EX x.P(x), $F }"
(** If-and-only-if rules **)
"iffR"   "{ $H,P ; $E,Q,$F } ==> { $H,Q ; $E,P,$F } ==> { $H ; $E, P <-> Q, $F }"
"iffL"   "{ $H,$G ; $E,P,Q } ==> { $H,Q,P,$G ; $E } ==> { $H, P <-> Q, $G ; $E }"
"TrueR"  "{ $H ; $E, True, $F }"
[...]
```

chega, com certeza, à conclusão de que esta teoria “engloba” os sistemas `G1`, `G1'`, `G1+Corte`, `G1'+Corte`, ali apresentados. Isto está a ser feito, porque sabemos que estes sistemas são todos equivalentes entre si.

Note que, com excepção dos axiomas `basic` e `FalseL` e da regra do corte, todas as regras se agrupam em pares: uma regra “esquerda” e uma regra “direita”. Se repararmos nas regras lógicas, vemos que todas elas são *regras de introdução*: o factor a destacar em cada regra aparece no seuante conclusão da regra, mas nunca nas premissas.

Este sistema tem propriedades que convém ainda realçar. Pondo de parte a regra do corte o sistema verifica a *propriedade de sub-fórmula* e a *propriedade da separação*.

- *Propriedade de sub-fórmula*: Em toda a prova com $\Gamma; \Delta$ como conclusão só ocorrem fórmulas que são sub-fórmulas de fórmulas que já ocorrem em Γ ou em Δ .
- *Propriedade da separação*: Na construção de uma prova para a conclusão $\Gamma; \Delta$ só são necessárias as regras lógicas relativas aos operadores que fazem inicialmente parte das fórmulas em Γ ou em Δ .

Como deve saber, quando se acrescenta a regra do corte, ambas as propriedades podem deixar de se verificar. No entanto, a regra do corte traz vantagens, no que diz respeito à reutilização de provas e à relação com os sistemas de dedução natural. Por enquanto, não vamos trabalhar com o corte. Nesta aula laboratorial vamos desenvolver varias provas sem utilizar a regra do corte. Ou seja, vamos trabalhar nos sistemas G1 ou G1', apresentados na parte teórica desta disciplina.

Esta teoria está definida à custa de uma teoria pré-definida do Isabelle, chamada **Sequents**. Por isso, podemos chamar o sistema da seguinte forma:

isabelle Sequents

E carregar a teoria G1 fazendo

```
> use_thy "G1";
```

7.2 Provas por Propagação em Retrocesso

Lance o sequirente $A \supset B \vee C ; (A \supset B) \vee (A \supset C)$ como objectivo a provar

```
> goal G1.thy "{ A-->B|C ; (A-->B)|(A-->C) }";
```

Com base na propriedade da separação, é fácil descobrir uma prova deste sequirente. Desenvolva então esta prova, e construa no papel o quadro da prova.

Exercício:

Use a teoria G1 para demonstrar os seguintes sequentes:

(i) $A \supset \neg B, A \wedge B ;$

(ii) $\neg\neg A ; A$

(iii) $\neg(P \wedge Q) ; P \supset \neg Q$

Construa os quadros das provas.

Lance agora o sequirente $\neg\exists x.\neg P(x) ; \forall x.P(x)$ como objectivo a provar

```
> goal G1.thy "{ ~ (EX x. ~P(x)) ; ALL x. P(x) }";
```

Para nos livrar-mos da negação que está no antecedente deste sequirente, podemos usar a definição `not_def` combinado com as regras `impL`, `FalseL`, ou então usar a regra `notL`. Vamos optar pela segunda solução (por ser a mais prática).

```
> br notL 1;
```

Neste ponto da prova, temos uma opção importante a tomar: usar a regra `allR`, para que no futuro a prova possa ser fechada pela regra `basic`.

```
> br allR 1;
```

A prova termina agora facilmente, fazendo

```
> br exwR 1;  
> br notR 1;  
> br basic 1;
```

Note que se tivéssemos optado por aplicar primeiro a regra `exwR` e só depois `allR`, não conseguiríamos fechar a prova.

Exercício:

Construa, em **G1**, a prova de $\neg(\forall x.P(x)) \supset (\exists x.\neg P(x))$

7.3 Questões de Avaliação Prática

1. Use a teoria **G1** para demonstrar o seguinte:

$$(\forall x.A(x)) \supset (\exists x.B(x)) ; (\exists x.A(x) \supset B(x))$$

Construa o quadro da prova, e justifique as opções tomadas.

8 Aulas 7 e 8 - Relação entre G1 e Nc e Eliminação do Corte

8.1 Regra do Corte. Relação de G1 com Nc. Eliminação do Corte.

Vantagens de se considerar o corte no processo de dedução:

- “Guardar provas” para serem utilizadas e provas posteriores. Por exemplo, guardar o lema $\Gamma; C$ para provar $\Gamma; \Delta$, fazendo:

$$\frac{\Gamma; C \quad \Gamma, C; \Delta}{\Gamma; \Delta} \text{ (corte)}$$

- Facilidade com que se passa de um sistema de Gentzen com corte, para um sistema de dedução Natural.

Teorema: $\Gamma \vdash A$ em Nc sse $\Gamma; A$ pertence a G1 + Corte.

Conversão de uma prova em Nc para G1 + Corte:

Nc	G1 + Corte
Regra de Introdução	Regra Direita
Regra de Eliminação	Regra Esquerda + Corte
Assunção	Regra Base

8.1.1 Exercícios

1. Relativamente a cada um dos teoremas que se seguem:

- (i) $\vdash A \wedge B \supset A \vee B$
- (ii) $\neg(A \vee B) \vdash \neg A \wedge \neg B$
- (iii) $\vdash \forall x. \exists y. P(x) \supset P(y)$
- (iv) $\vdash A \vee B \supset (\neg A \supset B)$
- (v) $\vdash A \wedge B \supset \neg(A \supset \neg B)$
- (vi) $\vdash \exists x. \exists y. A(x, y) \supset \exists y. \exists x. A(x, y)$

1.1. Desenvolva, no sistema de dedução natural, Nc, provas para estes teoremas.

1.2. Converta as provas encontradas em 1.1. em provas do sistema G1 + Corte.

1.3. Transforme cada prova de 1.2. numa prova que não use a regra do corte.

2. Prove que a seguinte regra é válida no sistema G1 + Corte

$$\frac{; A \vee B \quad A; C \quad B; C}{; C}$$

9 Aula 9 (Laboratorial) - Os Sistemas G1 e GS

A primeira parte desta sessão laboratorial destina-se à resolução de mais alguns exercícios e questões de avaliação prática sobre o *Sistema Dedutivo de Gentzen* para a Lógica de Primeira Ordem Clássica. Na segunda parte iremos trabalhar com o sistema **GS** de *Gentzen-Schutte* (ou *Gentzen Simétrico*).

9.1 O Sistema Dedutivo G1

Relembre a teoria G1 que já utilizou nas aulas anteriores.

```
G1 = Sequents +
consts
[...]
  True,False  :: o
  "="        :: ['a,'a] => o      (infixl 50)
  Not         :: o => o           ("~_" [40] 40)
  "&"         :: [o,o] => o       (infixr 35)
  "|"         :: [o,o] => o       (infixr 30)
  "-->","<->" :: [o,o] => o       (infixr 25)
  All         :: ('a => o) => o    (binder "ALL " 10)
  Ex          :: ('a => o) => o    (binder "EX " 10)
rules
(* Structural rules *)
  basic      "${H, P, $G ; $E, P, $F}"
  weakR      "${H ; $E, $F} ==> {H ; $E, P, $F}"
  weakL      "${H, $G ; $E} ==> {H, P, $G ; $E}"
  conR       "${E ; $F, A, $G, A} ==> {E ; $F, A, $G}"
  conL       "${F, A, $G, A ; $E} ==> {F, A, $G ; $E}"
  cut        "${H ; $E, P} ==> {H, P ; $E} ==> {H ; $E}"
(* Propositional rules *)
  conjR      "${H ; $E, P, $F} ==> {H ; $E, Q, $F} ==> {H ; $E, P&Q, $F}"
  conjL      "${H, P, Q, $G ; $E} ==> {H, P & Q, $G ; $E}"
  disjR      "${H ; $E, P, Q, $F} ==> {H ; $E, P|Q, $F}"
  disjL      "${H, P, $G ; $E} ==> {H, Q, $G ; $E} ==> {H, P|Q, $G ; $E}"
  impR       "${H, P ; $E, Q, $F} ==> {H ; $E, P-->Q, $F}"
  impL       "${H,$G ; $E,P} ==> {H, Q, $G ; $E} ==> {H, P-->Q, $G ; $E}"
  FalseL     "${H, False, $G ; $E}"
(* Definitions *)
  iff_def    "P<->Q == (P-->Q) & (Q-->P)"
  not_def    "~P == P --> False"
  True_def   "True == False-->False"
(* Quantifiers *)
  allR       "(!!x. {H ; $E, P(x), $F}) ==> {H ; $E, ALL x.P(x), $F}"
  allwL      "${H, P(x), $G ; $E} ==> {H, ALL x.P(x), $G ; $E}"
  exwR       "${H ; $E, P(x), $F} ==> {H ; $E, EX x.P(x), $F}"
  exL        "(!!x. {H, P(x), $G ; $E}) ==> {H, EX x.P(x), $G ; $E}"
end
[...]
```

Relembre também os teoremas já demonstrados em G1.ML.

```
[...]
(* LK Rules *)
"notR"   "{\$H, P; \$E, \$F} ==> {\$H ; \$E, ~P, \$F}"
"notL"   "{\$H, \$G ; \$E, P} ==> {\$H, ~P, \$G ; \$E}"
"allL"   "{\$H, P(x), \$G, ALL x.P(x) ; \$E} ==> {\$H, ALL x.P(x), \$G ; \$E}"
"exR"    "{\$H ; \$E, P(x), \$F, EX x.P(x)} ==> {\$H ; \$E, EX x.P(x), \$F}"
(** If-and-only-if rules **)
"iffR"   "{\$H,P ; \$E,Q,\$F} ==> {\$H,Q ; \$E,P,\$F} ==> {\$H ; \$E, P <-> Q, \$F}"
"iffL"   "{\$H,\$G ; \$E,P,Q} ==> {\$H,Q,P,\$G ; \$E} ==> {\$H, P <-> Q, \$G ; \$E}"
"TrueR"  "{\$H ; \$E, True, \$F}"
[...]
```

Entre agora no sistema isabelle `Sequents` e carregue esta teoria

```
> use_thy "G1";
```

Exercícios:

- Use a teoria `G1` para demonstrar que o sequente $C \supset A \wedge B, C \wedge D ; A \wedge B$ é válido. Associe o resultado que acabou de provar ao identificador “`lema1`”.
- Use agora o lema que demonstrou em 1. para provar $C \supset A \wedge B, C \wedge D ; A$

9.2 Questões de Avaliação Prática

- Usando a teoria `G1`, prove os sequentes:
 - $\exists x.\forall y.P(x, y) \wedge A \supset Q(x, y) , A ; \forall y.\exists x.P(x, y) \supset Q(x, y)$
 - $(\forall y.A(y)) \supset (\exists y.B(y) \wedge C(y)) ; \exists x.\neg A(x) \vee \exists x.C(x)$

Contraa os quadros das provas.

- Use a teoria `G1` para demonstrar que as seguintes regras são válidas:

(i)

$$\frac{Q ; P \quad P ; A \quad ; A \vee Q}{; A}$$

(ii)

$$\frac{E ; F \quad F ; A, C \quad ; E \vee A}{; A \vee C}$$

Desenhe as árvores de prova.

9.3 O Sistema GS de *Gentzen-Schutte* (ou *Gentzen Simétrico*)

Vamos agora usar os ficheiros `GS.thy` e `GS.ML` que deve copiar de `~mjf/ELP1/` para a sua área de trabalho.

```
GS = G1 +
rules
  axiom  "{; $E, P, $H, ~P, $F}"
  change "{; $E, P, $H, ~P, $F} ==> {; $E, ~P, $H, P, $F}"
  conj   "{; $E, P, $F} ==> {; $E, Q, $F} ==> {; $E, P&Q, $F}"
  disj   "{; $E, P, Q, $F} ==> {; $E, P|Q, $F}"
  all    "(!!x. {; $E, P(x), $F}) ==> {; $E, ALL x.P(x), $F}"
  ex     "{; $E, P(x), $F, EX x.P(x)} ==> {; $E, EX x.P(x), $F}"
  cut    "{; $E, P, $F} ==> {; $H, ~P, $G} ==> {; $E, $F, $H, $G}"
(* Definitions *)
not_not  "~~A == A"
not_conj "~(A&B) == ~A | ~B"
not_disj "~(A|B) == ~A & ~B"
not_all  "~(ALL x. A) == EX x.~A"
not_ex   "~(EX x. A) == ALL x. ~A"
imp_def  "A-->B == ~A | B"
end
```

Como sabe, o sistema **GS** usa sequentes com o antecedente vazio. Portanto, todos os sequentes são da forma $; \Delta$ (em isabelle, $\{; \Delta\}$), sendo Δ um multiconjunto de fórmulas.

Este sistema usa $\wedge, \vee, \forall, \exists$ como conectivas primitivas. A implicação $A \supset B$ é definida como $\neg A \vee B$ e a negação como uma transformação involutiva.

Como os sequentes só têm lado direito, temos apenas uma regra para cada conectiva, um axioma e uma regra auxiliar (`change`) que permite trocar a ordem porque uma fórmula e a sua negação aparecem no sequente. Esta regra é necessária porque, apesar do consequente ser um multiconjunto, ele está aqui implementado como uma sequência.

A regra do corte tem uma forma adaptada ao facto dos sequentes só terem lado direito.

O tratamento da negação é feito com um conjunto de regra de reescrita que indicam a forma como as fórmulas que tem a negação como conectiva principal, devem ser transformadas.

Pelos motivos acima descritos, são particularmente uteis no desenvolvimento de provas neste sistema os comandos `goalw`, `by(rewrite_tac ...)` e `bw`.

Em `GS.ML` já foi declarado

```
val defs = [not_not, not_conj, not_disj, not_all, not_ex, imp_def];
```

9.4 Provas por Propagação em Retrocesso

Lance o sequente $; \neg\neg A \supset A$ como objectivo a provar.

```
> goal GS.thy "{; ~~A --> A }";
```

Repare como podemos desenvolver esta prova:

```

> bw imp_def;
> br disj 1;
> bw not_not;
> br change 1;
> br axiom 1;

```

Construa o quadro da prova.

Lance, agora, o sequente ; $A \supset \neg B, \neg(\neg A \vee \neg B)$ como objectivo a provar. Como a implicação e a negação necessitam de ser reescritas, o mais conveniente é lançar a prova do seguinte modo

```

> goalw GS.thy defs "{; A --> ~B, ~(~A | ~B) }";

```

Repare como de uma só vez foi feita a reescrita da implicação, da negação da disjunção e da dupla negação (experimente lançar a prova com goal). A prova desenvolve-se agora muito facilmente fazendo:

```

> br disj 1;
> br conj 1;

```

Teoricamente, neste estado da prova, ambas as premissas deveriam fechar por axiom. Mas, como os multiconjuntos estão implementados como seqüências, e a ordem pela qual as fórmulas A e $\neg A$, e B e $\neg B$ aparecem está trocada, temos que primeiro aplicar a regra **change**.

Podemos, pois, terminar a prova fazendo:

```

> br change 2;
> br axiom 2;
> br change 1;
> br axiom 1;

```

Construa, no papel, o quadro da prova.

Exercícios:

Use a teoria GS para demonstrar os seguintes sequentes:

(i) ; $(A \supset B) \supset (A \supset \neg B) \supset \neg A$

(ii) ; $\neg(P \supset R), (Q \supset R) \supset (P \vee Q) \supset R$

(iii) ; $\exists x.\forall y.A(x, y) \supset \forall y.\exists x.A(x, y)$

(iv) ; $\exists x.P(x) \supset \exists x.P(x)$

Construa os quadros das provas.

10 Aula 10 - Resolução e Unificação

10.1 Resolução e Unificação

Resolução e Unificação são duas técnicas que estão na base dos sistemas de prova automática (como os sistemas que implementam a Programação Lógica), assim como dos Sistemas de Prova Assistida onde estão na base da semântica operacional da meta-lógica (como ocorre com o sistema *Isabelle*).

Um exemplo simples de lógica com resolução é o sistema dedutivo onde as asserções são sequentes $(\Gamma ; \Delta)$ sujeitos às seguintes restrições:

- (i) Δ ou é vazio ou é formado por um único literal;
- (ii) Γ é um multiconjunto de literais.

Estes sequentes designam-se por *cláusulas*, e uma cláusula da forma $(\Gamma ;)$ designa-se por *objectivo*.

A única regra de inferência usada é designada por *regra de resolução linear*, e é uma forma simplificada da regra do corte

$$\frac{\Gamma, A ; \quad \Gamma' ; A}{\Gamma, \Gamma' ;} \quad (\text{resolução linear})$$

Generalizando para situações em que os literais contêm variáveis livres, um objectivo $(\Gamma, A ;)$ é resolúvel com a cláusula $(\Gamma' ; B)$ quando:

- (i) as variáveis do objectivo e da cláusula são distintos;
- (ii) $\{A \stackrel{?}{=} B\}$ unifica.

Com estas condições é sempre possível isolar duas substituições diferentes σ e μ , tais que $\sigma(A) \equiv \mu(B)$, e portanto

$$\frac{\sigma(\Gamma), \sigma(A) ; \quad \mu(\Gamma') ; \mu(B)}{\sigma(\Gamma), \mu(\Gamma') ;} \quad \sigma(A) \equiv \mu(B)$$

Um *programa* P é um conjunto de cláusulas que não são objectivos, isto é, são todas da forma $(\Delta ; A)$. Uma *P-derivação* é uma árvore de dedução que tem por conclusão a *cláusula vazia* $(;)$ e em que todas as folhas ou são objectivos ou são elementos de P .

Sob o ponto de vista da interpretação semântica dos sequentes, um objectivo $(\Gamma ;)$ é um sequente que, sendo válido, indica que o conjunto de fórmulas Γ é inconsistente. Assim sendo, a cláusula vazia $(;)$ nunca pode ser válida.

Portanto, uma P -derivação com objectivo $(\Gamma ;)$ é uma prova de

$$(\Gamma ;) \Rightarrow (;)$$

Temos assim, ao meta-nível, uma negação da inconsistência de Γ . Em termos da lógica clássica, isto corresponde à afirmação de que Γ é *consistente* e, portanto, todas as fórmulas de Γ podem ser simultaneamente válidas.

É possível transformar qualquer dedução por resolução linear numa prova por propagação em retrocesso num sistema de Gentzen.

10.1.1 Exercícios

1. Usando a resolução linear, construa uma P-derivação com o programa

$$P \doteq \{ (R, Q ; A) , (H ; R) , (; H) , (; Q) \}$$

que tenha por objectivo inicial a cláusula:

- (i) $A ;$
 - (ii) $R, Q ;$
2. Transforme as deduções por resolução linear, feitas em 1., em provas por propagação em retrocesso do sistema de Gentzen.
 3. Como deve saber, dado o programa P e o objectivo $(\Gamma;)$, qualquer P-derivação iniciada em $(\Gamma;)$ prova a validade de $(\overline{P};\overline{\Gamma}) \in G1$. Use este teorema para provar, por resolução, a validade dos seguintes sequentes.

- (i) $A \supset B, A, A \supset C, B \wedge C \supset R ; R$

- (ii) $R \wedge Q \wedge S \supset A, R \wedge B \supset S, R, Q, S ; A \wedge R$

4. Use o algoritmo de simplificação de multi-unificandos para verificar se são unificáveis:

- (i) $\{ f(X, Y) \stackrel{?}{=} f(h(Y), h(Y)) \}$

- (ii) $\{ h(Y, f(X, g(Z))) \stackrel{?}{=} h(g(a), f(Y, X)) , m(Z) \stackrel{?}{=} m(a) \}$

Indique, se existir, o *unificador*.

5. Dado o seguinte conjunto de cláusulas

$$P \doteq \left\{ \begin{array}{l} p(d, X) \Leftarrow , p(a, e) \Leftarrow , p(a, b) \Leftarrow , p(b, c) \Leftarrow , \\ v(X, Y) \Leftarrow p(X, Z), p(Z, Y) \\ m(X, Y) \Leftarrow p(Z, X), p(Z, Y) \end{array} \right\}$$

Apresente uma P-derivação de objectivo:

- (i) $v(d, e)$

- (ii) $m(e, X)$

Indique a *resposta* e a *conclusão* da derivação encontrada.

11 Aula 11 (Laboratorial) - O Sistema S4

Nesta sessão laboratorial iremos trabalhar com um *Sistema Dedutivo de Gentzen para a Lógica Modal*.

11.1 O Sistema Dedutivo de Gentzen para a Lógica Modal

S4 = Modal0 +

```
rules
  lstar0      "|L>"
  lstar1      "$G |L> $H ==> []P, $G |L> []P, $H"
  lstar2      "$G |L> $H ==> P, $G |L> $H"
  rstar0      "|R>"
  rstar1      "$G |R> $H ==> <>P, $G |R> <>P, $H"
  rstar2      "$G |R> $H ==> P, $G |R> $H"
(* Rules for [] and <> *)
  boxR      "[| $E |L> $E'; $F |R> $F'; $G |R> $G' |] ==> \
\
          {$E'; $F', P, $G'} ==> {$E; $F, []P, $G}"
  boxL      "{$E,P,$F,[]P; $G} ==> {$E, []P, $F; $G}"
  diaR      "{$E ; $F,P,$G,<>P} ==> {$E ; $F, <>P, $G}"
  diaL      "[| $E |L> $E'; $F |L> $F'; $G |R> $G' |] ==> \
\
          {$E', P, $F'; $G'} ==> {$E, <>P, $F; $G}"
end
```

Modal0 = G1 +

```
consts
  box      :: "o=>o"      ("[]_" [50] 50)
  dia      :: "o=>o"      ("<>_" [50] 50)
  "@Lstar" :: "two_seqe"  ("(_)|L>(_)" [6,6] 5)
  "@Rstar" :: "two_seqe"  ("(_)|R>(_)" [6,6] 5)
[...]
```

Como sabe, genericamente, as lógicas modais aumentam a linguagem da Lógica de Primeira Ordem com dois operadores proposicionais unários, chamados *operadores modais*: o operador *necessidade* \Box e o operador *possibilidade* \Diamond .

Vamos trabalhar com o cálculo de seqüentes de Gentzen para o sistema S4. Este sistema, S4, corresponde ao sistema a que na parte teórica desta disciplina se chamou G1's.

Como pode confirmar, observando a teoria acima apresentada, o sistema S4 obtém-se do sistema G1, acrescentando à sintaxe os operadores modais: \Box de *necessidade* (aqui representado por $[]$) e \Diamond de *possibilidade* (aqui representado por $\langle \rangle$); assim como as relações de redução de contextos \triangleright_E (aqui representada por $|L\rangle$) e \triangleright_D (aqui representada por $|R\rangle$).

Relativamente às regras, foram acrescentadas:

- três regras que definem a relação de redução de contextos esquerdos;
- três regras que definem a relação de redução de contextos direitos;
- regras modais, para os operadores \Box e \Diamond .

No que respeita à redução de contextos, pode-se dizer que \triangleright_E retira do contexto todas as fórmulas não qualificadas com \Box (isto é, só mantem as fórmulas do tipo $\Box A$) e \triangleright_D retira do contexto todas as fórmulas não qualificadas com \Diamond (isto é, só mantem as fórmulas do tipo $\Diamond A$).

Recorde que as reduções de contextos foram introduzidas de modo a possibilitar que as condições de aplicabilidade das regras modais sejam apenas ditadas pelas suas fórmulas principais. As condições impostas nos contextos foi transferida para as relações \triangleright_E e \triangleright_D .

Para arrancar o Isabelle com o sistema S4, faça:

```
isabelle S4
```

11.2 Provas por Propagação em Retrocesso

Vamos começar com uma pequena prova. Lance o seguinte ; $\Box A \supset \Diamond A$ como objectivo a provar

```
> goal S4.thy "{ ; []A --> <>A }";
```

Faça agora

```
> br impR 1;
```

Neste ponto da prova tanto pode aplicar a regra de \Box à esquerda como de \Diamond à direita. Podemos escolher indistintamente qualquer delas, pois nenhuma das regras envolve reduções de contextos.

Por isso, podemos terminar a prova fazendo

```
> br boxL 1;
> br diaR 1;
> br basic 1;
```

Vejamos agora um caso um pouco mais complicado. Lance o seguinte $\Diamond(P \vee Q)$; $\Diamond P \vee \Diamond Q$ como objectivo a provar

```
> goal S4.thy "{<>(P | Q) ; <>P | <>Q}";
```

Faça agora

```
> br disjR 1;
```

Agora, tanto podemos tomar como fórmula principal uma fórmula qualificada com \Diamond que está no lado esquerdo do sequente, como no lado direito. Acontece que a regra associada a \Diamond do lado esquerdo envolve a redução de contextos e ao aplicarmos a regra, isso faz com que do lado esquerdo do sequente desapareçam todas as fórmulas não qualificadas com \Box , e do lado direito, todas as fórmulas não qualificadas com \Diamond . Por isso, devemos aplicar primeiro a regra `diaL` e só depois aplicar `diaR`, pois se aplicarmos primeiro `diaR`, estamos a retirar \Diamond de uma fórmula e essa fórmula será depois eliminada pela redução (o que torna a aplicação da regra `diaR` um passo desnecessário). Faça então

```
> br diaL 1;
Level 2
{<>(P | Q) ; <>P | <>Q}
  1. |L>$?E'1
  2. |L>$?F'1
  3. <>P, <>Q|R>$?G'1
  4. {$?E'1, P | Q, $?F'1 ; $?G'1}
```

Como pode ver, ficamos com quatro premissas em aberto. Isto deve-se ao formato da regra `diaL` que envolve a redução de dois subcontextos esquerdos (o que dá origem às premissas 1 e 2) e a redução do contexto direito (que origina a premissa 3). A premissa 4 é o novo sequente a provar, mas está dependente do resultado das reduções dos contextos que estão presentes nas premissas 1, 2 e 3. Como tal, convem tratar primeiro dessas premissas, e para isso devemos usar as regras definidoras das reduções de contextos.

```
> br lstar0 1;
> br lstar0 1;
> br rstar1 1;
> br rstar1 1;
> br rstar0 1;
```

Temos agora o sequente a provar completamente instanciado. A prova desenrola-se, facilmente, fazendo

```
> br diaR 1;
> br disjL 1;
> br basic 1;
> br diaR 1;
> br basic 1;
```

Volte, de novo, ao nível 1 da prova.

```
> choplev 1;
```

Como vimos anteriormente, neste ponto da prova a aplicação da regra `diaL` expande a prova em quatro premissas, três das quais sobre a redução de contextos. Já vimos também que a prova dessas premissas, embora simples, pode ser trabalhosa. O sistema `S4` fornece uma tática que permite aplicar uma regra por resolução, fazendo o sistema a redução de contextos automaticamente, caso isso seja necessário. Tal tática é `S4_Prover.rule_tac` e tem como argumentos o identificador da regra e a premissa à qual a regra vai ser aplicada. Podemos então fazer

```
> by (S4_Prover.rule_tac [diaL] 1);
```

Verifique como, de uma só vez, forão feitas todas as reduções de contextos e passou-se directamente para o estado da prova que verdadeiramente nos interessa (repare que ao escrever no papel o quadro da prova, é isto que realmente fazemos, reduzindo os contextos de modo implícito).

Podemos definir uma função que abrevie a aplicação desta tática fazendo

```
> fun brr r i = by (S4_Prover.rule_tac [r] i);
```

Note que esta função já poderia estar definida em ficheiro para ser carregada.

Podemos agora experimentar o que acabamos de definir

```
> undo();
> brr diaL 1;
```

Agora já sabe como terminar esta prova ...

Lance agora a prova do sequente $\Box(A \supset B), \Box A, C ; \Box B, D$

```
> goal S4.thy "{ [] (A --> B) , [] A , C ; [] B , D }";
```

Faça então

```
> brr boxR 1;
```

Repare como os contextos esquerdos e direitos foram reduzidos.

Podemos usar a tática `brr` ainda que na regra a aplicar não estejam envolvidas reduções de contextos. A prova desenvolve-se facilmente fazendo

```
> brr boxL 1;
> brr boxL 1;
> brr impL 1;
> brr basic 1;
> brr basic 1;
```

Faça no papel o quadro da prova.

Exercício:

Use a teoria `S4` para demonstrar os seguintes seqüentes:

- (i) $\neg\Box(\neg A) ; \Diamond A$
- (ii) $; \Diamond(P \wedge Q) \supset \Diamond P \wedge \Diamond Q$
- (iii) $; \Diamond\Box(\Diamond\Box P \supset \Box P)$

Construa os quadros das provas.

11.3 Questões de Avaliação Prática

1. Use a teoria `S4` para demonstrar o seqüente:

$$\Diamond P \vee \Diamond Q ; \Diamond(P \vee Q)$$

Construa o quadro da prova, e justifique as opções tomadas.

12 Aula 12 (Laboratorial) - O Sistema S4

Nesta sessão laboratorial destina-se à resolução de questões de avaliação prática para o *Sistema Dedutivo de Gentzen para a Lógica Modal*, S4.

12.1 O Sistema Dedutivo S4

S4 = Modal0 +

```
rules
  lstar0      "|L>"
  lstar1      "$G |L> $H ==> []P, $G |L> []P, $H"
  lstar2      "$G |L> $H ==> P, $G |L> $H"
  rstar0      "|R>"
  rstar1      "$G |R> $H ==> <>P, $G |R> <>P, $H"
  rstar2      "$G |R> $H ==> P, $G |R> $H"
(* Rules for [] and <> *)
  boxR        "[| $E |L> $E'; $F |R> $F'; $G |R> $G' |] ==> \
\
\                {$E'; $F', P, $G'} ==> {$E; $F, []P, $G}"
  boxL        "{$E,P,$F,[]P; $G} ==> {$E, []P, $F; $G}"
  diaR        "{$E ; $F,P,$G,<>P} ==> {$E ; $F, <>P, $G}"
  diaL        "[| $E |L> $E'; $F |L> $F'; $G |R> $G' |] ==> \
\
\                {$E', P, $F'; $G'} ==> {$E, <>P, $F; $G}"
end
```

Modal0 = G1 +

```
consts
  box          :: "o=>o"          ("[]_" [50] 50)
  dia          :: "o=>o"          ("<>_" [50] 50)
  "@Lstar"    :: "two_seqe"      ("(_)|L>(_)" [6,6] 5)
  "@Rstar"    :: "two_seqe"      ("(_)|R>(_)" [6,6] 5)
[...]
```

Como sabe, genericamente, as lógicas modais aumentam a linguagem da Lógica de Primeira Ordem com dois operadores proposicionais unários, chamados *operadores modais*: o operador *necessidade* \Box e o operador *possibilidade* \Diamond .

Vamos trabalhar com o cálculo de seqüentes de Gentzen para o sistema S4. Este sistema, S4, corresponde ao sistema a que na parte teórica desta disciplina se chamou G1's.

Para arrancar o Isabelle com o sistema S4, faça:

```
isabelle S4
```

Relembre que o sistema S4 fornece uma tática que permite aplicar uma regra por resolução, fazendo o sistema a redução de contextos automaticamente, caso isso seja necessário. Tal tática é `S4_Prover.rule_tac`, e tem como argumentos o identificador da regra e a premissa à qual a regra vai ser aplicada. Como já vimos, podemos definir uma função que abrevie a aplicação desta tática fazendo

```
> fun brr r i = by (S4_Prover.rule_tac [r] i);
```

Note que esta função já poderia estar definida em ficheiro para ser carregada.

12.2 Questões de Avaliação Prática

1. Use a teoria **S4** para demonstrar os seguintes sequentes:

(i) ; $\Box P \vee \Box Q \supset \Box(\Box P \vee \Box Q)$

(ii) ; $\neg\Box(\forall x.\neg B(x)) \supset \Diamond(\exists x.B(x))$

(iii) ; $\Box(P \vee Q) \supset \Box P \vee \Diamond Q$

(iv) ; $\neg\Diamond(\exists x.\neg A(x)) \supset \Box(\forall x.A(x))$

Construa os quadros das provas, e justifique as opções tomadas.

Referências

- [1] José Manuel Valença, *Sebenta Teórica de Elementos Lógicos da Programação I* (Manuscrito). Universidade do Minho, Departamento de Informática, 1997/98.
- [2] Lawrence C. Paulson. *Introduction to Isabelle*. Technical Report, University of Cambridge, Computer Laboratory, 1996.
- [3] Lawrence C. Paulson. *The Isabelle reference manual*. Technical Report, University of Cambridge, Computer Laboratory, 1996.
- [4] Lawrence C. Paulson. *Isabelle's object-logics*. Technical Report, University of Cambridge, Computer Laboratory, 1996.
- [5] Lawrence C. Paulson. *Toll Support for Logics of Programs*. University of Cambridge, Computer Laboratory, 1996.

A Exame Prático Laboratorial

Elementos Lógicos da Programação I

Exame Prático Laboratorial

1997/98

Use o sistema *isabelle* para resolver cada uma das questões abaixo apresentadas. Relativamente a cada questão, construa o quadro da prova, e indique a sequência de comandos *isabelle* que utilizou na sua construção.

1. Considere o sistema dedutivo de Hilbert para a lógica proposicional clássica, **Hc**. Use a teoria **Hc** para demonstrar

$$P \supset Q, \neg(Q \supset R) \supset \neg P \vdash P \supset R$$

2. Considere o sistema de dedução natural para a lógica proposicional intuicionista, **Ni**. Use a teoria **Ni_v2** para demonstrar

$$\vdash (B \supset C) \supset (A \wedge B) \supset A \supset C$$

3. Considere o sistema de dedução natural para a lógica de primeira ordem clássica, **Nc**. Use a teoria **Nc** para demonstrar

$$\vdash \neg(\exists x.P(x) \vee Q(x)) \supset \forall y.\neg Q(y) \vee P(y)$$

4. Considere o sistema de dedutivo de Gentzen para a lógica de primeira ordem clássica, **G1**. Use a teoria **G1** para demonstrar

$$\forall x.R(x) \vee Q(x) \supset \exists y.P(y) ; \exists x.R(x) \supset P(x) \vee Q(x)$$

Justifique as principais opções tomadas.

5. Use a teoria **G1** para demonstrar que a seguinte regra é válida:

$$\frac{A, C ; B \quad ; B \vee A \quad ; C}{; B}$$

6. Considere o sistema de dedutivo de Gentzen para a lógica modal, **S4**. Use a teoria **S4** para demonstrar

$$; \diamond \square (\diamond A \supset \square \diamond A)$$

Justifique as principais opções tomadas.