6. Automação

Neste capítulo aborda-se a automação da lógica Lact_N e da análise de organizações proposta no capítulo anterior (análise esta essencialmente suportada por deduções na lógica Lact_N).

Repare-se que existem propostos na literatura vários métodos para automatizar muitas das lógicas modais normais mais usuais[83]: métodos baseados em tableaux semânticos (veja-se e.g. [Fitting 72, 83, 88]); métodos baseados em "resolução" (veja-se e.g. [Enjalbert & Cerro 89]); métodos baseados no "cálculo de sequentes" (veja-se e.g. [Gallier 87]). Mas nenhum para automatizar lógicas modais clássicas não-normais[84].

Nesta dissertação opta-se por utilizar métodos baseados em tableaux semânticos para automatizar deduções na lógica Lact_N e, consequentemente, automatizar também a análise de organizações proposta no capítulo 5. Uma vez que a estratégia utilizada no método tableaux proposto nesta tese para automatizar deduções na lógica Lact_N pode também ser utilizada para automatizar deduções de algumas lógicas (uni)modais clássicas não-normais, dedicar-se-á uma secção deste capítulo à apresentação de alguns sistemas de tableaux para algumas dessas lógicas.

Na secção 6.1 faz-se uma introdução breve aos métodos tableaux, onde apenas se mencionará o método aplicado ao sistema de lógica modal PL (i.e. o sistema constituído por todas as tautologias). O principal texto de referência desta secção é o livro de Fitting [Fitting 90]. No entanto a notação aí proposta é aqui adaptada de modo a poder ser facilmente utilizada para mencionar diferentes sistemas de tableaux.

Na secção 6.2 apresentam-se várias regras para construção de tableaux (ditas regras de expansão de tableaux) e que estão associadas a axiomas-esquema e regras de inferência considerados em alguns sistemas de lógica modal clássica (uni)modal. São depois propostos vários sistemas de tableaux para automatizar as deduções de alguns sistemas de lógica (uni)modal clássica (proposicional) não-normal.

---

[83] Existem presentemente métodos de prova automática para lógicas modais normais do tipo rK, rKD, rKT, rK4, rKB, rKD4, rKTB, rKDB, rKT4 (ou S4), rKT5 (ou S5) e rK4+ $\Box(\Box A \rightarrow A) \rightarrow \Box A$.

[84] Note-se que, em [Gasquet & Herzig 94] é proposto um método de tradução de operadores modais clássicos em termos de combinações de operadores modais normais, obtendo-se assim lógicas multi-modais normais equivalentes, e para as quais existem métodos de prova eficientes. No entanto não existe nenhuma estratégia para escolha da tradução mais conveniente, ficando portanto em aberto quais os métodos de prova a aplicar aos sistemas de lógica modal clássica não mencionados por aqueles autores. Nesta dissertação optou-se portanto por apresentar um método de prova automática, sem recorrer a qualquer tradução do tipo aqui mencionado.

Na secção 6.3 é proposto um sistema de tableaux para a lógica $Lact_N$, por combinação e extensão de alguns dos sistemas de tableaux propostos na secção 6.2. Finaliza-se esta secção com o estudo das suas propriedades e referindo as estratégias utilizadas na sua implementação (veja-se o Anexo 1). Note-se que embora se tenha provado a correcção dos sistemas de tableaux propostos na secção 6.2 e 6.3, não foi possível provar a sua completude.

Finalmente, na secção 6.4 discute-se a automação da análise de organizações. A análise de organizações proposta no capítulo anterior é actualmente suportada por uma "bancada" implementada em Prolog (veja-se o Anexo 2). Conclui-se esta secção com a descrição da "bancada" e com exemplos da sua utilização para a análise de organizações muito simples.

6.1 Introdução Breve aos Métodos tableaux

Genericamente, os métodos tableaux são sistemas dedutivos de *refutação*, e.g. para se provar que uma fórmula A é um teorema, começa-se por se considerar a fórmula ¬A e tenta-se obter uma contradição. Em métodos tableaux essas provas tomam a forma de árvores, onde os nós são etiquetados por fórmulas, representando cada ramo uma conjunção das fórmulas (que ocorrem nos nós) e as ramificações disjunções, sendo a construção dessas árvores feita por utilização de regras denominadas "regras de expansão de tableaux" aplicadas à estrutura prévia da árvore em construção. As contradições são então detectadas através da confirmação de que todos os ramos do tableau obtido contêm simultaneamente uma fórmula e a sua negação.

O termo "semântico" é usualmente utilizado para adjectivar os métodos tableaux, uma vez que esta classe de métodos adoptam, em geral, como estratégia básica a construção "implícita" de modelos (caso existam) que tornem verdadeiras todas as fórmulas de um conjunto finito de fórmulas[85]. Dependendo do tipo de lógica, os respectivos métodos tableaux podem evidenciar ou não a estratégia mencionada. Em particular, nos métodos tableaux propostos para lógicas modais normais (veja-se e.g. [Fitting 72, 83, 88]), a utilização de primitivas semânticas está claramente explicitada pelo modo como são etiquetadas as fórmulas utilizadas no método, em que diferentes etiquetas são utilizadas para denotar diferentes "mundos possíveis" do modelo em construção. Contudo, os métodos tableaux propostos nesta dissertação afastam-se desta "perspectiva semântica" tradicional.

Note-se que diferentes lógicas diferem essencialmente na utilização de diferentes regras de

[85] Com base nesta ideia, os métodos tableaux suportam provas/deduções através da confirmação (implícita) de que os respectivos conjuntos de fórmulas não são verdadeiros em nenhum modelo. Os diferentes ramos de um tableau representam a pesquisa de diferentes modelos (ou de diferentes mundos de um modelo).

expansão de tableaux utilizadas. Consequentemente, opta-se aqui por definir tableau com a generalidade necessária para ser utilizada nas diferentes lógicas modais proposicionais discutidas neste capítulo, através da utilização do termo Φ-tableau, em que Φ denomina o conjunto de regras de expansão de tableaux utilizadas.

Formalmente, considera-se a seguinte definição de tableau[86]:

Definição 6.1: Φ-tableau para um conjunto de fórmulas Δ

Seja $\Delta=\{A_1, ..., A_n\}$ (para $n>0$) um conjunto finito e não vazio de fórmulas. Um Φ-tableau para Δ define-se indutivamente como se segue:

(i) a seguinte árvore só com um ramo é um Φ-tableau para Δ:

A_1
A_2
$\vdots$
A_n

(ii) se *T* é um Φ-tableau para Δ e se T^* resulta de *T* por aplicação de uma regra do conjunto de regras de expansão de tableaux denominado por Φ (do modo que se descreverá à frente), então T^* é um Φ-tableau para Δ ♦

Portanto, a aplicação da definição anterior na construção de tableaux para Δ, depende essencialmente do conjunto de regras (denominado por) Φ considerado.

Veja-se então quais as regras utilizadas na construção de Φ_{PL}-tableaux (ou simplesmente, PL-tableaux), i.e. tableaux para a lógica PL (ou para a lógica proposicional, caso não se utilize na linguagem nenhum operador modal[87]):

---

[86] Opta-se nesta dissertação por apresentar a noção de "tableau" com o mesmo grau de formalização adoptado em [Fitting 90], evitando assim recorrer a notações demasiado pesadas (e.g. formalização de árvore) que tornariam árdua a leitura e compreensão deste capítulo.

[87] Note-se que uma vez que será proposto apenas um conjunto Φ_Σ de regras de expansão de tableaux para cada diferente sistema lógico Σ, opta-se frequentemente neste capítulo por denominar o respectivo conjunto de regras de expansão pela denominação desses sistemas lógicos, i.e. apenas por Σ. Nos casos em que se pretender falar genericamente de tableaux, ou em casos em que seja claro qual o sistema lógico utilizado, utilizar-se-á o termo "tableau" em vez de "Φ-tableau". Mais ainda, quando se disser "seja *T* um Φ-tableau" subentende-se que *T* um Φ-tableau para algum conjunto finito e não vazio de fórmulas Δ.

Definição 6.2: Regras de expansão para a lógica PL

As regras de expansão de PL-tableaux são as seguintes:

$$(R\neg\neg)\ \frac{\neg\neg A}{A} \qquad (R\neg True)\ \frac{\neg True}{False} \qquad (R\neg False)\ \frac{\neg False}{True}$$

$$(R\alpha)\ \frac{\alpha}{\alpha_1 \\ \alpha_2} \qquad (R\beta)\ \frac{\beta}{\beta_1 \mid \beta_2}$$

em que α e β denotam, respectivamente, fórmulas conjuntivas e disjuntivas, (equivalentes a uma fórmula) da forma $\alpha_1 \wedge \alpha_2$ e $\beta_1 \vee \beta_2$ (de acordo com a Tabela 6-1 abaixo) ♦

A aplicação das regras da Definição 6.2 na construção de PL-tableaux é feita da seguinte maneira: dado um PL-tableau *T*, seleccione-se um ramo θ e uma fórmula não-atómica B em θ. Se B tiver a forma $\neg\neg$A, então adicione-se mais um nó etiquetado por A no final do ramo θ. De modo similar, se B tiver a forma ¬True, adicione-se False, e se B tiver a forma ¬False, adicione-se True. Se B tiver a forma α, então adicione-se no final do ramo θ mais um nó etiquetado por α_1 e, após esse nó, outro nó etiquetado por α_2. Finalmente, se B tiver a forma β, então adicione-se ao final do ramo θ um nó esquerdo e um nó direito etiquetados respectivamente por β_1 e β_2 (neste caso, o ramo θ origina dois ramos no PL-tableau resultante da aplicação da regra Rβ a *T*). Obtém-se assim um PL-tableau T^* e diz-se que T^* resulta de *T* por aplicação da respectiva regra de expansão de tableaux utilizada.

Note-se que se utiliza aqui a notação Smullyan - α e β (veja-se, e.g., [Smullyan 68]) - para representar fórmulas conjuntivas e disjuntivas de acordo com a Tabela 6-1.

Fórmulas conjuntivas			Fórmulas disjuntivas		
α	α_1	α_2	β	β_1	β_2
$(A \wedge B)$	A	B	$\neg(A \wedge B)$	$\neg A$	$\neg B$
$\neg(A \vee B)$	$\neg A$	$\neg B$	$(A \vee B)$	A	B
$\neg(A \rightarrow B)$	A	$\neg B$	$(A \rightarrow B)$	$\neg A$	B
$(A \leftrightarrow B)$	$(A \rightarrow B)$	$(B \rightarrow A)$	$\neg(A \leftrightarrow B)$	$\neg(A \rightarrow B)$	$\neg(B \rightarrow A)$

Tabela 6-1: α e β - fórmulas e componentes

De acordo com a tabela, α e β são equivalentes a $\alpha_1 \wedge \alpha_2$ e $\beta_1 \vee \beta_2$, respectivamente. A grande

vantagem desta notação reside essencialmente no facto de se poder sintetizar as regras de expansão de tableaux para fórmulas conjuntivas e disjuntivas equivalentes, evitando assim a apresentação de várias regras adicionais para um conjunto vasto de conectivos proposicionais. Por outro lado, evita-se também considerar no processo de aplicação das regras de expansão de tableaux, os aspectos não essenciais da estrutura lógica das fórmulas seleccionadas.

Para ilustrar a aplicação das regras de expansão de tableaux da Definição 6.2, considere-se o seguinte exemplo de um PL-tableau para $\{(A\rightarrow B),(B\rightarrow C),\neg(A\rightarrow C)\}$representado na Figura 6-1 (cuja numeração é apenas utilizada para referenciar os nós do tableau). Nesse tableau, os nós (1), (2) e (3) são introduzidos por aplicação da Definição 6.1(i). Os nós (4) e (5) são introduzidos por aplicação da regra $R\alpha$ ao nó (3), e os nós (6) e (7), por aplicação da regra $R\beta$ ao nó (1). Finalmente, os nós (8) e (9) são introduzidos por aplicação da regra $R\beta$ ao nó (2).

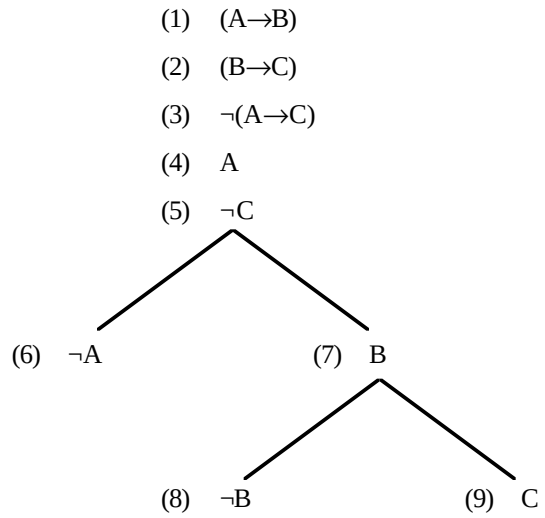


Figura 6-1: Um PL-tableau para {(A→B),(B→C),¬(A→C)}

As definições seguintes, apresentam o modo como os sistemas dedutivos baseados no método tableaux são utilizados para fazer deduções de uma fórmula A a partir de um conjunto de fórmulas Γ (aqui representado por $\Gamma\vdash_{\Phi}A$, em que Φ identifica o conjunto de regras de expansão de tableaux considerado).

Definição 6.3: Ramo fechado e Φ-tableau fechado

Um ramo θ de um Φ-tableau *T* para Δ diz-se fechado sse (i) existe uma fórmula A tal que A e ¬A ocorrem simultaneamente em θ, ou (ii) se False ocorre em θ. Um Φ-tableau *T* para Δ diz-se fechado sse todos os seus ramos são fechados. ♦

Definição 6.4: Deduções com tableaux

$\Gamma \vdash_{\Phi} A$ sse existe um $\Phi \cup \{R\Gamma\}$-tableau fechado para $\{\neg A\}$

em que se considera a regra de expansão de tableaux adicional:

$$(R\Gamma) \quad \frac{}{A} \quad [A \in \Gamma]$$

♦

A regra (RΓ) é considerada adicionalmente apenas para suportar deduções a partir de um conjunto de fórmulas Γ. Note-se que a expressão entre parênteses rectos representa a condição de aplicabilidade da regra (RΓ).

Repare-se que se considera que $\Phi \cup \{R\varnothing\} = \Phi$, uma vez que a regra R∅ nunca pode ser aplicada na construção de tableaux. Deste modo, como caso particular da definição anterior, tem-se: "$\varnothing \vdash_{\Phi} A$ sse existe um Φ-tableau fechado para $\{\neg A\}$".

Repare-se também que a definição acima poderia ter sido apresentada de outro modo, a saber: "$\Gamma \vdash_{\Phi} A$ sse existe um Φ-tableau fechado para $\Gamma \cup \{\neg A\}$". Contudo, esta definição só é equivalente à Definição 6.4 no caso em que Γ é um conjunto finito (veja-se o próximo resultado). No entanto utilizar-se-á nesta dissertação a Definição 6.4 pois evidencia todas as regras utilizadas na construção de tableaux.

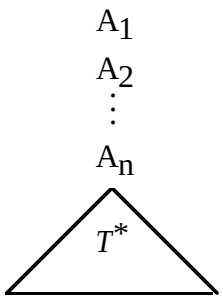
Resultado 6.1: Seja Γ um conjunto finito de fórmulas, então: existe um $\Phi \cup \{R\Gamma\}$-tableau fechado para $\{\neg A\}$ sse existe um Φ-tableau fechado para $\Gamma \cup \{\neg A\}$.

Demonstração: No caso de $\Gamma = \varnothing$, este resultado verifica-se trivialmente. Caso contrário:

(i) da direita para a esquerda: é imediato, pois basta considerar que as fórmulas de Γ no ramo inicial do Φ-tableau para $\Gamma \cup \{\neg A\}$ podem ser justificadas por utilização da regra (RΓ).

(ii) da esquerda para a direita: seja $\Gamma = \{A_1, ..., A_n\}$ e seja *T* um $\Phi \cup \{R\Gamma\}$-tableau fechado para $\{\neg A\}$. Considere-se então a árvore T^* obtida a partir de *T*, mas sem os nós obtidos por aplicação da regra (RΓ) (i.e. sem os nós etiquetados por uma fórmula $A_i \in \Gamma$, para i=1,...,n). Repare-se que a árvore T^* assim obtida tem a fórmula ¬A na raiz. Por outro lado tem o mesmo número de ramos que a árvore *T*. Mais, uma vez que *T* é um tableau fechado, cada um dos ramos da árvore T^* contem fórmulas da forma B e ¬B, excepto nos casos em que se tenha retirado da árvore *T* alguma(s) fórmula(s) $A_i \in \Gamma$, para i=1,...,n.

Então a árvore T^{**}, que se obtém de T^{*} juntando-lhe o ramo inicial (A_1, ..., A_n), i.e.



é um Φ-tableau para $\Gamma \cup \{\neg A\}$, uma vez que: a fórmula $\neg A$ está no ramo inicial de T^{**} (veja-se a Definição 6.1); e na construção de T^{**} pode-se considerar que se aplicam as mesmas regras de expansão aplicadas a T (excepto a regra (RΓ)), uma vez que as fórmulas retiradas de T foram repostas no ramo inicial de T^{**}. Esse ramo inicial "restaura" ainda os ramos da árvore T^{*} que não contêm fórmulas da forma B e $\neg B$. ♦

De acordo com as definições e o teorema anterior, pode-se concluir que $\{(A \rightarrow B),(B \rightarrow C)\} \vdash_{PL} (A \rightarrow C)$, uma vez que o PL-tableau para $\{(A \rightarrow B),(B \rightarrow C),\neg(A \rightarrow C)\}$ ilustrado na Figura 6-1 é fechado. Note-se que o ramo (1,2,3,4,5,6) é fechado devido a (4) e (6); o ramo (1,2,3,4,5,7,8) é fechado devido a (7) e (8); e o ramo (1,2,3,4,5,7,9) é fechado devido a (5) e (9).

Relativamente à prova da correcção forte (relativamente a uma classe de modelos C) de sistemas dedutivos baseados em sistemas de tableaux, i.e. $(\forall\ \Gamma \subseteq \text{Form(L)})\ (\forall\ A \in \text{Form(L)})\ (\Gamma \vdash_{\Phi} A \Rightarrow \Gamma \models_{C} A)$, utilizar-se-á nesta dissertação a mesma estratégia utilizada na literatura (veja-se, e.g. [Fitting 83, Fitting 88]), com base na seguinte noção central: tableau Γ-satisfeito na classe de modelos C (generalização da Definição 2.9).

Definição 6.5: Ramo Γ-satisfeito e tableau Γ-satisfeito na classe de modelos C

Um ramo θ de um tableau T diz-se Γ-satisfeito na classe de modelos C sse o conjunto das fórmulas que o constituem é Γ-satisfeito na classe de modelos C. Um tableau T diz-se Γ-satisfeito na classe de modelos C sse pelo menos um ramo de T é Γ-satisfeito na classe de modelos C.[88] ♦

Com base na noção anterior, a estratégia central da prova de correcção de sistemas de tableaux consiste na constatação de que a aplicação de cada regra de expansão de tableaux preserva a Γ-

[88] Repare-se que esta definição é análoga à definição proposta por [Fitting 90] (veja-se a Definição 3.4.1, página 49). No entanto são efectuadas as adaptações necessárias ao contexto lógico modal utilizado nesta dissertação.

satisfação de um tableau. Note-se que um algoritmo baseado num sistema de tableaux começa sempre com um $\Phi\cup\{R\Gamma\}$-tableau inicial [Definição 6.1(i)] e prossegue a sua construção por aplicação das regras de expansão consideradas em $\Phi\cup\{R\Gamma\}$. Portanto, caso as regras de expansão de tableaux preservem a Γ-satisfação, conclui-se que "se existe um $\Phi\cup\{R\Gamma\}$-tableau fechado para Δ, então Δ não é Γ-satisfeito", a partir do qual a prova de correcção forte de um sistema de tableaux é imediata. Veja-se os próximos resultados:

Resultado 6.2: Sejam Γ, Δ (finito) $\subseteq$ Form(L), seja C uma classe de modelos mínimos e seja Φ um conjunto de regras de expansão de tableaux que preservam a Γ-satisfação na classe de modelos C. Então:

se existe um Φ-tableau fechado para Δ, então Δ não é Γ-satisfeito em C.

Demonstração: Suponha-se, *per absurdum*, que existe um Φ-tableau fechado T para Δ e que Δ é Γ-satisfeito em C. Ora a construção de T foi iniciada com um tableau T^* consistindo apenas de um único ramo cujos nós são etiquetados por Δ [Definição 6.1(i)]. Portanto, uma vez que Δ é Γ-satisfeito em C, também T^* é Γ-satisfeito em C. Por outro lado, todos os tableaux subsequentes construídos a partir de T^* também são Γ-satisfeitos em C (pois por hipótese as regras de expansão de tableaux em Φ preservam a Γ-satisfação na classe de modelos C). Portanto, em particular, o tableau T é Γ-satisfeito em C. Mas, uma vez que T é um tableau fechado, este não pode ser Γ-satisfeito em C [Definição 6.3; Definição 6.5], o que contradiz a conclusão anterior. Consequentemente Δ não pode ser Γ-satisfeito em C. ♦

Resultado 6.3: Seja $\Gamma \subseteq$ Form(L), $A \in$ Form(L), seja C uma classe de modelos mínimos e seja Φ um conjunto de regras de expansão de tableaux que preservam a Γ-satisfação na classe de modelos C. Então:

se $\Gamma \vdash_{\Phi} A$ então $\Gamma \models_{C} A$.

Demonstração: Suponha-se $\Gamma \vdash_{\Phi} A$. Tem-se:

$\Gamma \vdash_{\Phi} A$
sse [Definição 6.4]
existe um $\Phi\cup\{R\Gamma\}$-tableau fechado para $\{\neg A\}$
$\Rightarrow$ [Resultado 6.2] (pois as regras de $\Phi\cup\{R\Gamma\}$ preservam a Γ-satisfação em C)[89]

[89] Repare-se que a regra ($R\Gamma$) preserva sempre a Γ-satisfação (em qualquer classe de modelos C).

{¬A} não é Γ-satisfeito em C
sse [Resultado 2.11]
$\Gamma \models_C A$ ♦

Veja-se, para finalizar esta secção, que o sistema de PL-tableaux é fortemente correcto relativamente a qualquer classe de modelos mínimos C.

Resultado 6.4: Sejam $\Gamma \subseteq$ Form(L) e seja C uma classe de modelos mínimos. Então as regras de expansão de Φ_{PL} preservam a Γ-satisfação na classe de modelos C.

Demonstração: Esta demonstração é análoga à do teorema 3.4.2 em [Fitting 90]. Seja C uma classe de modelos mínimos e seja T um tableau. Suponha-se que T é Γ-satisfeito em C e que se aplica ao ramo θ do tableau T uma das regras de expansão de tableaux em Φ_{PL} obtendo-se assim um tableau T^*. Ora uma vez que T é um tableau Γ-satisfeito em C então existe pelo menos um ramo τ de T que é Γ-satisfeito em C [Definição 6.5]. Veja-se então que também T^* é Γ-satisfeito em C. (no que se segue representar-se-á por Θ, Θ'_1, Θ'_2 os conjuntos das fórmulas que constituem respectivamente os ramos θ, θ'_1 e θ'_2)

Caso θ≠τ: Trivial, já que τ é ainda um ramo de T^*, e portanto T^* permanece Γ-satisfeito em C.

Caso θ=τ : Então θ é Γ-satisfeito em C, i.e. [Definição 6.5]:

$(\exists\, M \in C)\ (\exists\, \gamma \text{ em } M)\ (\forall\, A \in \Theta \cup \Gamma)\ M,\gamma \models A$ (1)

Veja-se então, caso a caso, a aplicação de cada uma das regras de expansão em Φ_{PL} ao tableau T, em que T^* difere de T no ramo θ' (resp. nos ramos θ'_1 e θ'_2, no caso da aplicação da regra (Rβ)):

(i) regra (R¬¬): Neste caso o ramo θ' foi obtido a partir da fórmula da forma ¬¬B em θ, por extensão de θ com a fórmula da forma B, obtendo-se Θ'=Θ∪{B}. Então θ' é Γ-satisfeito em C [(1); Definição 2.5(iv)] e portanto T^* é Γ-satisfeito em C.

(ii) regra (R¬True): Veja-se, por *redução ao absurdo*, que esta regra nunca é aplicada ao ramo θ, caso que se está aqui a considerar. Suponha-se que (R¬True) é aplicada ao ramo θ=τ do tableau T. Nesse caso a fórmula ¬True existiria em θ e portanto $M,\gamma \models$ ¬True. Então $M,\gamma \not\models$ True [Definição 2.5(iv)]. Mas isto contradiz a Definição 2.5(ii).

(iii) regra (R¬False): Neste caso o ramo θ' foi obtido a partir da fórmula da forma ¬False em θ, por extensão de θ com a fórmula da forma True, obtendo-se

$\Theta'=\Theta\cup\{True\}$. Então θ' é Γ-satisfeito em C [(1); Definição 2.5(ii)] e portanto T^* é Γ-satisfeito em C.

(iv) regra ($R\alpha$): Neste caso o ramo θ' foi obtido a partir da fórmula da forma α em θ. Portanto θ foi estendido com as fórmulas α_1 e α_2 para produzir o ramo θ' em T^*, obtendo-se $\Theta'=\Theta\cup\{\alpha_1, \alpha_2\}$. Então θ' é Γ-satisfeito em C [(1); Definição 2.5] e portanto T^* é Γ-satisfeito em C. (pela Definição 2.5 tem-se "$M,\gamma\models\alpha$ sse ($M,\gamma\models\alpha_1$ e $M,\gamma\models\alpha_2$)", para todos os casos de α-fórmulas considerados na Tabela 6-1)

(iv) regra ($R\beta$): Neste caso os ramos θ'_1 e θ'_2 foram obtidos a partir da fórmula da forma β em θ. Portanto θ foi estendido com as fórmulas β_1 ou β_2 para produzir os ramos θ'_1 e θ'_2 em T^*, obtendo-se $\Theta'_1=\Theta\cup\{\alpha_1\}$ e $\Theta'_2=\Theta\cup\{\alpha_2\}$. Então ou θ'_1 é Γ-satisfeito em C ou θ'_2 é Γ-satisfeito em C [(1); Definição 2.5], e portanto T^* é Γ-satisfeito em C. (pela Definição 2.5 tem-se "$M,\gamma\models\beta$ sse ($M,\gamma\models\beta_1$ ou $M,\gamma\models\beta_2$)", para todos os casos de β-fórmulas considerados na Tabela 6-1) ♦

Resultado 6.5: Seja Γ, Δ (finito)$\subseteq$ Form(L), A$\in$ Form(L) e seja C uma classe de modelos mínimos. Então:

(1) se existe um Φ_{PL} -tableau fechado para Δ, então Δ não é Γ-satisfeito em C.

(2) se $\Gamma\vdash_{\Phi_{PL}} A$ então $\Gamma\models_C A$.

Demonstração: Imediato [Resultado 6.2; Resultado 6.3; Resultado 6.4]. ♦

A completude forte do sistema de PL-tableaux pode ser demonstrada de modo análogo à efectuada em [Fitting 90] (capítulo 3) para sistemas de tableaux proposicionais. Contudo não se mencionará aqui a técnica aí empregue, uma vez que não se provou a completude para os sistemas de tableaux propostos à frente nesta dissertação.

Note-se que uma vez demonstrada a correcção forte e a completude forte de um sistema de Φ-tableaux em relação a uma classe de modelos C, passa-se a dispor de um método sintáctico para a caracterização da noção de consequência lógica nessa classe de modelos. No caso de também se ter uma axiomatização à Hilbert, fortemente correcta e completa face à mesma classe de modelos, passa-se a dispor de dois sistemas dedutivos, de natureza puramente sintáctica, equivalentes. Qual a vantagem em possuir os dois ? A vantagem reside em que cada um deles é mais adequado para certas facetas do problema abordado. A discussão de quais os princípios lógicos que caracterizam os conceitos de acção é, numa primeira fase, claramente mais simples de efectuar numa base axiomática. Por outro lado, é muito mais fácil automatizar deduções sobre um sistema de tableaux do que num sistema axiomático à Hilbert. E isto conduz a um aspecto ainda não referido, mas que

provavelmente já surgiu ao leitor desta dissertação. É que a prova de que $\Gamma \vdash_{\Phi} A$ apenas exige a existência de "um" $\Phi \cup \{R\Gamma\}$-tableau fechado para $\{\neg A\}$, mas é preciso encontrá-lo. Isto é, é preciso definir um algoritmo que em tempo finito termine a sua execução garantindo tal existência, ou a sua não existência. Ora, de acordo com a Definição 6.1, tal não é à partida completamente trivial, pois por exemplo nada impede (de acordo com essa definição) que na expansão de um mesmo ramo, uma mesma regra de expansão seja sucessivamente aplicada à mesma fórmula (um tableau diz-se "estrito" - "strict" - se na expansão de um mesmo ramo, a nenhuma fórmula foi aplicada mais do que uma das regras de expansão). Poder-se-á questionar porque não impedir tal (obrigando a que os tableaux tenham que ser estritos), e a resposta é simples: é que (como se refere em [Fitting 72]) para algumas lógicas (que não a lógica proposicional) tal impediria a respectiva completude. É assim fundamental procurar adequadas estratégias de implementação de um sistema de tableaux, que restrinjam (guiem) o modo de aplicação das suas regras, de modo a tornar estes "realizáveis" computacionalmente, procurando não pôr em causa a sua completude. (Note-se que tais restrições de implementação não afectam obviamente a correcção de um sistema de tableaux.)

Nesta tese, como já se referiu, não foi possível provar a completude dos sistemas de tableaux propostos (Embora se tenha provado a sua correcção, como seria fundamental para garantir que as deduções efectuadas nesses sistemas estão correctas). Assim, não se pode colocar a questão de como certas restrições de implementação afectam a sua eventual completude. De qualquer modo, tais restrições/opções de implementação estiveram sempre presentes, conduzindo até à formulação de certo tipo de tableaux em alternativa a outros porventura mais simples (mas conduzindo a uma menor eficiência na implementação), como se ilustrará e discutirá nas próximas secções.

6.2 Método tableaux para Sistemas de Lógica Modal Clássica Proposicional

O método tableaux anterior pode ser estendido a sistemas de lógica modal clássica proposicional não-normais (em que se considera genericamente o operador modal $\Box$), por utilização de subconjuntos das seguintes regras de expansão de $\Phi \cup \{R\Gamma\}$-tableaux, com $(R\Gamma) \not\in \Phi$[90] (em que as expressões dentro dos parênteses rectos representam, caso existam, condições adicionais de aplicabilidade das regras nos $\Phi \cup \{R\Gamma\}$-tableaux em que estas forem utilizadas):

(RrE) $\dfrac{\Box A,\ \neg \Box B}{\text{False}}$ $[\vdash_{\Phi} A \leftrightarrow B]$

[90] Assumir-se-á que está implícito em tudo o que se segue que $(R\Gamma) \not\in \Phi$.

(RT) $\frac{\Box A}{A}$

(RNo) $\frac{\Box A}{\text{False}}$ $[\vdash_{\Phi} A]$

(RNoF) $\frac{\Box A}{\text{False}}$ $[\vdash_{\Phi} \neg A]$

(RC) $\frac{\neg \Box(C_1 \wedge ... \wedge C_n),\ \Box A}{\neg \Box (Ci_1 \wedge ... \wedge Ci_k)}$ $[\vdash_{\Phi} A \leftrightarrow (Ci_{k+1} \wedge ... \wedge Ci_n)]$, (para $1 \leq k < n$)

(em que $\{i_1, ..., i_n\}$ é uma permutação de $\{1, ..., n\}$)

(RD) $\frac{\Box A,\ \Box B}{\text{False}}$ $[\vdash_{\Phi} A \leftrightarrow \neg B]$

(RDC) $\frac{\Box\neg(C_1 \wedge ... \wedge C_n),\ \Box A}{\neg \Box (Ci_1 \wedge ... \wedge Ci_k)}$ $[\vdash_{\Phi} A \leftrightarrow (Ci_{k+1} \wedge ... \wedge Ci_n)]$, (para $1 \leq k < n$)

(em que $\{i_1, ..., i_n\}$ é uma permutação de $\{1, ..., n\}$)

Note-se que se opta aqui por definir as regras de expansão para $\Phi \cup \{R\Gamma\}$-tableaux, com $\{R\Gamma\} \notin \Phi$, a fim de salientar que as condições adicionais de aplicabilidade dessas regras - entre parênteses rectos - referem a utilização de Φ-tableaux, mas sem a regra $(R\Gamma)$ (i.e. de acordo com a Definição 6.4, sem a regra de expansão adicional para considerar deduções a partir de um conjunto de hipóteses Γ): observe-se que elas têm a forma $[\vdash_{\Phi} ...]$ e não $[\Gamma \vdash_{\Phi} ...]$.

Repare-se que regras com duas premissas, da forma

$\frac{A,\ B}{C}$ $[\vdash_{\Phi} D]$,

podem ser aplicadas na construção de um $\Phi \cup \{R\Gamma\}$-tableau nos casos em que um seu ramo θ contenha fórmulas da forma A e B e se verifica $\vdash_{\Phi} D$. Nestes casos adiciona-se um nó etiquetado por C no final do ramo θ.

A formulação das regras anteriores é motivada pelos axiomas esquemas e regras de inferência utilizados nas axiomatizações das lógicas modais para as quais se pretende aqui propor sistemas de tableaux. As regras de expansão de tableaux (RrE), (RT), (RNo), (RNoF) e (RC) estão

associadas, respectivamente, à regra de inferência (rE) e aos axiomas-esquemas (T), (No), (NoF) e (C). E as regras (RD) e (RDC) ao axioma-esquema (D).

A regra (RrE) é motivada pelo facto de se poder inferir de (rE) a regra de prova (na terminologia da nota de rodapé 11) "de $\vdash A \leftrightarrow B$ infere-se $\vdash (\Box A \wedge \neg \Box B) \rightarrow False$". A regra (RT) sai de forma óbvia do esquema (T). A regra (RNo) pelo facto de se poder inferir de (rE) e (No) a regra de prova "de $\vdash A$ infere-se $\vdash \Box A \rightarrow False$". A regra (RNoF) pelo facto de se poder inferir de (rE) e (NoF) a regra de prova "de $\vdash \neg A$ infere-se $\vdash \Box A \rightarrow False$". A regra (RC) pelo facto de se poder inferir de (rE) e (C) a regra de prova "de $\vdash A \leftrightarrow D$ infere-se $\vdash (\neg \Box (D \wedge B) \wedge \Box A) \rightarrow \neg \Box B$". E as regras (RD) e (RDC) pelo facto de se poder inferir de (rE) e (D) a regra de prova "de $\vdash A \leftrightarrow \neg B$ infere-se $\vdash (\Box A \wedge \Box B) \rightarrow False$" e de se poder inferir de (rE), (D) e (C) a regra de prova "de $\vdash A \leftrightarrow D$ infere-se $\vdash (\Box \neg (D \wedge B) \wedge \Box A) \rightarrow \neg \Box B$" (a necessidade da regra (RDC) será salientada à frente).

Dada a motivação utilizada para a regra (RC), repare-se que quando se detecta no tableau uma fórmula da forma $\neg \Box C$ (durante o processo de construção de tableaux) e se pretende considerar a aplicação da regra de prova "de $\vdash A \leftrightarrow D$ infere-se $\vdash (\neg \Box (D \wedge B) \wedge \Box A) \rightarrow \neg \Box B$" é necessário que C seja convertida numa fórmula da forma $D \wedge B$ equivalente para a qual exista também no tableau uma fórmula da forma $\Box A$ tal que A seja equivalente a D. Para resolver este problema opta-se por converter a fórmula C na "forma clausal proposicional" $(C_1 \wedge ... \wedge C_n)$ [91] e por procurar uma fórmula D da forma $(Ci_{k+1} \wedge ... \wedge Ci_n)$ (em que $\{i_1, ..., i_n\}$ é uma permutação de $\{1, ..., n\}$). Daí a formulação da regra (RC).

Utiliza-se uma estratégia análoga quando se detecta uma fórmula da forma $\Box C$ e se pretende considerar a aplicação da regra de prova "de $\vdash A \leftrightarrow D$ infere-se $\vdash (\Box (\neg (D \wedge B)) \wedge \Box A) \rightarrow \neg \Box B$". Neste caso é necessário que $\neg C$ seja convertida numa fórmula da forma $D \wedge B$ equivalente para a qual exista também no tableau uma fórmula da forma $\Box A$ tal que A seja equivalente a D. Para resolver este problema opta-se por converter a fórmula $\neg C$ na "forma clausal proposicional" $(C_1 \wedge ... \wedge C_n)$ e por procurar uma fórmula D da forma $(Ci_{k+1} \wedge ... \wedge Ci_n)$. Daí a formulação da regra (RDC).

As conversões mencionadas podem ser explicitadas no método tableaux por utilização das regras de expansão

$$(R\neg\Box\,cl) \qquad \frac{\neg \Box A}{\neg \Box\, clausal(A)}$$

[91] Isto é, cada C_i em $(C_1 \wedge ... \wedge C_n)$ é uma disjunção de literais (i.e. símbolos proposicionais, a negação de símbolos proposicionais, True ou False) ou de fórmulas da forma $\Box A$ e $\neg \Box A$. O termo "forma clausal proposicional" é aqui utilizado para evidenciar que fórmulas da forma $\Box A$ e $\neg \Box A$ desempenham o mesmo papel que os literais. Por exemplo, a fórmula $((\neg p1 \vee p2) \wedge (p3 \vee \Box (p1 \rightarrow p3)))$ está na forma clausal proposicional.

(R□cl) $\dfrac{\Box A}{\Box \neg \text{clausal}(\neg A)}$

em que clausal(A) representa uma fórmula na "forma clausal proposicional" e equivalente à fórmula A. Na implementação destas regras utilizou-se o algoritmo (devidamente adaptado) proposto em [Fitting 90], páginas 26-27, para converter uma fórmula A numa fórmula equivalente na "forma clausal proposicional".

A formulação das regras de expansão de tableaux aqui apresentadas foi guiada pela necessidade de assegurar a decidibilidade do método proposto, impondo, sempre que possível, que as fórmulas adicionadas a um tableau, através das regras de expansão, apresentassem complexidade menor - i.e. menor número de conectivos lógicos, sub-fórmulas e operadores modais - do que as fórmulas utilizadas como premissas dessas regras. Daí que não se tenha optado por exemplo pela seguinte formulação da regra de expansão de tableaux associada ao esquema (C):

(RC') $\dfrac{\Box A,\ \Box B}{\Box (A \wedge B)}$

Por outro lado, a formulação de algumas das regras anteriores reflecte também preocupações relacionadas com a implementação do método proposto. Por exemplo, em vez da regra (RC), poder-se-ia propor a regra

(RC'') $\dfrac{\neg \Box (C_1 \wedge \ldots \wedge C_n)}{\neg \Box (Ci_1 \wedge \ldots \wedge Ci_k) \mid \neg \Box (Ci_{k+1} \wedge \ldots \wedge Ci_n)}$, (para $1 \leq k < n$)

No entanto tentou-se evitar, sempre que possível, a definição de regras cuja aplicação provocasse a ramificação de um tableau. De facto, na implementação adoptada, cada ramificação provoca a duplicação do espaço em memória necessário à representação de um tableau.

Os sistemas de tableaux a seguir propostos utilizam subconjuntos das regras de expansão anteriores, conjuntamente com as regras da Definição 6.2 e as abreviaturas da Tabela 6-1. Importa porém esclarecer dois aspectos diferentes relativamente ao tratamento de fórmulas "proposicionais" (veja-se a nota de rodapé 21, página 20) durante o processo de construção de tableaux. O tratamento de fórmulas cujo operador principal é um conectivo booleano é feito por aplicação das regras da Definição 6.2 e pelas regras de abreviatura da Tabela 6-1 (em que os As e Bs da tabela podem ser quaisquer fórmulas de L). Por outro lado, qualquer fórmula "proposicional" no âmbito de um □ (ou de ¬□) pode ser manipulável tal como na lógica proposicional, mas mantendo-se no

âmbito de um $\Box$ (ou de $\neg\Box$), em virtude da correcção forte da regra (rE). De facto tal correcção forte (mais a correcção de $\vdash_{\Phi_{PL}}$) garante que se poderia considerar regras de expansão de tableaux da forma

$$(R\Box) \qquad \frac{\Box A}{\Box B} \quad [\vdash_{\Phi_{PL}} A\leftrightarrow B]$$

$$(R\neg\Box) \qquad \frac{\neg\Box A}{\neg\Box B} \quad [\vdash_{\Phi_{PL}} A\leftrightarrow B]$$

das quais ($R\Box cl$) e ($R\neg\Box cl$) são casos particulares, já que $\vdash_{\Phi_{PL}} A\leftrightarrow clausal(A)$ e $\vdash_{\Phi_{PL}} A\leftrightarrow\neg clausal(\neg A)$ e que sistemas de PL-tableaux são fortemente completos (veja-se [Fitting 90], capítulo 3).

Na definição seguinte propõem-se conjuntos de regras de expansão de tableaux para automatizar deduções de alguns dos sistemas de lógica modal clássica proposicional não-normais.

Definição 6.6: Sistemas de tableaux para lógicas modais clássicas proposicionais

Na tabela estão indicados os conjuntos de regras de expansão de tableaux Φ_Σ a considerar para os sistemas de lógica modal clássica proposicional Σ indicados.

Sistemas de lógica modal clássica Σ	Regras de expansão de tableaux Φ_Σ
E	PL$\cup${RrE}
ET	PL$\cup${RrE, RT}
EC	PL$\cup${RrE, RC, R$\neg\Box$cl}
ENo	PL$\cup${RrE, RNo}
ENoF	PL$\cup${RrE, RNoF}
ED	PL$\cup${RrE, RD}
ECT	PL$\cup${RrE, RC, RT, R$\neg\Box$cl}
ECTNo	PL$\cup${RrE, RC, RT, RNo, R$\neg\Box$cl}
ECTNoF	PL$\cup${RrE, RC, RT, RNoF, R$\neg\Box$cl}
ECNoF	PL$\cup${RrE, RC, RD, RDC, RNoF, R$\neg\Box$cl, R$\Box$cl}
ECNoNoF	PL$\cup${RrE, RC, RD, RDC, RNo, RNoF, R$\neg\Box$cl, R$\Box$cl}

♦

Repare-se que numa lógica do tipo ECNoF são necessárias as regras adicionais (RD) e (RDC)[92]. A razão é a seguinte: apesar do esquema (D) ser obtido de (C) e (NoF) nas lógicas modais clássicas, de facto (RC) não gera as fórmulas às quais a regra (RNoF) deve ser aplicada. Repare-se também que na lógica do tipo ECT, o esquema (D) também é obtido (desta vez de (C) e (T)). No entanto nessa classe de sistemas não é necessário considerar (RD) e (RDC), uma vez que a regra (RT) permite detectar contradições ao nível proposicional/tautológico.

Para ilustrar a utilização dos sistemas de tableaux da Definição 6.6, considere-se os seguintes exemplos de tableaux - ilustrados nas figuras seguintes, e em que se omitem os tableaux associados às condições adicionais de aplicabilidade das regras de expansão utilizadas (pois tratar-se-iam nestes exemplos de meros tableaux proposicionais) - que suportam as deduções: $\{\Box(p2\wedge p4),\Box(p1\wedge p3)\}\vdash_{EC}\Box(p1\wedge p2\wedge p3\wedge p4)$, $\{\Box(p1\wedge p2)\}\vdash_{ED}\neg\Box\neg(p1\wedge p2)$, $\{\Box p1,\Box p2\}\vdash_{ED}\neg\Box\neg(p1\wedge p2)$ e $\{\Box p1,\Box p2\}\vdash_{ET}\neg\Box\neg(p1\wedge p2)$.

Na Figura 6-2 apresenta-se um EC-tableau fechado para $\{\Box(p2\wedge p4),\Box(p1\wedge p3),\neg\Box(p1\wedge p2\wedge p3\wedge p4)\}$: os nós (1), (2) e (3) são introduzidos por aplicação da Definição 6.1(i); o nó (4) é introduzido por aplicação da regra (RC) aos nós (2) e (3). Note-se que a aplicação da regra (RC) é ilustrativa da necessidade de utilização de permutações na formulação da regra (RC). O único ramo obtido é fechado devido a (1) e (4). Consequentemente $\{\Box(p2\wedge p4),\Box(p1\wedge p3)\}\vdash_{EC}\Box(p1\wedge p2\wedge p3\wedge p4)$ [Resultado 6.1; Definição 6.4].

(1) $\Box(p2\wedge p4)$

(2) $\Box(p1\wedge p3)$

(3) $\neg\Box(p1\wedge p2\wedge p3\wedge p4)$

(4) $\neg\Box(p2\wedge p4)$

Figura 6-2: Um EC-tableau para $\{\Box(p2\wedge p4),\Box(p1\wedge p3),\neg\Box(p1\wedge p2\wedge p3\wedge p4)\}$

Na Figura 6-3 apresenta-se um ED-tableau fechado para $\{\Box(p1\wedge p2),\neg\neg\Box\neg(p1\wedge p2)\}$: os nós (1) e (2) são introduzidos por aplicação da Definição 6.1(i); o nó (3) é introduzido por aplicação da regra (R$\neg\neg$) ao nó (2); o nó (4) é introduzido por aplicação da regra (RD) aos nós (1) e (3). Este tableau é fechado devido a (4). Consequentemente $\{\Box(p1\wedge p2)\}\vdash_{ED}\neg\Box\neg(p1\wedge p2)$ [Resultado

[92] Repare-se que a regra seguinte aplicada a $\Phi\cup\{R\Gamma\}$-tableaux

(RD) $\dfrac{\Box A}{\text{False}}$ $[\Gamma\vdash_{\Phi}\Box\neg A]$

permitiria substituir as regras (RD) e (RDC). No entanto não foi provado que esta regra preserva a Γ-satisfação de tableaux, que como se viu é necessário para se provar a correcção de sistemas de tableaux utilizando esta regra.

6.1; Definição 6.4].

(1) $\Box(p1 \wedge p2)$

(2) $\neg\neg\Box\neg(p1 \wedge p2)$

(3) $\Box\neg(p1 \wedge p2)$

(4) False

Figura 6-3: Um ED-tableau para $\{\Box(p1 \wedge p2), \neg\neg\Box\neg(p1 \wedge p2)\}$

Na Figura 6-4 apresentam-se um ED-tableau e um ET-tableau para $\{\Box p1, \Box p2, \neg\neg\Box\neg(p1 \wedge p2)\}$, ambos fechados. No ED-tableau (o da esquerda) os nós (1), (2) e (3) são introduzidos por aplicação da Definição 6.1(i); o nó (4) é introduzido por aplicação da regra (R$\neg\neg$) ao nó (3); o nó (5) é introduzido por aplicação da regra (RDC) aos nós (1) e (4). Este tableau é fechado devido a (2) e (5). Consequentemente $\{\Box p1, \Box p2\} \vdash_{ED} \neg\Box\neg(p1 \wedge p2)$. No ET-tableau os nós (1'), (2'), (3') e (4') são introduzidos com as mesmas justificações do ED-tableau anterior; os nós (5'), (6') e (7') são introduzidos por aplicação da regra (RT), respectivamente, aos nós (1'), (2') e (4'); Os nós (8') e (9') são obtidos por aplicação da regra (Rβ) ao nó (7'). Este tableau é fechado devido a (5') e (8') - no ramo (1',2',3',4',5',6',7',8') - e devido a (6') e (9') - no ramo (1',2',3',4',5',6',7',9'). Consequentemente $\{\Box p1, \Box p2\} \vdash_{ET} \neg\Box\neg(p1 \wedge p2)$.

(1) $\Box p1$

(2) $\Box p2$

(3) $\neg\neg\Box\neg(p1 \wedge p2)$

(4) $\Box\neg(p1 \wedge p2)$

(5) $\neg\Box p2$

(1') $\Box p1$

(2') $\Box p2$

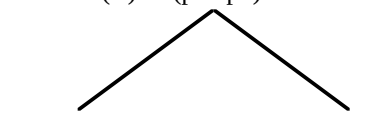
(3') $\neg\neg\Box\neg(p1 \wedge p2)$

(4') $\Box\neg(p1 \wedge p2)$

(5') $p1$

(6') $p2$

(7') $\neg(p1 \wedge p2)$



(8') $\neg p1$ (9') $\neg p2$

Figura 6-4: Um ED-tableau e um ET-tableau para $\{\Box p1, \Box p2), \neg\neg\Box\neg(p1 \wedge p2)\}$

Finaliza-se esta secção demonstrando que cada sistema de tableaux Φ_Σ proposto na Definição 6.6 é fortemente correcto relativamente à classe de modelos mínimos C_Σ apresentada na Tabela 6-2, i.e. $(\forall\, \Gamma \subseteq \text{Form(L)})\ (\forall\, A \in \text{Form(L)})\ (\Gamma \vdash_{\Phi_\Sigma} A \Rightarrow \Gamma \models_{C_\Sigma} A)$.

Repare-se que cada classe de modelos $\mathcal{C}_{\Sigma}$ mencionada na Tabela 6-2 é definida pelo conjunto de modelos mínimos satisfazendo as condições que tornam válidos os axiomas-esquema utilizados na axiomatização do sistema Σ (veja-se a Tabela 2-2 da página 26).

Sistemas Φ_{Σ}	Classes de modelos $\mathcal{C}_{\Sigma}$
Φ_{E}	$\mathcal{C}_{E} = \{M: M$ é um modelo mínimo$\}$
Φ_{ET}	$\mathcal{C}_{ET} = \{M: M \in \mathcal{C}_{E}$ e verifica a condição (t)$\}$
Φ_{EC}	$\mathcal{C}_{EC} = \{M: M \in \mathcal{C}_{E}$ e verifica a condição (c)$\}$
Φ_{ENo}	$\mathcal{C}_{ENo} = \{M: M \in \mathcal{C}_{E}$ e verifica a condição (no)$\}$
Φ_{ENoF}	$\mathcal{C}_{ENoF} = \{M: M \in \mathcal{C}_{E}$ e verifica a condição (nof)$\}$
Φ_{ED}	$\mathcal{C}_{ED} = \{M: M \in \mathcal{C}_{E}$ e verifica a condição (d)$\}$
Φ_{ECT}	$\mathcal{C}_{ECT} = \{M: M \in \mathcal{C}_{E}$ e verifica as condições (c) e (t)$\}$
Φ_{ECTNo}	$\mathcal{C}_{ECTNo} = \{M: M \in \mathcal{C}_{E}$ e verifica as condições (c), (t) e (no)$\}$
Φ_{ECTNoF}	$\mathcal{C}_{ECTNoF} = \{M: M \in \mathcal{C}_{E}$ e verifica as condições (c), (t) e (nof)$\}$
Φ_{ECNoF}	$\mathcal{C}_{ECNo} = \{M: M \in \mathcal{C}_{E}$ e verifica as condições (c) e (nof)$\}$
$\Phi_{ECNoNoF}$	$\mathcal{C}_{ECNoNoF} = \{M: M \in \mathcal{C}_{E}$ e verifica as condições (c), (no) e (nof)$\}$

Tabela 6-2: Sistemas de tableaux Φ_{Σ} e classes de modelos $\mathcal{C}_{\Sigma}$ em que são fortemente correctos

Por outro lado, cada um dos sistemas Σ mencionados é determinado pela classe de modelos $\mathcal{C}_{\Sigma}$ (veja-se [Chellas 80]). Portanto, a demonstração da correcção forte de cada sistema de tableaux Φ_{Σ} proposto permitirá concluir ainda $(\forall\, \Gamma \subseteq \mathrm{Form(L)})\ (\forall\, A \in \mathrm{Form(L)})\ (\Gamma \vdash_{\Phi\Sigma} A \Rightarrow \Gamma \vdash_{\Sigma} A)$, o que assegura que com o sistema de tableaux Φ_{Σ} não se obtêm deduções incorrectas (face às pretendidas no correspondente sistema axiomático Σ).

Na demonstração da correcção forte destes sistemas de tableaux utilizar-se-á a estratégia referida na secção anterior. No entanto, dada a existência das condições de aplicabilidade de algumas das regras de expansão de tableaux aqui propostas, a demonstração de que "cada regra de expansão de tableaux preserva a satisfação de um tableau" necessita de ser feita por indução na complexidade - número de encaixes modais ("embeded modalities") - das fórmulas dos tableaux aos quais essas regras são aplicadas.

As próximas definições e teoremas apresentam os resultados preliminares necessários à demonstração da correcção forte.

Definição 6.7: Número de encaixes modais de fórmulas e de conjuntos finitos de fórmulas

Seja Σ um sistema de lógica modal proposicional. A função ne: Form(L) $\rightarrow \mathbb{N}_0$ estabelece o número de encaixes modais das fórmulas de Form(L) e define-se indutivamente como se segue:

(i) $ne(p_i) = 0$, para i = 1, 2, ...
(ii) $ne(True) = 0$
(iii) $ne(False) = 0$
(iv) $ne(\neg A) = ne(A)$
(v) $ne(A \wedge B) = máximo(\{ne(A), ne(B)\})$
(vi) $ne(A \vee B) = máximo(\{ne(A), ne(B)\})$
(vii) $ne(A \rightarrow B) = máximo(\{ne(A), ne(B)\})$
(viii) $ne(A \leftrightarrow B) = máximo(\{ne(A), ne(B)\})$
(ix) $ne(\Box A) = ne(A)+1$

A função ne^*: $\{\Gamma: \Gamma \subseteq 2^{Form(L)}$ e Γ finito$\} \rightarrow \mathbb{N}_0$ estabelece o número de encaixes modais de conjuntos de fórmulas de Form(L) e define-se como se segue:

(x) $ne^*(\varnothing) = 0$
(xi) $ne^*(\Gamma) = máximo(\{ne(A): A \in \Gamma\})$, se $\Gamma \neq \varnothing$ ♦

Repare-se que, de acordo a definição anterior, no caso de se considerar a linguagem L_{PL} tem-se: $(\forall\ A \in Form(L_{PL}))\ ne(A)=0$. Note-se também que da definição anterior sai trivialmente: "se $X \subseteq Y$, então $ne^*(X) \leq ne^*(Y)$".

Definição 6.8: Número de encaixes modais de um tableau

Seja *T* um tableau (para algum conjunto de fórmulas finito e não vazio Δ) e θ um ramo de *T*. Então NE(θ) e NE(*T*) denotam, respectivamente, o número de encaixes modais das fórmulas do ramo θ e do tableau *T*, e são definidos por:

(i) $NE(\theta) = ne^*(\Theta)$, em que Θ denota o conjunto das fórmulas que constituem θ;
(ii) NE(*T*) = máximo({NE(γ): γ é um ramo do tableau *T*}) ♦

Note-se que NE(θ) e NE(*T*) estão bem definidos, uma vez que Θ (para cada ramo θ) é finito e o número de ramos de um tableau também é finito.

Os Resultados seguintes estabelecem limites para o número de encaixes modais de tableaux. Como se pode observar, apenas a regra de expansão (RΓ) contribui para aumentar o número de encaixes modais de um tableau durante a sua construção.

Lema 6.1: Seja $A \in$ Form(L). Então:

ne(A) = ne(clausal(A)).

Demonstração: Como foi dito anteriormente, clausal(A) representa uma fórmula na "forma clausal proposicional" e equivalente a A, obtida de A por utilização do algoritmo proposto em [Fitting 90], páginas 26-27 (em que sub-fórmulas de A da forma $\Box B$ e $\neg\Box B$ desempenham o mesmo papel que os literais). Utilizando esse algoritmo é imediato demonstrar que ne(A)=ne(clausal(A)). ♦

Resultado 6.6: Seja T um $\Phi\cup\{R\Gamma\}$-tableau e T^* o tableau obtido por aplicação de uma das regras de expansão de $\Phi\cup\{R\Gamma\}$. Então:

(i) $NE(T) \leq NE(T^*)$;

(ii) $NE(T) = NE(T^*)$, caso a regra de expansão utilizada não seja a regra (RΓ).

Demonstração: Seja T um $\Phi\cup\{R\Gamma\}$-tableau e T^* o tableau obtido por aplicação de uma das regras de expansão de $\Phi\cup\{R\Gamma\}$. Então o tableau T^* difere do tableau T apenas num ramo de T, ao qual foram adicionadas fórmulas (eventualmente dois ramos no caso da utilização da regra (Rβ)) - veja-se [Definição 6.1] e o processo de construção de tableaux descrito na página 127.

(i) $NE(T) \leq NE(T^*)$: imediato, uma vez que "se $X \subseteq Y$, então $ne^*(X) \leq ne^*(Y)$" [Definição 6.8].

(ii) $NE(T) = NE(T^*)$, caso a regra de expansão utilizada não seja a regra (RΓ): imediato, uma vez que, com a excepção de (RΓ), nenhuma das regras de expansão de tableaux anteriormente propostas - i.e. (R¬¬), (R¬True), (R¬False), (Rα), (Rβ), (RrE), (RT), (RNo), (RNoF), (RC), (RD), (RDC), (R□cl) e (R¬□cl) - adicionam ao tableau T^* fórmula(s) com número de encaixes modais superiores aos já existentes no tableau T. No caso das regras (R□cl) e (R¬□cl) considere-se o Lema 6.1 ♦

Resultado 6.7: Seja Σ um sistema de lógica modal, Δ (finito) $\subseteq$ Form(L) e T um Φ_Σ-tableau para Δ. Então:

$ne^*(\Delta) = NE(T)$

Demonstração: Seja Σ um sistema de lógica modal, Δ (finito)$\subseteq$Form(L) e T um Φ_{Σ}-tableau para Δ. Então T é obtido por aplicação das regras de Φ a partir de um Φ_{Σ}-tableau para Δ inicial T^{*}, consistindo este na árvore só com um ramo e contendo exactamente todas as fórmulas de Δ (veja-se [Definição 6.1] e o processo de construção de tableaux descrito na página 127). Portanto $ne^{*}(\Delta)=NE(T^{*})$ [Definição 6.8]. E uma vez que regra $(R\Gamma)\notin\Phi_{\Sigma}$ ($\Gamma\subseteq$Form(L)), tem-se $ne^{*}(\Delta)=NE(T)$ [Resultado 6.6(ii)]. ♦

Os próximos resultados destinam-se a demonstrar que cada sistema de tableaux Φ_{Σ} proposto na Definição 6.6 é fortemente correcto relativamente à classe de modelos mínimos C_{Σ} apresentada na Tabela 6-2.

Lema 6.2 (revisão): Seja Δ (finito)$\subseteq$Form(L), A$\in$Form(L) e seja Σ um dos sistemas mencionados na Definição 6.6. Então:

(1) as regras de Φ_{PL} preservam a $\varnothing$-satisfação na classe de modelos C_{Σ}.
(2) se existe um Φ_{PL}-tableau fechado para Δ, então Δ não é $\varnothing$-satisfeito em C_{Σ}.
(3) se $\vdash_{\Phi_{PL}} A$ então $\models_{C\Sigma} A$.

Demonstração: Imediato [Resultado 6.2; Resultado 6.3; Resultado 6.4]. ♦

O resultado seguinte reúne demonstrações análogas às do Resultado 6.4 e do Resultado 6.5, mas agora aplicados aos sistemas de tableaux propostos para os sistemas E, ET, EC, ENo, ENoF, ED, ECTNo, ECTNoF, ECNoF e ECNoNoF. Esta concentração de resultados deve-se ao facto de a demonstração de que as regras de Φ_{Σ} preservam a satisfação na classe de modelos C_{Σ} (análogo do Resultado 6.4) necessitar do resultado "se $\vdash_{\Phi\Sigma} A$ então $\models_{C\Sigma} A$" (análogo ao Resultado 6.5, que como se viu depende do Resultado 6.4). Este problema deve-se às condições de aplicabilidade das regras de expansão de tableaux propostas nesta secção.

Resultado 6.8: Seja Δ (finito)$\subseteq$Form(L), A$\in$Form(L) e seja Σ um dos sistemas mencionados na Definição 6.6. Então:

(1) as regras de Φ_{Σ} preservam a $\varnothing$-satisfação na classe de modelos C_{Σ}.
(2) se existe um Φ_{Σ}-tableau fechado para Δ, então Δ não é $\varnothing$-satisfeito em C_{Σ}.
(3) se $\vdash_{\Phi\Sigma} A$ então $\models_{C\Sigma} A$.

Demonstração: A estratégia que será seguida nesta demonstração é a de demonstrar estes resultados por indução no número de encaixes modais $ne^*(\Delta)$ [Resultado 6.7] de um Φ_Σ-tableau *T* fechado para Δ. Provar-se-á por indução em $n\geq 0$ os seguintes resultados equivalentes a (1), (2) e (3):

(1') $(\forall\ n\geq 0)\ D_1(n)$;

(2') $(\forall\ n\geq 0)\ D_2(n)$;

(3') $(\forall\ n\geq 0)\ D_3(n)$

onde $D_1(n)$, $D_2(n)$ e $D_3(n)$ são:

$D_1(n)$: qualquer que seja Δ, se $n=ne^*(\Delta)$ então as regras de expansão de Φ_Σ-tableaux para Δ preservam a $\varnothing$-satisfação na classe de modelos $\mathcal{C}_\Sigma$.

$D_2(n)$: qualquer que seja Δ, se $n=ne^*(\Delta)$ e existe um Φ_Σ-tableau fechado para Δ, então Δ não é $\varnothing$-satisfeito em $\mathcal{C}_\Sigma$.

$D_3(n)$: qualquer que seja a fórmula A, se $n=ne^*(\{\neg A\})$ e $\vdash_{\Phi_\Sigma} A$ então $\models_{\mathcal{C}_\Sigma} A$.

Veja-se então a demonstração de (1'), (2') e (3'). Considera-se aqui o caso em que Σ é ECNoNoF e a classe de modelos $\mathcal{C}_{ECNoNoF}=\{M: M\in \mathcal{C}_E$ e verifica as condições (c), (no) e (nof)$\}$. Repare-se que os restantes sistemas Σ em discussão requerem uma demonstração muito semelhante, sendo apenas necessário que se considerem diferentes classes de modelos verificando um conjunto diferente de condições.

base de indução (n=0):

$\underline{D_1(0)}$: Seja $\Delta\subseteq$ Form(L) e suponha-se que $ne^*(\Delta)=0$. Então qualquer $\Phi_{ECNoNoF}$-tableau *T* para Δ verifica $NE(T)=0$ [Resultado 6.7]. Isto significa que estes tableaux contêm apenas fórmulas sem operadores modais e portanto as regras de expansão (RrE), (RC), (RD), (RDC), (RNo), (RNoF), (R$\neg\Box$ cl) e (R$\Box$ cl) não foram aplicadas durante o processo da sua construção (pois as premissas dessas regras referem fórmulas com a modalidade $\Box$). Consequentemente estes tableaux são também Φ_{PL}-tableaux para Δ. Logo as regras de expansão de $\Phi_{ECNoNoF}$-tableaux para Δ preservam a $\varnothing$-satisfação na classe de modelos $\mathcal{C}_{ECNoNoF}$ [Resultado 6.4].

$\underline{D_2(0)}$: Imediato [$D_1(0)$] (demonstração análoga à efectuada no Resultado 6.2).

$\underline{D_3(0)}$: Imediato [$D_1(0)$] (demonstração análoga à efectuada no Resultado 6.3).

passo de indução (n=k): Suponha-se, por hipótese de indução, que $(\forall\ n<k)\ D_1(n)$, $(\forall\ n<k)\ D_2(n)$ e $(\forall\ n<k)\ D_3(n)$.

$\underline{D_1(k)}$: Seja $\Delta\subseteq$ Form(L) e suponha-se que $ne^*(\Delta)=k$. Então qualquer $\Phi_{ECNoNoF}$-tableau *T* para Δ verifica $NE(T)=k$ [Resultado 6.7]. Seja então que *T* um $\Phi_{ECNoNoF}$-tableau para Δ e suponha-se que *T* é $\varnothing$-satisfeito em $\mathcal{C}_{ECNoNoF}$ e que se aplica ao ramo θ do tableau *T* uma das regras de expansão de tableaux em $\Phi_{ECNoNoF}$ obtendo-se assim um tableau T^*. Ora uma vez que *T* é um tableau $\varnothing$-satisfeito em $\mathcal{C}_{ECNoNoF}$

então existe pelo menos um ramo τ de T que é $\varnothing$-satisfeito em C_{ECNoNoF} [Definição 6.5]. Veja-se então que também T^* é $\varnothing$-satisfeito em C_{ECNoNoF}. (no que se segue representar-se-á por Θ, Θ'_1, Θ'_2 os conjuntos das fórmulas que constituem respectivamente os ramos θ, θ'_1 e θ'_2). Caso $\theta \neq \tau$: Trivial, já que τ é ainda um ramo de T^*, e portanto T^* permanece $\varnothing$-satisfeito em C_{ECNoNoF}; Caso $\theta = \tau$: Então θ é $\varnothing$-satisfeito em C_{ECNoNoF}, i.e. [Definição 6.5]:

$$(\exists\, M \in C_{\text{ECNoNoF}})\ (\exists\, \alpha \text{ em } M)\ (\forall\, A \in \Theta)\ M,\alpha \models A \qquad (1)$$

Veja-se então, caso a caso, a aplicação de cada uma das regras de expansão em Φ_{ECNoNoF} ao tableau T, em que T^* difere de T no ramo θ' (resp. nos ramos θ'_1 e θ'_2, no caso da aplicação da regra (Rβ)):

(i) regras (R$\neg\neg$), (R$\neg$True), (R$\neg$False), (Rα) e (Rβ): Prova análoga à do Resultado 6.4.

(ii) regra (RrE): Veja-se, por *redução ao absurdo*, que esta regra nunca é aplicada ao ramo θ, caso que se está aqui a considerar. Suponha-se que (RrE) é aplicada ao ramo $\theta = \tau$ do tableau T. Nesse caso existiriam em θ fórmulas da forma $\Box A$ e $\neg \Box B$ e tal que $\vdash_{\Phi_{\text{ECNoNoF}}} A \leftrightarrow B$. Tem-se [(1)]:

$M,\alpha \models \Box A$ e $M,\alpha \models \neg \Box B$

sse

$\alpha \in f(||A||^M)$ e $\alpha \notin f(||B||^M)$ [93]

$\Rightarrow$

$f(||A||^M) \neq f(||B||^M)$ (2)

Por outro lado, uma vez que NE(T)=k, tem-se [Definição 6.7; Definição 6.8]:

$ne^*(\{\Box A, \neg \Box B\}) \leq k$

sse

$ne^*(\{\neg(A \leftrightarrow B)\}) < k$

$\Rightarrow$ (por hipótese de indução: $(\forall\, n<k)\ D_3(n)$; $\vdash_{\Phi_{\text{ECNoNoF}}} A \leftrightarrow B$)

$\models_{C\text{ECNoNoF}} A \leftrightarrow B$

$\Rightarrow$ [Definição 2.7; Resultado 2.9]

$||A||^M = ||B||^M$

$\Rightarrow$

$f(||A||^M) = f(||B||^M)$

Mas isto contradiz (2).

[93] Uma vez que nesta secção se considera apenas sistemas de lógica modal clássica uni-modal, nos modelos mínimos utilizar-se-á apenas *f* para designar a função associada ao operador modal de necessidade $\Box$.

(iii) regra (RC): Neste caso o ramo θ' foi obtido a partir de fórmulas da forma $\neg\Box(C_1\wedge...\wedge C_n)$ (para $n>1$) e $\Box A$ em θ, por extensão de θ com a fórmula da forma $\neg\Box(Ci_1\wedge...\wedge Ci_k)$ (para $1\leq k<n$), obtendo-se $\Theta'=\Theta\cup\{\neg\Box(Ci_1\wedge...\wedge Ci_k)\}$. Por outro lado a aplicação da regra (RC) impõe que se verifique $\vdash_{\mathbf{\Phi}_{ECNoNoF}} A\leftrightarrow(Ci_{k+1}\wedge...\wedge Ci_n)$. Tem-se portanto [(1)]:

$M,\alpha\models\neg\Box(C_1\wedge...\wedge C_n)$ e $M,\alpha\models\Box A$

sse

$\alpha\notin f(||C_1\wedge...\wedge C_n||^M)$ e $\alpha\in f(||A||^M)$ (3)

Por outro lado, uma vez que $NE(T)=k$, tem-se [Definição 6.7; Definição 6.8]:

$ne^*(\{\neg\Box(C_1\wedge...\wedge C_n),\Box A\})\leq k$

sse

$ne^*(\{\neg(A\leftrightarrow(Ci_{k+1}\wedge...\wedge Ci_n))\})<k$

$\Rightarrow$ (por hipótese de indução; $\vdash_{\mathbf{\Phi}_{ECNoNoF}} A\leftrightarrow(Ci_{k+1}\wedge...\wedge Ci_n)$)

$\models_{\mathcal{C}_{ECNoNoF}} A\leftrightarrow(Ci_{k+1}\wedge...\wedge Ci_n)$

$\Rightarrow$ [Definição 2.7; Resultado 2.9]

$||A||^M=||(Ci_{k+1}\wedge...\wedge Ci_n)||^M$,

$\Rightarrow$

$f(||A||^M)=f(||(Ci_{k+1}\wedge...\wedge Ci_n)||^M)$

$\Rightarrow$ [(3)] ($M\in\mathcal{C}_{ECNoNoF}$ e portanto verifica a condição (c))

$\alpha\notin f(||(Ci_1\wedge...\wedge Ci_k)||^M)$

sse

$M,\alpha\models\neg\Box(Ci_1\wedge...\wedge Ci_k)$

Então Θ' é $\emptyset$-satisfeito em $\mathcal{C}_{ECNoNoF}$ e portanto também T^* é $\emptyset$-satisfeito em $\mathcal{C}_{ECNoNoF}$ [Definição 6.5].

(iv) regra (RD): Veja-se, por *redução ao absurdo*, que esta regra nunca é aplicada ao ramo θ, caso que se está aqui a considerar. Suponha-se que (RD) é aplicada ao ramo $\theta=\tau$ do tableau T. Nesse caso existiriam em θ fórmulas da forma $\Box A$ e $\Box B$ e tal que $\vdash_{\mathbf{\Phi}_{CECNoNoF}} A\leftrightarrow\neg B$. Tem-se [(1)]:

$M,\alpha\models\Box A$ e $M,\alpha\models\Box B$

sse

$\alpha\in f(||A||^M)$ e $\alpha\in f(||B||^M)$

$\Rightarrow$ (modelos da classe $\mathcal{C}_{ECNoNoF}$ satisfazem a condição (d))

$\alpha\in f(||A||^M)$ e $\alpha\notin f(||\neg B||^M)$

$\Rightarrow$

$f(||A||^M)\neq f(||\neg B||^M)$ (4)

Por outro lado, uma vez que $NE(T)=k$, tem-se [Definição 6.7; Definição 6.8]:

$ne^*(\{\Box A, \Box B\}) \leq k$

sse

$ne^*(\{\neg(A \leftrightarrow \neg B)\}) < k$

$\Rightarrow$ (por hipótese de indução: $(\forall\ n<k)\ D_3(n);\ \vdash_{\Phi_{ECNoNoF}} A \leftrightarrow \neg B$)

$\vDash_{CECNoNoF} A \leftrightarrow \neg B$

$\Rightarrow$ [Definição 2.7; Resultado 2.9]

$||A||^M = ||\neg B||^M$

$\Rightarrow$

$f(||A||^M) = f(||\neg B||^M)$

Mas isto contradiz (4).

(v) regra (RDC): Prova análoga a (iii), tomando adicionalmente em consideração a condição (d).

(vi) regra (RNo): Veja-se, por *redução ao absurdo*, que esta regra nunca é aplicada ao ramo θ, caso que se está aqui a considerar. Suponha-se que (RNo) é aplicada ao ramo $\theta = \tau$ do tableau *T*. Nesse caso existiria em θ uma fórmula da forma $\Box A$ e tal que $\vdash_{\Phi_{ECNoNoF}} A$. Tem-se [(1)]:

$M, \alpha \vDash \Box A$

sse

$\alpha \in f(||A||^M)$ (5)

Por outro lado, uma vez que $NE(T)=k$, tem-se [Definição 6.7; Definição 6.8]:

$ne^*(\{\Box A\}) \leq k$

sse

$ne^*(\{\neg A\}) < k$

$\Rightarrow$ (por hipótese de indução: $(\forall\ n<k)\ D_3(n);\ \vdash_{\Phi_{ECNoNoF}} A$)

$\vDash_{CECNoNoF} A$

$\Rightarrow$

$||A||^M = W$

$\Rightarrow$ [(5)]

$f(W) \neq \varnothing$

Mas isto contradiz o facto de o modelo *M* satisfazer a restrição (no).

(vii) regra (RNoF): prova análoga a (vi), tomando neste caso em consideração a condição (nof).

(viii) regra ($R\neg\Box$ cl): Considere-se a regra mais geral ($R\neg\Box$). Neste caso o ramo θ' foi obtido a partir de uma fórmula da forma $\neg\Box A$ em θ, por extensão de θ com a fórmula da forma $\neg\Box B$, obtendo-se $\Theta' = \Theta \cup \{\neg\Box B\}$. Tem-se portanto [(1)]:

$M, \alpha \vDash \neg\Box A$

sse

$\alpha \notin f(||A||^M)$ (6)

Por outro lado a aplicação da regra ($R\neg\Box$) impõe que se verifique $\vdash_{\Phi_{PL}} A\leftrightarrow B$. Tem-se:

$\vdash_{\Phi_{PL}} A\leftrightarrow B$

$\Rightarrow$ [Lema 6.2(3)]

$\vDash_{C_{ECNoNoF}} A\leftrightarrow B$

$\Rightarrow$ [Definição 2.7; Resultado 2.9]

$||A||^M=||B||^M$

$\Rightarrow$

$f(||A||^M)=f(||B||^M)$

$\Rightarrow$ [(6)]

$\alpha \notin f(||B||^M)$

sse

$M,\alpha \vDash \neg\Box B$

Então Θ' é $\varnothing$-satisfeito em $C_{ECNoNoF}$ e portanto também T^* é $\varnothing$-satisfeito em $C_{ECNoNoF}$ [Definição 6.5].

(ix) regra ($R\Box$ cl): prova análoga a (viii).

Fica portanto demonstrado $D_1(k)$. Consequentemente verifica-se (1').

$D_2(k)$: Imediato [$D_1(k)$] (demonstração análoga à efectuada no Resultado 6.2). Consequentemente verifica-se (2').

$D_3(k)$: Imediato [$D_1(k)$] (demonstração análoga à efectuada no Resultado 6.3). Consequentemente verifica-se (3'). ♦

Resultado 6.9: Seja $\Gamma \subseteq Form(L)$ e seja Σ um dos sistemas mencionados na Definição 6.6. Então as regras de Φ_Σ preservam a Γ-satisfação na classe de modelos C_Σ.

Demonstração: Demonstração análoga à efectuada no Resultado 6.4 e no Resultado 6.8(1) (mas sem ser por indução no número de encaixes modais). ♦

Resultado 6.10: Seja $\Gamma \subseteq Form(L)$, $A \in Form(L)$ e seja Σ um dos sistemas mencionados na Definição 6.6. Então:

se $\Gamma \vdash_{\Phi_\Sigma} A$ então $\Gamma \vDash_{C_\Sigma} A$.

Demonstração: Imediato [Resultado 6.3; Resultado 6.9] ♦

6.3 Demonstrador de Teoremas de Lact$_N$

A estratégia adoptada na construção de um sistema de tableaux para a lógica Lact$_N$ é a de combinar os sistemas de tableaux associados aos sistemas lógicos dos vários operadores modais utilizados na lógica Lact$_N$ (quando estes são considerados isoladamente), considerando também regras de expansão de tableaux adicionais para lidar com os relacionamentos inter-modais.

No que se segue utilizar-se-á a notação proposta na secção anterior para denominar regras de expansão e sistemas de tableaux, mas agora indexados pelo operador modal em questão.

Repare-se que alguns dos sistemas de tableaux propostos na secção anterior estão associados aos sistemas lógicos dos vários operadores modais utilizados na lógica Lact$_N$ (quando considerados isoladamente), a saber: um sistema de tableaux Φ_{ECTNo} associado a cada um dos operadores modais E_x e G_x (para x=1,...,N), i.e. os sistemas $\Phi ECTNo_{E_x}$ e $\Phi ECTNo_{G_x}$; e um sistema de tableaux $\Phi ECNoNoF_{{}_xI_y}$ associado a cada um dos operadores modais ${}_xI_y$ (para x,y=1,…,N).

Para lidar com os relacionamentos inter-modais, propõe-se aqui a utilização de subconjuntos das seguintes regras de expansão de $\Phi\cup\{R\Gamma\}$-tableaux da forma seguinte (genericamente, para os operadores modais $\Box_1$, $\Box_2$, ...):

(RQ$_{\Box_1,\Box_2}$) $$\frac{\Box_1\Box_2 A}{\Box_1 A}$$

(RnoQ$_{\Box_1,\Box_2}$) $$\frac{\Box_1\Box_2 A}{\neg\Box_1 A}$$

(R$\Box_1\rightarrow\Box_2$) $$\frac{\Box_1 A}{\Box_2 A}$$

(R$\Box_1\Box_2\rightarrow\Box_3$) $$\frac{\Box_1\Box_2 A}{\Box_3 A}$$

(R$\Box_1\Box_2\rightarrow\Box_1\Box_3$) $$\frac{\Box_1\Box_2 A}{\Box_1\Box_3 A}$$

$$(R\Box_1\&\Box_2\rightarrow\Box_3)\quad \frac{\neg\Box_3 A,\ \Box_1 B \quad [\vdash_{\Phi} A\leftrightarrow B]}{\neg\Box_2 A}$$

A formulação das regras anteriores é motivada pelos axiomas-esquema multi-modais utilizados na lógica Lact_N. A Tabela 6-3 apresenta a forma genérica desses esquemas multi-modais e as respectivas regras de expansão de tableaux associadas (onde $\Box_1$, $\Box_2$ e $\Box_3$ poderão designar, ou não, diferentes operadores modais).[94]

Esquema multi-modal	Regra de expansão de tableaux
$\Box_1\Box_2 A\rightarrow\Box_1 A$	$RQ_{\Box 1,\Box 2}$
$\Box_1\Box_2 A\rightarrow\neg\Box_1 A$	$RnoQ_{\Box 1,\Box 2}$
$\Box_1 A\rightarrow\Box_2 A$	$R\Box_1\rightarrow\Box_2$
$\Box_1\Box_2 A\rightarrow\Box_3 A$	$R\Box_1\Box_2\rightarrow\Box_3$
$(\Box_1 A\wedge\Box_2 A)\rightarrow\Box_3 A$	$R\Box_1\&\Box_2\rightarrow\Box_3$
$\Box_1\Box_2 A\rightarrow\Box_1\Box_3 A$	$R\Box_1\Box_2\rightarrow\Box_1\Box_3$

Tabela 6-3: Esquemas multi-modais e respectivas regras de expansão de tableaux

A motivação das regras de expansão de tableaux ($RQ_{\Box 1,\Box 2}$), ($RnoQ_{\Box 1,\Box 2}$), ($R\Box_1\rightarrow\Box_2$), ($R\Box_1\Box_2\rightarrow\Box_3$) e ($R\Box_1\Box_2\rightarrow\Box_1\Box_3$) é óbvia. A regra ($R\Box_1\&\Box_2\rightarrow\Box_3$) é motivada pelo facto de se poder inferir do esquema $\Box_1\&\Box_2\rightarrow\Box_3$ e de ($\Box_1$-rE) a regra de prova "de $\vdash A\leftrightarrow B$ infere-se $(\neg\Box_3 A\wedge\neg\Box_1 B)\rightarrow\neg\Box_2 B$".

A formulação da regra ($R\Box_1\&\Box_2\rightarrow\Box_3$) reflecte também preocupações relacionadas com a implementação do método proposto (análogas às consideradas na secção anterior a propósito da regra (RC)). Por exemplo, em vez da regra ($R\Box_1\&\Box_2\rightarrow\Box_3$), poder-se-ia propor a regra

$$(R'\Box_1\&\Box_2\rightarrow\Box_3)\quad \frac{\neg\Box_3 A}{\neg\Box_1 A \mid \neg\Box_2 A}$$

No entanto, como já se referiu, tentou-se evitar, sempre que possível, a definição de regras cuja aplicação provocasse a ramificação de um tableau, uma vez que na implementação adoptada, cada

[94] Os esquemas $\Box_1\Box_2 A\rightarrow\Box_1 A$, $\Box_1\Box_2 A\rightarrow\neg\Box_1 A$, $\Box_1 A\rightarrow\Box_2 A$, $\Box_1\Box_2 A\rightarrow\Box_3 A$ e $(\Box_1 A\wedge\Box_2 A)\rightarrow\Box_3 A$ representam respectivamente os axiomas-esquema (G_x-Q), (EEnoE), (EG), (Dir.control) e (Trans.inf) considerados na lógica Lact_N. E o esquema $\Box_1\Box_2 A\rightarrow\Box_1\Box_3 A$ representa qualquer um dos axiomas-esquema (EEEG) e (GEGG).

ramificação provoca a duplicação do espaço em memória necessário à representação de um tableau.

Mas nem sempre se pode respeitar este critério de eficiência. Relativamente à regra ($R\Box_1\&\Box_2\rightarrow\Box_3$), são também necessários alguns cuidados adicionais dependendo dos tipos de sistemas de lógica modal considerados para cada um dos operadores envolvidos, quer naquela regra, quer noutras regras relacionadas com o operador $\Box_3$. Por exemplo, em casos em que se considere a regra ($\Box_3$-RC) e conjuntamente as regras ($R\Box_1\&\Box_2\rightarrow\Box_3$) e ($R\Box_4\&\Box_5\rightarrow\Box_3$), é necessário considerar adicionalmente a regra

$$(\Box_3\text{-RC''})\quad \frac{\neg\Box_3(C_1\wedge\ldots\wedge C_n)}{\neg\Box_3(Ci_1\wedge\ldots\wedge Ci_k)\mid\neg\Box_3(Ci_{k+1}\wedge\ldots\wedge Ci_n)}\ ,\ (\text{para } 1\leq k<n)$$

a fim de assegurar que as regras ($R\Box_1\&\Box_2\rightarrow\Box_3$) e ($R\Box_4\&\Box_5\rightarrow\Box_3$) são aplicadas em casos em que na construção de um tableau aparecem - num mesmo ramo - fórmulas da forma $\neg\Box_3(A\wedge B)$, $\Box_1A$, $\Box_2A$, $\Box_4B$ e $\Box_5B$.

Para ilustrar esta última observação, considere-se o tableau ilustrado na Figura 6-5 e que suporta a dedução $\{\Box_1p1,\Box_2p1,\Box_4p2,\Box_5p2\}\vdash_\Phi\Box_3(p1\wedge p2)$, para $\Phi=\Phi EC_{\Box_1}\cup\Phi E_{\Box_2}\cup\Phi E_{\Box_3}\cup\Phi E_{\Box_4}\cup\Phi E_{\Box_5}\cup\{\Box_3\text{-RC''},R\Box_1\&\Box_2\rightarrow\Box_3,R\Box_4\&\Box_5\rightarrow\Box_3\}$.

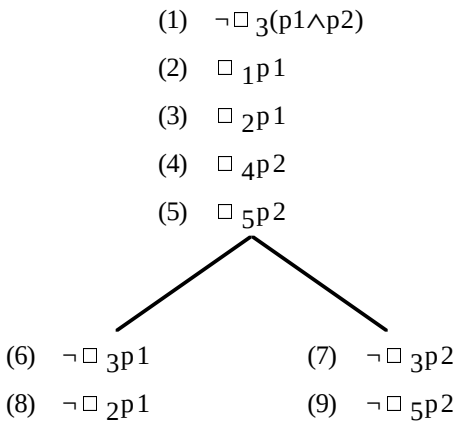


Figura 6-5: Um $\Phi EC_{\Box_1}\cup\Phi E_{\Box_2}\cup\Phi E_{\Box_3}\cup\Phi E_{\Box_4}\cup\Phi E_{\Box_5}\cup\{\Box_3\text{-RC''},R\Box_1\&\Box_2\rightarrow\Box_3,$ $R\Box_4\&\Box_5\rightarrow\Box_3, R\{\Box_1p1,\Box_2p1,\Box_4p2,\Box_5p2\}\}$-tableau para $\{\neg\Box_3(p1\wedge p2)\}$

Na figura apresenta-se um Φ-tableau para $\{\neg\Box_3(p1\wedge p2)\}$: O nó (1) é introduzido por aplicação da Definição 6.1(i); os nós (2) a (5) por aplicação da regra ($R\{\Box_1p1,\Box_2p1,\Box_4p2,\Box_5p2\}$); os nós (6) e (7) por aplicação da regra ($\Box_3$-RC'') ao nó (1); o nó (8) por aplicação da regra ($R\Box_1\&\Box_2\rightarrow\Box_3$) aos nós (2) e (6); e o nó (9) por aplicação da regra ($R\Box_4\&\Box_5\rightarrow\Box_3$) aos nós (4)

e (7). Este tableau é fechado devido a (3) e (8) - no ramo (1,2,3,4,5,6,8) - e devido a (5) e (9) - no ramo (1,2,3,4,5,7,9). Consequentemente obtém-se a dedução pretendida [Definição 6.4]. Repare-se que as aplicações das regras ($R\Box_1\&\Box_2\rightarrow\Box_3$) e ($R\Box_4\&\Box_5\rightarrow\Box_3$) não seriam possíveis caso não se aplicasse primeiro a regra ($\Box_3$-RC"). Repare-se ainda que a regra ($\Box_3$-RC), tal como foi formulada (veja-se a página 135), também não seria aplicável neste caso.

No sistema de tableaux para a lógica $Lact_N$ utilizar-se-á a seguinte concretização da regra ($R\Box_1\&\Box_2\rightarrow\Box_3$) associada ao esquema (Trans.inf) - i.e. $({}_xI_yA\wedge G_yA)\rightarrow G_xA$:

$$(R_xI_y\&G_y\rightarrow G_x)\quad \frac{\neg G_xA,\ {}_xI_yB}{\neg G_yA}\quad [\vdash_{\Phi Lact_N} A\leftrightarrow B]$$

E uma vez que se considera a regra (G_x-RC) no sistema de tableaux associado ao operador G_x, será necessário considerar a regra adicional (G_x-RC") pelas razões que acabaram de ser mencionadas.

Por outro lado, cada uma das regras ($RQ_{\Box_1,\Box_2}$), ($RnoQ_{\Box_1,\Box_2}$), ($R\Box_1\Box_2\rightarrow\Box_3$) e ($R\Box_1\Box_2\rightarrow\Box_1\Box_3$) necessita ser combinada com a regra de expansão de tableaux adicional

$$(\Box_1\text{-RrEred})\quad \frac{\Box_1(C_1\wedge\ldots\wedge C_n)\quad [\vdash_{\Phi}(Ci_{k+1}\wedge\ldots\wedge Ci_n)]}{\Box_1(Ci_1\wedge\ldots\wedge Ci_k)}\quad ,\ (\text{para } 1\leq k<n)$$

a fim de assegurar que aquelas regras são aplicadas em casos em que na construção de um tableau aparecem fórmulas da forma $\Box_1(\Box_2A\wedge B)$ e em que $\vdash_{\Phi} B$. Neste caso considere-se ainda a regra

$$(R\Box_1)\qquad \frac{\Box_1A\quad [\vdash_{\Phi_{PL}} A\leftrightarrow B]}{\Box_1B}$$

ou, como caso particular, a seguinte regra

$$(R\Box_1\text{cl"})\qquad \frac{\Box_1A}{\Box_1\text{clausal}(A)}$$

a fim de permitir o tratamento de qualquer fórmula “proposicional” no âmbito de um $\Box_1$ e portanto possibilitar a utilização da regra ($\Box_1$-RrEred).

Para ilustrar a aplicação da regra ($\Box_1$-RrEred), considere-se o tableau ilustrado na Figura 6-6 e que suporta a dedução $\{\Box_1(\Box_2p1\wedge(p2\rightarrow p2))\}\vdash_{\Phi}\Box_3p1$, para $\Phi=\Phi E_{\Box_1}\cup\Phi E_{\Box_2}\cup\Phi E_{\Box_3}\cup\{R\Box_1\Box_2\rightarrow\Box_3,\Box_1\text{-RrEred}\}$. Na figura apresenta-se um $\Phi E_{\Box_1}\cup\Phi E_{\Box_2}\cup\Phi E_{\Box_3}\cup\{\Box_1$-RrEred,

$R\Box_1\Box_2\rightarrow\Box_3$, $R\{\Box_1(\Box_2 p1\wedge(p2\rightarrow p2))\}$-tableau fechado para $\{\neg\Box_3 p1\}$: O nó (1) é introduzido por aplicação da Definição 6.1(i); o nó (2) por aplicação da regra ($R\{\Box_1(\Box_2 p1\wedge(p2\rightarrow p2))\}$); o nó (3) por aplicação da regra ($\Box_1$-RrEred) ao nó (2); e o nó (4) por aplicação da regra ($R\Box_1\Box_2\rightarrow\Box_3$) ao nó (3). Este tableau é fechado devido a (1) e (4). Consequentemente obtém-se a dedução pretendida [Definição 6.4]. A condição de aplicabilidade da regra ($\Box_1$-RrEred) é avaliada pelo $\Phi E_{\Box 1}\cup\Phi E_{\Box 2}\cup\Phi E_{\Box 3}\cup\{\Box_1$-RrEred, $R\Box_1\Box_2\rightarrow\Box_3\}$-tableau fechado para $\{\neg(p2\rightarrow p2)\}$ da direita. Repare-se que, no tableau da esquerda, a aplicação da regra ($R\Box_1\Box_2\rightarrow\Box_3$) não seria possível caso não se aplicasse primeiro a regra ($\Box_1$-RrEred).

(1)	$\neg\Box_3 p1$	(3.1)	$\neg(p2\rightarrow p2)$
(2)	$\Box_1(\Box_2 p1\wedge(p2\rightarrow p2))$	(3.2)	$p2$
(3)	$\Box_1\Box_2 p1$	(3.3)	$\neg p2$
(4)	$\Box_3 p1$		

Figura 6-6: Um $\Phi E_{\Box 1}\cup\Phi E_{\Box 2}\cup\Phi E_{\Box 3}\cup\{\Box_1$-RrEred, $R\Box_1\Box_2\rightarrow\Box_3$, $R\{\Box_1(\Box_2 p1\wedge(p2\rightarrow p2))\}$-tableau para $\{\neg\Box_3 p1\}$

No sistema de tableaux para a lógica $Lact_N$ utilizar-se-á concretizações das regras ($RQ_{\Box 1,\Box 2}$), ($RnoQ_{\Box 1,\Box 2}$) e ($R\Box_1\Box_2\rightarrow\Box_3$) associadas respectivamente aos esquemas (G_X-Q), (EEnoE) e (Dir.control) e regra ($R\Box_1\Box_2\rightarrow\Box_1\Box_3$) associada aos esquemas (EEEG) e (GEGG). É portanto necessário considerar como regras adicionais: (E_X-RrEred) e (G_X-RrEred), para x=1,..., N.

Veja-se, para terminar, um sistema de tableaux para a lógica $Lact_N$, que se passará a denominar de $TLact_N$.

Definição 6.9: O sistema de tableaux $TLact_N$

No sistema de tableaux $TLact_N$ utiliza-se o seguinte conjunto de regras de expansão de tableaux:

$$\Phi Lact_N = \cup_{\{x,y=1,...,N\}} (\Phi ECTNo_{Ex}\cup\Phi ECTNo_{Gx}\cup\{G_X\text{-}RC''_{Gx},E_X\text{-}RrEred,RE_X cl''\} \cup\{G_X\text{-}RrEred_{Gx},RG_X cl'',RE_X\rightarrow G_X\})\cup$$
$$\cup_{\{x,y=1,...,N\}} (\Phi ECNoNoF_{xIy}\cup\{RQ_{Gx,Gy},RE_XE_y\rightarrow E_XG_y,RG_XE_y\rightarrow G_XG_y\}\cup \{RE_XG_y\rightarrow_X I_y,\ R_XI_y\&G_y\rightarrow G_X\})\ \cup$$
$$\cup_{\{x,y=1,...,N\ e\ x\neq y\}} \{RnoQ_{Ex,Ey}\}$$ ♦

Veja-se então que o sistema dedutivo $Tlact_N$ é fortemente correcto (i.e. se $\Gamma\vdash_{\Phi Lact_N} A$, então

$\Gamma \models_{\mathcal{C}\mathrm{Lact_N}} A$).

Resultado 6.11: Sejam Γ, Δ (finito) $\subseteq$ Form(L) e $A \in$ Form(L). Então:

(1) as regras de $\Phi_{\mathrm{Lact_N}} \cup \{R\Gamma\}$ preservam a Γ-satisfação na classe de modelos $\mathcal{C}_{\mathrm{Lact_N}}$.

(2) se existe um $\Phi_{\mathrm{Lact_N}} \cup \{R\Gamma\}$-tableau fechado para Δ, então Δ não é Γ-satisfeito em $\mathcal{C}_{\mathrm{Lact_N}}$.

(3) se $\Gamma \vdash_{\Phi_{\mathrm{Lact_N}}} A$, então $\Gamma \models_{\mathcal{C}_{\mathrm{Lact_N}}} A$.

Demonstração: Demonstração análoga às efectuadas nos Resultado 6.8, Resultado 6.9 e Resultado 6.10. ♦

Consequentemente o sistema $\mathrm{TLact_N}$ suporta correctamente as deduções na lógica $\mathrm{Lact_N}$.

Na implementação do sistema $\mathrm{TLact_N}$ foi necessário adoptar restrições à aplicação das regras de expansão de tableaux em $\Phi\mathrm{Lact_N}$ de modo a assegurar a terminação na procura de refutações. Repare-se que os sistemas de tableaux descritos não impõem restrições à aplicação das regras de expansão de tableaux. À partida, estas podem ser aplicadas de modo não determinista. Inclusivamente as mesmas regras podem ser aplicadas consecutivamente à(s) mesma(s) fórmula(s) durante o processo de construção de um tableau.

A estratégia adoptada na implementação do sistema $\mathrm{TLact_N}$ foi precisamente evitar a reutilização num mesmo ramo de cada regra ao mesmo conjunto de fórmulas (i.e. premissas). Note-se que esta estratégia é similar à utilizada nos "tableaux estritos" (veja-se [Fitting 90], página 39), onde se consideram apenas tableaux em que nenhuma fórmula pode ser utilizada duas vezes pelas várias regras de expansão de tableaux. No entanto, aqui permite-se que isso aconteça, desde que as regras aplicadas sejam diferentes (à excepção da regra (RC")).

Na implementação do sistema $\mathrm{TLact_N}$ (veja-se o Anexo 1) utiliza-se a implementação apresentada em [Fitting 90] para tableaux proposicionais, estendida com as regras de expansão de tableaux consideradas em $\mathrm{TLact_N}$ e de acordo com a estratégia anterior.

6.4 Automação da Especificação e Análise de Organizações

A especificação e análise de organizações apresentada no capítulo 5 é actualmente suportada por uma "bancada" implementada em Prolog (veja-se o Anexo 2). A "bancada" suporta a descrição de uma organização *Org*, gera o conjunto de comportamentos possíveis B_{Org} e calcula respostas dos tipos R1, R2, R3' e R3". A "bancada" utiliza no cálculo dessas respostas o método de prova

automática proposto na secção anterior. No entanto, uma vez que não foi determinada a completude do sistema $TLact_N$, apenas por utilização deste sistema dedutivo não se pode garantir que se exclui a possibilidade da inclusão de conjuntos de acções incoerentes durante a geração do modelo B_{Org} (de uma organização *Org*). Na subsecção 6.4.1 apresenta-se um resultado adicional que permite resolver este problema. Finalmente, na subsecção 6.4.2 apresenta-se a estrutura funcional da "bancada" e ilustra-se a sua utilização.

6.4.1 Geração do Modelo

A completude do sistema $Tlact_N$ não foi determinada. Consequentemente, não poderia ser ignorada a possibilidade do procedimento computacional implementando o sistema de tableaux $Tlact_N$ incluir comportamentos incoerentes no conjunto B_{Org} de uma organização *Org*[95]. No entanto, o próximo teorema apresenta uma solução de compromisso para resolver este problema.

Resultado 6.12: Seja *Org* a estrutura de uma organização, $\Gamma \subseteq \Delta_{Org}$ (veja-se a Definição 5.2), $E^-(\Gamma)=\{A: E_xA \in \Gamma \text{ e } x \in Ag\}$ e ${}_xI_y^-(\Gamma)=\{A: {}_xI_yA \in \Gamma\}$ $(x,y \in Ag)$.[96] Então:

$Con_{Lact_N}\ \Gamma$ sse se verificam simultaneamente as seguintes três condições:

(1) $Con_{PL}\ E^-(\Gamma)$;

(2) $Con_{PL}\ {}_xI_y^-(\Gamma)$, para todo $x,y \in Ag$;

(3) $\nvdash_{PL} B$, para todo o $B \in (E^-(\Gamma) \cup \bigcup_{\{x,y \in Ag\}} {}_xI_y^-(\Gamma))$.

Demonstração: Seja *Org* a estrutura de uma organização, $\Gamma \subseteq \Delta_{Org}$, $E^-(\Gamma)=\{A: E_xA \in \Gamma \text{ e } x \in Ag\}$ e ${}_xI_y^-(\Gamma)=\{A: {}_xI_yA \in \Gamma\}$ $(x,y \in Ag)$.

(I) Se $Con_{Lact_N}\ \Gamma$ então verificam-se simultaneamente (1), (2) e (3). Demonstração por *contra-recíproco*: Suponha-se que uma das condições (1), (2) ou (3) não se verifica:

(I-a) A condição (1) não se verifica, i.e. $C\varnothing n_{PL}\ E^-(\Gamma)$. Tem-se:

$C\varnothing n_{PL}\ E^-(\Gamma)$
sse [Definição 2.2; Resultado 2.2(iii)]
$E^-(\Gamma) \vdash_{Lact_N} False$
sse [Definição 2.1]

---

[95] Repare-se que uma vez que a completude do sistema $Tlact_N$ não foi determinada não se pode garantir que se verifica: "se $\Gamma \nvdash_{Lact_N} False$ então $Con_{Lact_N}\Gamma$.

[96] Note-se que como $\Gamma \subseteq \Delta_{Org}$, tem-se: $E^-(\Gamma)$, ${}_xI_y^-(\Gamma) \subseteq Form(L_{PL})$. Tal será relevante em partes desta demonstração, como se assinalará oportunamente.

$(\exists\ n\geq 0)\ (\{A_1,...,A_n\}\subseteq E^-(\Gamma)$ e $\vdash_{Lact_N} (A_1\wedge...\wedge A_n)\rightarrow False)$ (4)

$\Rightarrow$ [Resultado 2.2(vi)]

$\Gamma\vdash_{Lact_N} (A_1\wedge...\wedge A_n)\rightarrow False$ (5)

Por outro lado, uma vez que $\{A_1,...,A_n\}\subseteq E^-(\Gamma)$ [(4)], tem-se:

$\{A_1,...,A_n\}\subseteq E^-(\Gamma)$

sse

$(\exists\ \{x_1,...,x_n\}\subseteq Ag)\ (\{E_{x1}A_1,...,E_{xn}A_n\}\subseteq\Gamma)$

$\Rightarrow$ [Resultado 2.2(iv); Resultado 2.1; $(E_{xi}$-T)]

$(\forall\ i=1,...,n)\ \Gamma\vdash_{Lact_N} (A_1\wedge...\wedge A_n)$

$\Rightarrow$ [(5); Resultado 2.1]

$\Gamma\vdash_{Lact_N} False$

sse

$C\varnothing n_{Lact_N}\Gamma$

(I-b) A condição (2) não se verifica, i.e. $(\exists\ x,y\in Ag)\ C\varnothing n_{PL}\ {}_xI_y^-(\Gamma)$. Tem-se:

$C\varnothing n_{PL}\ {}_xI_y^-(\Gamma)$

sse [Definição 2.2; Resultado 2.2(iii)]

${}_xI_y^-(\Gamma)\vdash_{Lact_N} False$

sse [Definição 2.1]

$(\exists\ n\geq 0)\ (\{A_1,...,A_n\}\subseteq\ {}_xI_y^-(\Gamma)$ e $\vdash_{Lact_N} (A_1\wedge...\wedge A_n)\rightarrow False)$ (6)

$\Rightarrow$ [$\vdash_{Lact_N} False\rightarrow(A_1\wedge...\wedge A_n)$; $({}_xI_y$-rE)]

$\vdash_{Lact_N} {}_xI_y(A_1\wedge...\wedge A_n)\leftrightarrow{}_xI_yFalse$

$\Rightarrow$ [Resultado 2.2(vi)]

$\Gamma\vdash_{Lact_N} {}_xI_y(A_1\wedge...\wedge A_n)\leftrightarrow{}_xI_yFalse$ (7)

Por outro lado, uma vez que $\{A_1,...,A_n\}\subseteq\ {}_xI_y^-(\Gamma)$ [(6)], tem-se:

$\{A_1,...,A_n\}\subseteq\ {}_xI_y^-(\Gamma)$

sse

$\{{}_xI_yA_1,...,\ {}_xI_yA_n\}\subseteq\Gamma$

$\Rightarrow$ [Resultado 2.2(iv); Resultado 2.1; $({}_xI_y$-C)]

$\Gamma\vdash_{Lact_N} {}_xI_y(A_1\wedge...\wedge A_n)$

$\Rightarrow$ [(7); Resultado 2.1]

$\Gamma\vdash_{Lact_N} {}_xI_yFalse$

$\Rightarrow$ [$({}_xI_y$-NoF)]

$\Gamma\vdash_{Lact_N} False$

sse

$C\varnothing n_{Lact_N}\Gamma$

(I-c) A condição (3) não se verifica, i.e. $(\exists\ B\in(E^{-}(\Gamma)\cup\bigcup_{\{x,y\in Ag\}}\ {}_{x}I_{y}^{-}(\Gamma)))\vdash_{PL}B$.

Seja $B\in(E^{-}(\Gamma)\cup\bigcup_{\{x,y\in Ag\}}\ {}_{x}I_{y}^{-}(\Gamma))$.

Caso $B\in E^{-}(\Gamma)$: Tem-se:

$B\in E^{-}(\Gamma)$

sse

$(\exists\ x\in Ag)\ E_{x}B\in\Gamma$

$\Rightarrow$ [Resultado 2.2(iv)]

$\Gamma\vdash_{Lact_{No}}E_{x}B$

$\Rightarrow$ [$\vdash_{PL}B$; (E_{x}-rE); Resultado 2.1; Resultado 2.2(vi)]

$\Gamma\vdash_{Lact_{No}}E_{x}True$

$\Rightarrow$ [(E_{x}-No)]

$\Gamma\vdash_{Lact_{N}}False$

sse

$C\varnothing n_{Lact_{N}}\Gamma$

Caso $B\in\bigcup_{\{x,y\in Ag\}}\ {}_{x}I_{y}^{-}(\Gamma)$: Prova análoga ao caso anterior.

(II) Se se verificam simultaneamente (1), (2) e (3), então $Con_{Lact_{N}}\Gamma$: Suponha-se, *per absurdum*, que se verificam (1), (2), (3) e $C\varnothing n_{Lact_{N}}\Gamma$. Tem-se:

$C\varnothing n_{Lact_{N}}\Gamma$

$\Rightarrow$ [Definição 2.2; Definição 2.1; $\Gamma\subseteq\Delta_{Org}$]

$(\exists\ E_{x_1}A_1,...,E_{x_k}A_k,x_{k+1}I_{y_{k+1}}A_{k+1},...,x_nI_{y_n}A_n\in\Gamma)$ $\quad(0\leq k\leq n$ e $n>0$[97]$)$

$\vdash_{Lact_{N}}(E_{x_1}A_1\wedge...\wedge E_{x_k}A_k\wedge x_{k+1}I_{y_{k+1}}A_{k+1}\wedge...\wedge x_nI_{y_n}A_n)\rightarrow False$

$\Rightarrow$ [Definição 2.1; Definição 2.2]

$C\varnothing n_{Lact_{N}}\{E_{x_1}A_1,...,E_{x_k}A_k,x_{k+1}I_{y_{k+1}}A_{k+1},...,\ x_nI_{y_n}A_n\}$

sse [Resultado 4.7]

$\{E_{x_1}A_1,...,E_{x_k}A_k,x_{k+1}I_{y_{k+1}}A_{k+1},...,x_nI_{y_n}A_n\}$ não é $\varnothing$-satisfeito em $\mathcal{C}Lact_{N}$

sse [Definição 2.9]

$(\forall\ M\in\mathcal{C}Lact_{N})\ (\forall\ \alpha$ em $M)\ (\forall\ B\in\{E_{x_1}A_1,...,E_{x_k}A_k,x_{k+1}I_{y_{k+1}}A_{k+1},...,x_nI_{y_n}A_n\})$

$M,\alpha\not\models B$ (8)

A restante parte desta demonstração é dedicada à obtenção de uma contradição com esta última conclusão. Isto é, demonstrar-se-á que se verifica $(\exists\ M\in\mathcal{C}Lact_{N})\ (\exists\ \alpha$ em $M)\ (\exists\ B\in\{E_{x_1}A_1,...,E_{x_k}A_k,x_{k+1}I_{y_{k+1}}A_{k+1},...,x_nI_{y_n}A_n\})\ M,\alpha\models B$. No que segue analisa-se o caso em que $0<k<n$ e $n>1$. Os casos em que $k=0$ ou $k=n$ exigem apenas ligeiras adaptações na demonstração que se segue.

[97] Repare-se que $n\neq0$. Caso $n=0$ ter-se-ia $\vdash_{Lact_{N}}False$, o que contradiz o facto de a lógica $Lact_{N}$ ser coerente [Resultado 4.3].

Uma vez que $\{A_1,...,A_k\} \subseteq E^-(\Gamma)$ e $(\forall\ j=k+1,...,n)\ \ A_j \in {}_{x_j}I_{y_j}{}^-(\Gamma)$, tem-se [(1); (2); (3); Resultado 2.2(xii)]: $Con_{PL}\{A_1,...,A_k\}$; $(\forall\ j=k+1,...,n)\ \ Con_{PL}\{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}{}^-(\Gamma)$; e $(\forall\ j= 1,...,n) \nvdash_{PL} A_j$. (Repare-se que $\{A_1,...,A_k\} \subseteq Form(L_{PL})$) Daqui sai, por resultados da lógica proposicional (veja-se, e.g. [Hamilton 78]):

(P1) $(\exists$ valoração v$)\ (\forall\ j=1,...,k)\ v(A_j)=1$;

(P2) $(\forall\ j=k+1,...,n)\ (\exists$ valoração $u_j)\ (\forall\ B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}{}^-(\Gamma))\ u_j(B)=1$;

(P3) $(\forall\ j= 1,...,n)\ (\exists$ valoração $z_j)\ z_j(A_j)=0$;

Veja-se então como construir o modelo *M* que contradiz (8). Seja $M=\langle W, f_1,...,f_N, g_1,...,g_N, {}_1h_1,...,{}_1h_N,..., {}_Nh_1,...,{}_Nh_N, V\rangle$ onde:

(i) $W=\{\alpha,\beta,\delta_{k+1},...,\delta_n,\eta_1,...,\eta_n\}$;

(ii) $f_x: 2^W \rightarrow 2^W$ $\quad (x = 1, ..., N)$;

(iii) $g_x: 2^W \rightarrow 2^W$ $\quad (x = 1, ..., N)$;

(iv) ${}_xh_y: 2^W \rightarrow 2^W$ $\quad (x,y = 1, ..., N)$;

(v) $V: \{p_1, p_2, ...\} \rightarrow 2^W$

e verificando ainda as seguintes condições:

(vi) para i=1, 2, ...

(vi.1) $\alpha \in V(p_i)$ sse $v(p_i)=1$

(vi.2) $\beta \in V(p_i)$ sse $v(p_i)=1$

(vi.3) $(\forall\ j=k+1,...,n)\ (\delta_j \in V(p_i)$ sse $u_j(p_i)=1)$

(vi.4) $(\forall\ j=1,...,n)\ (\eta_j \in V(p_i)$ sse $z_j(p_i)=1)$

(vii) para todo X subconjunto de W e x=1,...,N:

$f_x(X) = \{\alpha\}$, se $X \in E^*$; e

$= \varnothing$, caso contrário

(viii) para todo X subconjunto de W e x=1,...,N:

$g_x(X) = f_x(X)$

(ix) para todo X subconjunto de W e x,y=1,...,N:

${}_xh_y(X) = \{\alpha\}$, se $X \in {}_xI_y{}^*$; e

$= \varnothing$, caso contrário

em que os conjuntos E^* e ${}_xI_y{}^*$ são construídos indutivamente como se segue:

conjunto E^*: (e1) $||A_j||^M \in E^*$ (para j=1,...,n); [98]

(e2) se $X,Y \in E^*$ então $X \cap Y \in E^*$;

(e3) nada mais pertence a E^*.

conjunto ${}_xI_y{}^*$ (para x,y=1,...,N):

[98] Note-se que esta definição não é problemática. Uma vez que $A_j \in Form(L_{PL})$, para determinar $||A_j||^M$ basta conhecer a função V que foi definida de modo independente.

(i1) se $B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}^{-}(\Gamma)$ então $\|B\|^M \in {}_{x_j}I_{y_j}^{*}$ (para j=k+1,...,n);
(i2) se $X,Y \in {}_{x}I_{y}^{*}$ então $X \cap Y \in {}_{x}I_{y}^{*}$ (para x,y=1,...,N);
(i3) nada mais pertence a ${}_{x}I_{y}^{*}$.

A definição anterior é concebida com o objectivo de obter a veracidade no mundo α de cada uma das fórmulas em $\{E_{x_1}A_1,...,E_{x_k}A_k, x_{k+1}I_{y_{k+1}}A_{k+1},...,x_nI_{y_n}A_n\}$ (imediato [(vii); (ix); Definição 2.12]) e simultaneamente satisfazer as restrições às funções f_x, g_x e ${}_{x}h_{y}$ (x,y = 1, ..., N) de modo a garantir que $M \in C\text{Lact}_N$. Repare-se as definições do conjunto W da função V em M permitem obter a veracidade de cada uma das fórmulas A_j (j=1,...,n) nos mundos convenientes de modo satisfazer as restrições apresentadas na Definição 4.2. A Figura 6-7 apresenta uma representação do conjunto W com indicação da veracidade das fórmulas A_j (j=1,...,n) nos mundos de W (veja-se o Lema 1 adiante).

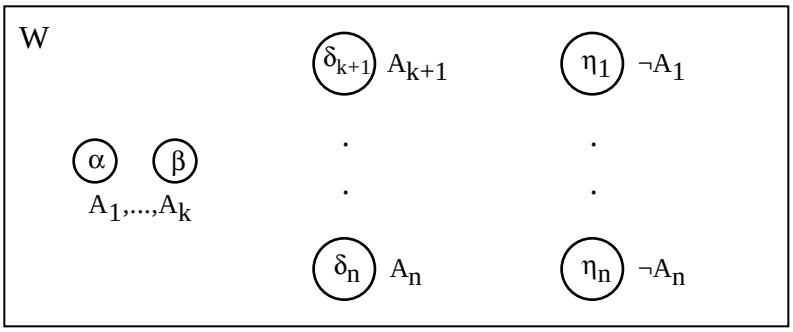


Figura 6-7: Veracidade das fórmulas A_j (j=1,...,n) nos mundos de W.

Os mundos $\eta_1,...,\eta_n$ permitem obter $\|A_j\|^M \neq W$ (para j=1,...,n), condição necessária à satisfação das restrições (noe) e (noi), uma vez que se pretende que se verifique: ($\forall$ j=1,...,k) $f_{x_j}(\|A_j\|^M) \neq \varnothing$ e ($\forall$ j=k+1,...,n) ${}_{x_j}h_{y_j}(\|A_j\|^M) \neq \varnothing$. Os mundos $\delta_{k+1},...,\delta_n$ permitem garantir que $\|A_j\|^M \neq \varnothing$ (para j=k+1,...,n), condição necessária á satisfação da restrição (nofi), uma vez que se pretende que se verifique: ($\forall$ j=k+1,...,n) ${}_{x_j}h_{y_j}(\|A_j\|^M) \neq \varnothing$.[99] Finalmente, o mundo adicional β permite garantir que se verifica a condição (eenoe). Repare-se que se não se considerasse o mundo β, poder-se-ia verificar e.g. $\|A_1\|^M = \|A_2\|^M = \{\alpha\}$ e $x_1 \neq x_2$, e nesse caso a alínea (vii) da definição acima permitiria concluir $f_{x_1}(f_{x_2}(\|A_1\|)) \subseteq f_{x_1}(\|A_1\|)$, não se verificando portanto a restrição (eenoe). Finalmente, note-se que as construções apresentadas para os conjuntos E^* e ${}_{x}I_{y}^{*}$ (para x,y=1,...,N) permitem garantir a satisfação das restrições (ce), (cg) e (ci).

[99] Note-se que se A_j, A_r (j≠r, j,r=k+1,...,n) pertencem ao mesmo ${}_{x}I_{y}^{-}(\Gamma)$ então quer A_j quer A_r são verdadeiras quer em δ_j quer em δ_r. Logo $\varnothing \notin {}_{x}I_{y}^{*}$ (veja-se o Lema 6 adiante nesta demonstração).

Passa-se a enunciar lemas que serão utilizados para demonstrar que $M \in C\text{Lact}_N$:

Lema 1: Seja $B \in \text{Form}(L_{PL})$. Então:

(i) $\alpha \in ||B||^M$ sse $v(B)=1$

(ii) $\beta \in ||B||^M$ sse $v(B)=1$

(iii) $(\forall\ j=k+1,...,n)\ (\delta_j \in ||B||^M$ sse $u_j(B)=1)$

(iv) $(\forall\ j=1,...,n)\ (\eta_j \in ||B||^M$ sse $z_j(B)=1)$

Demonstração: Demonstração por indução na complexidade da fórmula $B \in \text{Form}(L_{PL})$: Imediato [(vi.1); (vi.2); (vi.3); (vi.4); Resultado 2.8].

Lema 2: $(\forall\ X \in E^*)\{\alpha,\beta\} \subseteq X$

Demonstração: Demonstração por indução na construção de E^*.

base de indução: X é $||A_j||^M$ (para j=1,...,k). Tem-se:

$X=||A_j||^M$

$\Rightarrow$ [(P1)]

$v(A_j)=1$

$\Rightarrow$ [Lema 1(i e ii)]

$\{\alpha,\beta\} \subseteq ||A_j||^M$ (i.e. $\{\alpha,\beta\} \subseteq X$)

passo de indução: X é $Y \cap Z$ e $Y,Z \in E^*$. Suponha-se, por hipótese de indução, que $\{\alpha,\beta\} \subseteq Y$ e $\{\alpha,\beta\} \subseteq Z$. Então $\{\alpha,\beta\} \subseteq Y \cap Z$, i.e. $\{\alpha,\beta\} \subseteq X$.

Lema 3: $\{\alpha\} \notin E^*$

Demonstração: Imediato [Lema 2].

Lema 4: $\varnothing \notin E^*$

Demonstração: Imediato [Lema 2].

Lema 5: $W \notin E^*$

Demonstração: Demonstra-se, por indução na construção de E^*, que: $(\forall\ X \in E^*)\ X \neq W$.

base de indução: X é $||A_j||^M$ (para j=1,...,k). Tem-se:

$X=||A_j||^M$

$\Rightarrow$ [(P3)]

$z_j(A_j)=0$

$\Rightarrow$ [Lema 1(iv)]

$\eta_j \notin ||A_j||^M$ (i.e. $\eta_j \notin X$)

$\Rightarrow$

$X \neq W$

passo de indução: X é $Y \cap Z$ e $Y,Z \in E^*$. Suponha-se, por hipótese de indução, que $Y \neq W$ e $Z \neq W$. Então $Y \cap Z \neq W$, i.e. $X \neq W$.

Lema 6: $\varnothing \notin {}_xI_y^*$

Demonstração: Caso ${}_xI_y^* = \varnothing$: Imediato. Caso ${}_xI_y^* \neq \varnothing$, então existe j ($k<j\leq n$) tal que $x=x_j$ e $y=y_j$. Demonstra-se, por indução na construção de ${}_{x_j}I_{y_j}^*$, que "se $X \in {}_{x_j}I_{y_j}^*$ então $\delta_j \in X$":

base de indução: X é $||B||^M$ e $B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}^-(\Gamma)$. Tem-se:

$X = ||B||^M$ e $B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}^-(\Gamma)$

$\Rightarrow$ [(P2)]

$u_j(B)=1$

$\Rightarrow$ [Lema 1(iii)]

$\delta_j \in ||B||^M$ (i.e. $\delta_j \in X$)

passo de indução: X é $Y \cap Z$ e $Y,Z \in {}_{x_j}I_{y_j}^*$. Suponha-se, por hipótese de indução, que $\delta_j \in Y$ e $\delta_j \in Z$. Então $\delta_j \in Y \cap Z$, i.e. $\delta_j \in X$.

Lema 7: $W \notin {}_xI_y^*$

Demonstração: Caso ${}_xI_y^* = \varnothing$: Imediato. Caso ${}_xI_y^* \neq \varnothing$, então existe j ($k<j\leq n$) tal que $x=x_j$ e $y=y_j$. Demonstra-se, por indução na construção de ${}_{x_j}I_{y_j}^*$, que "se $X \in {}_{x_j}I_{y_j}^*$ então $X \neq W$":

base de indução: seja X é $||B||^M$ e $B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}^-(\Gamma)$. Tem-se:

$X = ||B||^M$ e $B \in \{A_{k+1},...,A_n\} \cap {}_{x_j}I_{y_j}^-(\Gamma)$

$\Rightarrow$

$(\exists\, i\ (k<i\leq n))\ B=A_i$

$\Rightarrow$ [(P3)]

$z_i(A_i)=0$

$\Rightarrow$ [Lema 1(iv)]

$\eta_i \notin ||A_i||^M$ (i.e. $\eta_i \in X$)

$\Rightarrow$

$X \neq W$

passo de indução: X é $Y \cap Z$ e $Y,Z \in {}_{x_j}I_{y_j}^*$. Suponha-se, por hipótese de indução, que $Y \neq W$ e $Z \neq W$. Então $Y \cap Z \neq W$, i.e. $X \neq W$.

Veja-se finalmente que $M \in C\mathrm{Lact}_N$. Note-se que na definição de *M* se mantêm as assinaturas das funções V, f_X, g_X e ${}_xh_Y$ (x,y = 1, ..., N). No que se segue X, Y são subconjuntos de W.

(i) $W \neq \varnothing$: Imediato, pela definição de W.

(ii) (te), i.e. $f_X(X) \subseteq X$:

O caso em que $X \notin E^*$ é imediato (pois $f_X(X)=\emptyset$). Caso $X \in E^*$ tem-se $f_X(X)=\{\alpha\}$[definição de f_X]. Então $f_X(X) \subseteq X$ [Lema 2].

(iii) (ce), i.e. $f_X(X) \cap f_X(Y) \subseteq f_X(X \cap Y)$:

Os casos em que $X \notin E^*$ ou $Y \notin E^*$ são imediatos (pois $\emptyset \subseteq f_X(X \cap Y)$). Caso $X \in E^*$ e $Y \in E^*$ tem-se $f_X(X)=f_X(Y)=\{\alpha\}$ [definição de f_X]. Então $f_X(X \cap Y)=\{\alpha\}$ [(e2)].

(iv) (noe), i.e. $f_X(W) = \emptyset$: Imediato [Lema 5; definição de f_X].

(v) (eenoe), i.e. $f_X(f_Y(X)) \subseteq W - f_X(X)$, para $x \neq y$:

Basta provar que $f_X(f_Y(X))=\emptyset$. Prova por *redução ao absurdo*: Suponha-se que $f_X(f_Y(X)) \neq \emptyset$. Nesse caso tem-se $f_Y(X) \in E^*$ [definição de f_X], e portanto ou se verifica $f_Y(X)=\{\alpha\}$ ou se verifica $f_Y(X)=\emptyset$ [definição de f_Y]. Consequentemente $\{\alpha\} \in E^*$ ou $\emptyset \in E^*$. Mas isto contradiz o Lema 3 ou o Lema 4.

(vi) (tg), (cg) e (nog): Provado respectivamente em (ii), (iii) e (iv), uma vez que $g_X(X)=f_X(X)$.

(vii) (eg), i.e. $f_X(X) \subseteq g_X(X)$: Imediato, uma vez que $g_X(X)=f_X(X)$.

(viii) (ci), i.e. ${}_Xh_Y(X) \cap {}_Xh_Y(Y) \subseteq {}_Xh_Y(X \cap Y)$: Prova análoga a (iii).

(ix) (nofi), i.e. ${}_Xh_Y(\emptyset)=\emptyset$: Imediato [Lema 6; definição de ${}_Xh_Y$].

(x) (noi), i.e. ${}_Xh_Y(W)=\emptyset$: Prova análoga a (iv), mas utilizando o Lema 7.

(xi) (trans.inf), i.e. ${}_Xh_Y(X) \cap g_Y(X) \subseteq g_X(X)$:

Os casos em que $X \notin E^*$ ou $X \notin {}_XI_Y{}^*$ são imediatos (pois $\emptyset \subseteq g_X(X)$). Caso $X \in E^*$ e $X \in {}_XI_Y{}^*$ tem-se $g_X(X)=g_Y(X)={}_Xh_Y(X)=\{\alpha\}$[definição de g_X; definição de ${}_Xh_Y$], pelo que se verifica (trans.inf).

(xii) as restantes restrições utilizando a composição das funções f_X e g_Y (x,y = 1, ..., N), i.e. (qg), (dir.control), (eeeg) e (gegg), provam-se de modo análogo a (v).

Portanto, uma vez que *M* satisfaz os requisitos apresentados na Definição 4.2, pode concluir-se que $M \in \mathrm{CLact}_N$. Pode pois concluir-se: ($\exists\ M \in C\mathrm{Lact}_N$) ($\exists\ \alpha$ em *M*) ($\forall\ B \in \{E_{X1}A_1,...,E_{Xk}A_k, {}_{Xk+1}I_{Yk+1}A_{k+1},...,{}_{Xn}I_{Yn}A_n\}$) $M,\alpha \vDash B$, que contradiz (8). ♦

O teorema anterior permite portanto "mover" o cálculo de coerência de um conjunto $\Gamma \subseteq \Delta_{Org}$ (de uma organização *Org*) da lógica Lact_N para a lógica proposicional PL (note-se mais uma vez que: se $\Gamma \subseteq \Delta_{Org}$ então $E^-(\Gamma) \subseteq \mathrm{Form}(L_{PL})$ e ${}_XI_Y{}^-(\Gamma) \subseteq \mathrm{Form}(L_{PL})$, para todo $x,y \in Ag$). E uma vez que os tableaux proposicionais são simultaneamente correctos, completos e decidíveis[100] (veja-se, e.g., [Fitting 90]), pode-se assegurar, com base no teorema anterior, que o conjunto de comportamentos possíveis B_{Org} de uma organização *Org* pode ser correctamente gerado

[100] I. e. é possível construir um algoritmo que responda em tempo finito à questão: $\Gamma \vdash_{PL} A$?.

computacionalmente.[101]

6.4.2 Bancada e Exemplos de Utilização

A "bancada" utilizada na automação da análise de organizações tem a seguinte arquitectura, ilustrada na Figura 6-8.

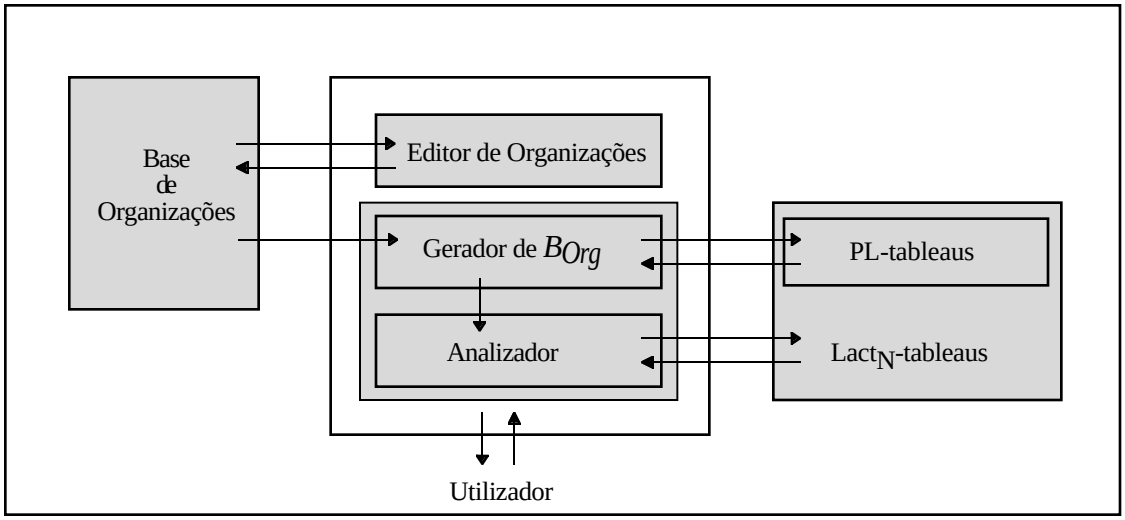


Figura 6-8: Arquitectura da "bancada"

A "bancada" é constituída fundamentalmente por quatro componentes:

(i) Base de Organizações;
(ii) Editor de Organizações;
(iii) Sistema de tableaux; e
(iv) Análise de organizações.

trocando informações no sentido das setas apresentadas na Figura 6-8.

Na "Base de Organizações" são armazenadas estruturas de organizações previamente definidas por um utilizador da "bancada". A definição de organizações é suportada pelo "Editor de Organizações" que disponibiliza as usuais operações de criação, modificação e eliminação de uma

[101] De facto, foi a necessidade de assegurar a geração correcta do conjunto de comportamentos possíveis B_{Org} que motivou a utilização de fórmulas de PL (e não de fórmulas de $Lact_N$) nas capacidades e canais de influência na estrutura de uma organização (veja-se a Definição 5.1, na página 93).

organização. Por sua vez, o "Sistema de tableaux" garante a automação de todas as deduções envolvidas na análise de uma organização. Finalmente, a componente "Análise de organizações" permite ao utilizador analisar uma organização de acordo com a análise proposta na secção 5.2. Esta componente gera o conjunto de comportamentos possíveis B_{Org} de uma organização *Org* com base no Resultado 6.12 e calcula respostas dos tipos R1, R2, R3' e R3", utilizando no cálculo dessas respostas o "Sistema de tableaux". Repare-se que na geração do conjunto de comportamentos possíveis B_{Org} apenas se utilizam PL-tableaux.

A "bancada" interage com o utilizador apresentando (em cada momento da sua execução) o conjunto de operações que podem ser seleccionadas. Na Figura 6-9 apresenta-se graficamente a "árvore de menus" utilizada na "bancada".[102]

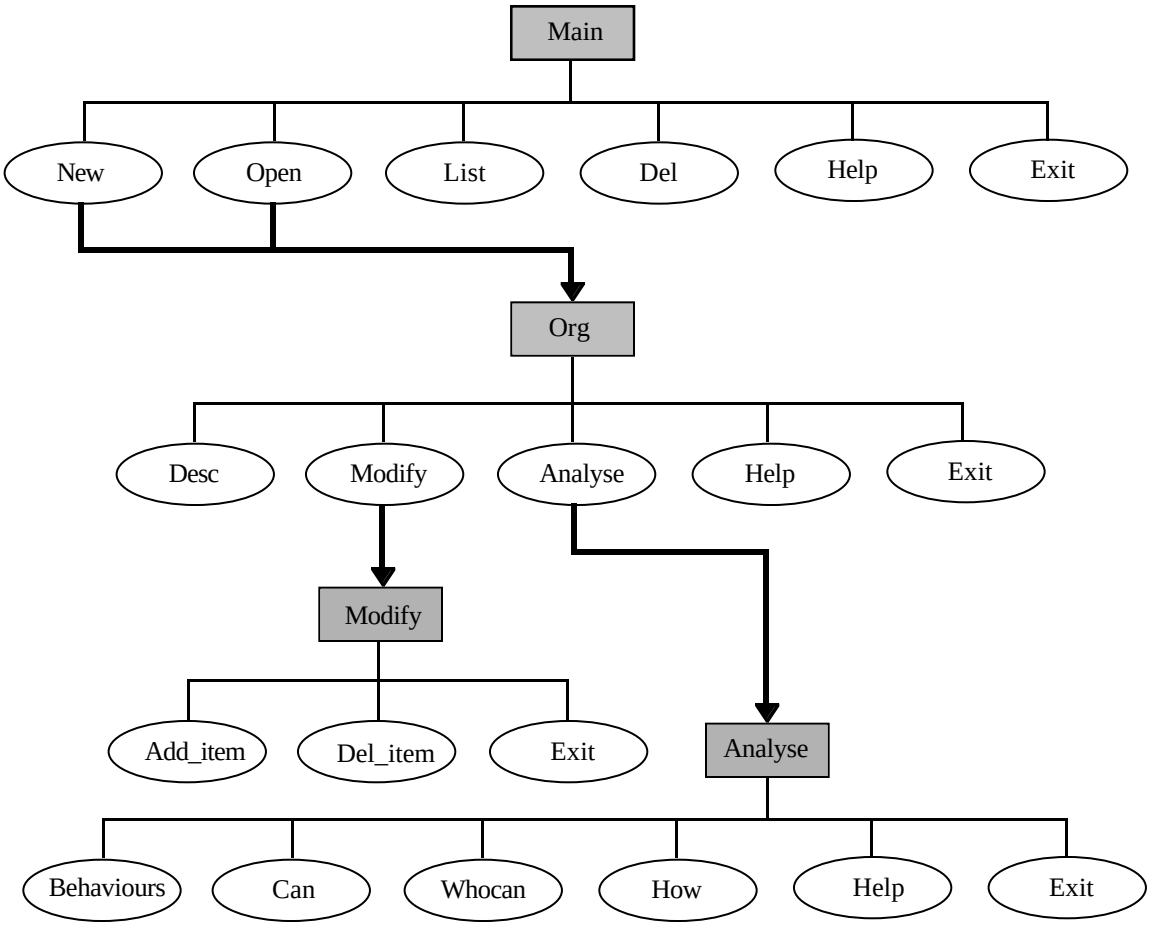


Figura 6-9: "Árvore de menus" da "bancada"

[102] Os menus estão em inglês uma vez que se pretendia apresentar esta bancada em encontros internacionais (como aliás já foi feito na *Third International Workshop on Deontic Logic in Computer Science (Deon' 96)*, Sesimbra, Portugal).

Cada rectângulo representa um menu e cada elipse uma das operações de um menu. As setas a "negrito" - "bold" - referem quando a selecção de uma operação conduz a outro menu.

Existem quatro menus na "bancada". O menu "Main" é apresentado no início da execução da "bancada" e disponibiliza as operações necessárias à selecção da organização a ser analisada (que pode estar ou não já definida na "Base de Organizações"). O menu "Org" é apresentado ao utilizador quando este selecciona uma organização no menu anterior ("organização activa"), e disponibiliza as operações necessárias à edição ou análise dessa organização. O menu "Modify" possibilita ao utilizador modificar a estrutura da "organização activa". E o menu "Analyse" disponibiliza as operações de análise da "organização activa".

Passa-se a descrever as várias operações de cada um dos menus da "bancada".

Menu "Main":

- "New" - Criação de uma nova organização (inicialmente sem capacidade e canais de influência). Activação do menu "Org" (no qual a organização criada é a "organização activa").
- "Open" - Selecção de uma organização armazenada na "Base de Organizações". Activação do menu "Org" (no qual a organização seleccionada é a "organização activa").
- "List" - Listagem de todas as organizações armazenadas na "Base de Organizações".
- "Del" - Eliminação de uma organização armazenada na "Base de Organizações".
- "Help" - Descrição das operações do menu "Main".
- "Exit" - Terminação da execução da "bancada".

Menu "Org":

- "Desc" - Apresentação da estrutura da "organização activa".
- "Modify" - Activação do menu "Modify".
- "Analyse" - Activação do menu "Analyse".
- "Help" - Descrição das operações do menu "Org".
- "Exit" - Reactivação do menu "Main" (deixa de haver "organização activa").

Menu “Modify”:

“Add_item” - Adicionar uma capacidade ou canal de influência à “organização activa”.

“Del_item” - Eliminar uma capacidade ou canal de influência da “organização activa”.

“Exit” - Reactivação do menu “Org”.

Menu “Analyse”:

“Behaviours” - Listagem do conjunto de comportamentos possíveis da “organização activa”.

“Can” - Cálculo de respostas do tipo R1 (na “organização activa”).

“Whocan” - Cálculo de respostas do tipo R2 (na “organização activa”). Durante a execução desta operação é pedido ao utilizador a lista de agentes que devem ser utilizados no cálculo deste tipo de respostas (veja-se “Frontier” à frente).

“How” - Cáculo de respostas do tipo R3' e R3" (na “organização activa”).

“Help” - Descrição das operações do menu “Analyse”.

“Exit” - Reactivação do menu “Org”.

Como ilustração da utilização da bancada à análise de organizações, considere-se a seguinte organização apresentada na Figura 6-10 e representada por org3=({a,b,c,d},{Cap_ap1,Cap_cp2, Cap_dp3},{a$>_{(p2\wedge p3)}$b,b$>_{p1}$c,b$>_{p2}$d}).

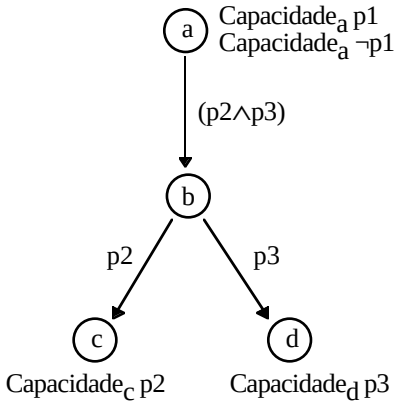


Figura 6-10: Diagrama da organização org3.

A sequência de interacções seguinte ilustra a utilização da "bancada" à análise da organização *org3* (onde se realça a "negrito" - "bold" - as respostas calculadas pela "bancada") :

```
org3 : desc, modify, analyse, help, exit ? -> help.
  desc      : current organization description;
  modify    : modify current organization;
  analyse   : analyse current organization;
  exit      : return main menu.

org3 : desc, modify, analyse, help, exit ? -> desc.
 Org_desc:
  [cap(c,p2),cap(d,p3),cap(a,p1),cap(a,neg p1),c(b,c,p2),c(b,d,p3),c(a,b,p2 and p3)]

org3 : desc, modify, analyse, help, exit ? -> analyse.

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> help.
  behaviours  : list of possible behaviours of the current organization;
  can         : analyse possibility in the current organization;
  whocan      : analyse who can do or ensure in the current organization;
  how         : analyse what must an agent do to ensure in the current organization;
  exit        : return previous menu.

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> behaviours.
 list of possible behaviours:
     empty behaviour
     i(b,d,p3)
     i(b,d,p3) and e(a,p1)
     e(c,p2) and e(d,p3) and e(a,neg p1) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(a,neg p1)
     e(c,p2) and e(d,p3) and e(a,p1) and i(b,c,p2) and i(a,b,p2 and p3)
     e(c,p2) and e(d,p3) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(c,p2)
     i(b,d,p3) and e(c,p2) and e(a,p1)
     e(d,p3) and e(a,neg p1) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(c,p2) and e(a,neg p1)
     e(d,p3) and e(a,p1) and i(b,c,p2) and i(a,b,p2 and p3)
     e(d,p3) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(d,p3)
     i(b,d,p3) and e(d,p3) and e(a,p1)
     e(c,p2) and e(a,neg p1) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(d,p3) and e(a,neg p1)
     e(c,p2) and e(a,p1) and i(b,c,p2) and i(a,b,p2 and p3)
     e(c,p2) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(d,p3) and e(c,p2)
     i(b,d,p3) and e(d,p3) and e(c,p2) and e(a,p1)
     e(a,neg p1) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and e(d,p3) and e(c,p2) and e(a,neg p1)
     e(a,p1) and i(b,c,p2) and i(a,b,p2 and p3)
     i(b,c,p2) and i(a,b,p2 and p3)
     i(b,d,p3) and i(a,b,p2 and p3)
     i(b,d,p3) and i(a,b,p2 and p3) and e(a,p1)
     e(c,p2) and e(d,p3) and e(a,neg p1) and i(b,c,p2)
     i(b,d,p3) and i(a,b,p2 and p3) and e(a,neg p1)
     e(c,p2) and e(d,p3) and e(a,p1) and i(b,c,p2)
     e(c,p2) and e(d,p3) and i(b,c,p2)
     i(b,d,p3) and i(a,b,p2 and p3) and e(c,p2)
     i(b,d,p3) and i(a,b,p2 and p3) and e(c,p2) and e(a,p1)
     e(d,p3) and e(a,neg p1) and i(b,c,p2)
     i(b,d,p3) and i(a,b,p2 and p3) and e(c,p2) and e(a,neg p1)
     e(d,p3) and e(a,p1) and i(b,c,p2)
     e(d,p3) and i(b,c,p2)
     i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3)
```

i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3) and e(a,p1)
e(c,p2) and e(a,neg p1) and i(b,c,p2)
i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3) and e(a,neg p1)
e(c,p2) and e(a,p1) and i(b,c,p2)
e(c,p2) and i(b,c,p2)
i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2)
i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2) and e(a,p1)
e(a,neg p1) and i(b,c,p2)
i(b,d,p3) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2) and e(a,neg p1)
e(a,p1) and i(b,c,p2)
i(b,c,p2)
i(b,d,p3) and i(b,c,p2)
i(b,d,p3) and i(b,c,p2) and e(a,p1)
e(c,p2) and e(d,p3) and e(a,neg p1) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(a,neg p1)
e(c,p2) and e(d,p3) and e(a,p1) and i(a,b,p2 and p3)
e(c,p2) and e(d,p3) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(c,p2)
i(b,d,p3) and i(b,c,p2) and e(c,p2) and e(a,p1)
e(d,p3) and e(a,neg p1) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(c,p2) and e(a,neg p1)
e(d,p3) and e(a,p1) and i(a,b,p2 and p3)
e(d,p3) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(d,p3)
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(a,p1)
e(c,p2) and e(a,neg p1) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(a,neg p1)
e(c,p2) and e(a,p1) and i(a,b,p2 and p3)
e(c,p2) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(c,p2)
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(c,p2) and e(a,p1)
e(a,neg p1) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(c,p2) and e(a,neg p1)
e(a,p1) and i(a,b,p2 and p3)
i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(a,p1)
e(c,p2) and e(d,p3) and e(a,neg p1)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(a,neg p1)
e(c,p2) and e(d,p3) and e(a,p1)
e(c,p2) and e(d,p3)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(c,p2)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(c,p2) and e(a,p1)
e(d,p3) and e(a,neg p1)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(c,p2) and e(a,neg p1)
e(d,p3) and e(a,p1)
e(d,p3)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3) and e(a,p1)
e(c,p2) and e(a,neg p1)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3) and e(a,neg p1)
e(c,p2) and e(a,p1)
e(c,p2)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2) and e(a,p1)
e(a,neg p1)
i(b,d,p3) and i(b,c,p2) and i(a,b,p2 and p3) and e(d,p3) and e(c,p2) and e(a,neg p1)
e(a,p1)

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> can.
formula -> g(b, p2 and p3).
YES! Consider, e.g. the possible behaviour:
i(b,d,p3) and i(b,c,p2) and e(d,p3) and e(c,p2)

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> can.
formula -> g(a, p1 and neg p1).
NO! There is no possible behaviour that satisfies this formula.[103]

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> whocan.
formula -> g(who, p2 and p3).
frontier -> [a,b,c,d].
who = [a,b]

analyse(org3) : behaviours, can, whocan, how, help, exit ? -> how.
formula -> g(a, p1 and p2 and p3).
Agent a must do:
mandatory acts: i(a,b,p2 and p3) and e(a,p1)
optional acts: No optional acts!

Veja-se, para finalizar, a sequência de interacções utilizada na análise da organização org4=({a,b,c},{Cap_ap1,Cap_ap2,Cap_ap3,Cap_bp1,Cap_cp2},{a>$_{p1}$b,a>$_{p2}$c}), apresentada na Figura 6-11:

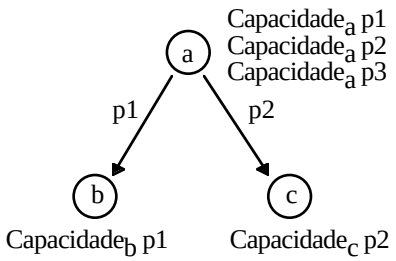


Figura 6-11: Diagrama da organização org4.

org4 : desc, modify, analyse, help, exit ? -> desc.
Org_desc:
[cap(a,p1),cap(a,p2),cap(a,p3),cap(b,p1),cap(c,p2),c(a,b,p1),c(a,c,p2)]

org4 : desc, modify, analyse, help, exit ? -> analyse.

analyse(org4) : behaviours, can, whocan, how, help, exit ? -> can.
formula -> g(a, p1 and p2 and p3).
YES! Consider, e.g. the possible behaviour:
e(a,p1) and e(a,p2) and e(a,p3)

analyse(org4) : behaviours, can, whocan, how, help, exit ? -> can.
formula -> g(b,p2).
NO! There is no possible behaviour that satisfies this formula.

analyse(org4) : behaviours, can, whocan, how, help, exit ? -> whocan.
formula -> g(who,p1).
frontier -> [a,b,c].
who = [a,b]

[103] Dado que não se provou a completude do sistema de tableaux $TLact_N$, a "bancada" deveria responder: "Penso que não! Pelo menos não consegui provar a existência de um comportamento possível que satisfaça esta fórmula.". Adaptações análogas deveriam ser consideradas nas restantes respostas apresentadas neste exemplo.

analyse(org4) : behaviours, can, whocan, how, help, exit ? -> how.
formula -> g(a, p1 and p2 and p3).
Agent a must do:
mandatory acts: e(a,p3)
optional acts: (e(a,p1) and e(a,p2)) or (i(a,b,p1) and e(a,p2)) or (i(a,c,p2) and e(a,p1)) or (i(a,c,p2) and i(a,b,p1))

7. Algumas Extensões à Lógica Proposta

Neste capítulo apresentam-se extensões à lógica proposta no capítulo 4. Foi aí proposta uma distinção clara entre as noções de "acção directa" e "acção indirecta" e discutiu-se a sua utilidade na caracterização da *Interacção entre Agentes*. Discute-se aqui uma outra distinção útil na caracterização da *Interacção entre Agentes*: a distinção entre as noções de *acção com-sucesso* e *acção não-necessariamente-com-sucesso*. Por outro lado retoma-se neste capítulo a discussão dos "princípios de transmissão da acção" deixada em aberto no capítulo 4.

Na secção 7.1 aprofunda-se a lógica de acção anteriormente proposta com base na distinção entre *acção com-sucesso* e *acção não-necessariamente-com-sucesso*. Dá-se especial ênfase ao conceito de *tentativa* e à sua caracterização formal. Salienta-se o poder expressivo adicional obtido pela introdução desse conceito e ilustra-se a sua utilização na descrição de actividades organizacionais envolvendo múltiplos agentes. Discutem-se também os problemas envolvidos na representação detalhada de actividades organizacionais e seu relacionamento com o nível de abstracção introduzido pelo conceito de *tentativa*. Mais uma vez a resolução destes problemas exige que se introduzam princípios de "transmissão da acção". Na secção 7.2 apresenta-se uma caracterização formal dos princípios de "transmissão da acção", através da utilização de um operador condicional semelhante ao proposto em [Jones & Sergot 96].

7.1 Acções com-sucesso vs. Acções não-necessariamente-com-sucesso

As noções de "acção directa" e de "acção indirecta" caracterizados respectivamente pelos operadores E_X e G_X são exemplos de noções de "acção com sucesso", no sentido em que se verifica intuitivamente "se o agente x produz ou assegura A, então A verifica-se", ideia esta formalmente caracterizada pela adopção do esquema (T) como princípio lógico para ambos os operadores. Este princípio permite portanto considerar que expressões da forma $OG_XA \wedge G_XA$ e $OG_XA \wedge \neg G_XA$ representam uma aceitável caracterização de "cumprimento" e "não cumprimento" da responsabilidade de obter A, respectivamente. Contudo, "não cumprimento" representado desta forma não evidencia o grau de participação por parte de um agente responsável na actividade organizacional requerida para obter A. E essa participação pode ser um elemento importante a ter em consideração na análise de diferentes situações de "não cumprimento". Considere-se, por

exemplo, a situação na qual um agente responsável *x* ordena a um subordinado a execução de uma tarefa específica *A*, mas em que o subordinado não obedece a essa ordem. Esta situação é obviamente diferente da situação em que o agente responsável nada faz. E, dependendo da organização, estas duas situações podem ter consequências substancialmente diferentes, e.g. a punição do subordinado no primeiro caso e a punição do agente responsável no segundo caso.

A introdução da distinção entre "acção com sucesso" e "acção não necessariamente com sucesso" permite distinguir as duas situações anteriores. Concretamente, pretende-se denominar por "acção não necessariamente com sucesso" as acções executadas por um agente na tentativa de alcançar um objectivo, onde essa tentativa pode não ter sucesso.

Para representar acções "não necessariamente com sucesso" propõe-se a utilização de um operador modal (indexado a agentes) H_X, em que expressões da forma H_XA são lidas por "o agente x tenta obter A". Utilizando este operador, os dois casos de "não cumprimento" de responsabilidade mencionados atrás passam agora a poder ser representados por expressões da forma $OG_XA\wedge\neg G_XA\wedge H_XA$ (no primeiro caso) e $OG_XA\wedge\neg H_XA$ (no segundo caso).

Para caracterizar logicamente a noção anterior propõe-se a utilização de um sistema do tipo EC, i.e. uma lógica com o seguinte axioma-esquema:

$(H_X\text{-C})$ $\quad (H_XA\wedge H_XB)\rightarrow H_X(A\wedge B)$

e a regra de inferência:

$(H_X\text{-rE})$ $\quad$ de $A\leftrightarrow B$, infere-se $H_XA\leftrightarrow H_XB$

em que o esquema (C) captura a intuição: "o agente x tenta obter tudo o que tenta obter isoladamente".

Comparando esta formalização com as formalizações propostas para os operadores E_X e G_X, repare-se que a ausência do esquema (T) é claramente justificada pelas intuições discutidas relativamente à noção de "acção não necessariamente com sucesso". Por outro lado, não se adopta aqui o esquema (No) para o operador H_X, uma vez que se considera que algum sentido possa ser dado a situações em que um agente tenta obter aquilo que é logicamente verdade.

Relativamente à interacção entre diferentes agentes, adopta-se para o operador H o esquema

$(H_X\text{-Q})$ $\quad H_XH_yA\rightarrow H_XA$

com a leitura "sempre que um agente x tenta obter que o agente y tenta obter A, o agente x também tenta obter A".[104]

Os princípios lógicos do operador H são simples. Contudo, outros princípios podem ser discutidos se se assumirem algumas hipóteses adicionais. Por exemplo, se se impuser "racionalidade" ao comportamento de cada agente, então deve adoptar-se o esquema (H_X-D), i.e. $H_XA \rightarrow \neg H_X \neg A$. Mais, se se assumir que todos os agentes coordenam as suas actividades de modo racional, pode ainda adoptar-se o esquema $H_XA \rightarrow \neg H_Y \neg A$. Estes "critérios de racionalidade" podem sem dúvida ser adoptados para descrever o comportamento de agentes (ou conjuntos) de agentes "ideais". No entanto podem considerar-se também situações em que tais critérios não podem ser adoptados.

Relativamente ao inter-relacionamento entre o operador H e os operadores de "sucesso" E e G, propõe-se aqui o esquema seguinte, reflectindo a ideia "assegurar é uma tentativa de obter (com sucesso)":

(GH) $G_XA \rightarrow H_XA$

Note-se que, deste princípio e do esquema (EG), pode-se deduzir o esquema $E_XA \rightarrow H_XA$.

No que respeita a outros relacionamentos entre os operadores E, G e H, propõe-se ainda a adopção dos seguintes princípios (à semelhança dos princípios (EEEG) e (GEGG) apresentados no capítulo 4):

(EGEH) $E_XG_YA \rightarrow E_XH_YA$
(GGGH) $G_XG_YA \rightarrow G_XH_YA$
(HEHG) $H_XE_YA \rightarrow H_XG_YA$
(HGHH) $H_XG_YA \rightarrow H_XH_YA$

Estes princípios têm que ser explicitamente adoptados uma vez que não se adoptou a regra (rM) - o fecho para a implicação - na caracterização do operador H, e no entanto pretende-se manter esse

[104] Note-se que, dadas as intuições discutidas a propósito do conceito de "acção indirecta", a adopção deste esquema sugere que a noção de "acção não necessariamente com sucesso" aqui formalizada pode ser entendida como um caso particular de "acções indirectas". Isto sugere que se possa discutir e formalizar um conceito de "acção directa não necessariamente com sucesso", caracterizado, e.g. pelo operador T_X e respeitando o princípio $T_XT_YA \rightarrow \neg T_XA$. No entanto omite-se aqui tal formalização pois não se constatou que esta pudesse contribuir para a representação de distinções relevantes no domínio de aplicação desta dissertação.

fecho para algumas das implicações representando inter-relacionamentos entre os vários conceitos de acção aqui formalizados. Por exemplo, o esquema (HGHH), que estabelece que "sempre que o agente x tenta obter que y assegura A, x também tenta obter que o agente y tenta obter A" expressa o fecho do operador H para o princípio (GH). Note-se que a partir destes princípios e do esquema (H_X-Q), pode deduzir-se entre outros os esquemas $H_XE_yA \rightarrow H_XA$ e $H_XG_yA \rightarrow H_XA$.

A caracterização semântica da lógica dos operadores E, G e H pode ser efectuada de modo análogo ao proposto na lógica $Lact_N$ apresentada na Definição 4.2, através de modelos mínimos, pelo que se omite aqui a sua caracterização.

Para além das distinções anteriormente salientadas, a introdução do operador H_X possibilita ainda expressar tipos mais fracos de obrigações/responsabilidades da forma OH_XA, apesar de estas apenas ocorrerem em organizações em circunstâncias nas quais há razões fortes para prever que a obtenção de A é difícil[105]. Por outro lado, obrigações da forma $O\neg H_XA$ podem ocorrer mais frequentemente, uma vez que representam situações nas quais até uma tentativa é proibida (e.g. o agente x está proibido de aceder à informação confidencial da empresa). Consequentemente, $O\neg G_XA \wedge O\neg H_XA$ e $O\neg G_XA \wedge \neg O\neg H_XA$ traduzem duas situações relevantes e distintas (claro que é de esperar que as propriedades lógicas das modalidades O, G_X e H_X tornem redundante o primeiro termo da conjunção em $O\neg G_XA \wedge O\neg H_XA$, i.e. devem ser válidas/teoremas fórmulas da forma $O\neg H_XA \rightarrow O\neg G_XA$[106]).

Por outro lado, através da combinação/interacção dos operadores modais E, G e H, podem salientar-se ainda algumas noções importantes de *controlo-interpessoal* relativamente a A. Analogamente às combinações E_XE_yA, E_XG_yA, G_XE_yA e G_XG_yA, expressões da forma E_XH_yA e G_XH_yA também podem ser utilizadas para representar espécies de exercícios de "influências com sucesso", mas desta vez apenas no sentido em que x obtém como resultado que y tenta obter A. No entanto, expressões da forma H_XE_yA, H_XG_yA representam "influências não necessariamente com sucesso" de x para y.

---

[105] De facto, uma característica comum das responsabilidades organizacionais é a de que estas são descritas de forma a deixar claro que se requer que os agentes responsáveis tenham sucesso.

[106] Que se verificará "automaticamente" - dado o esquema (GH) - se a lógica de O for normal. No entanto, nesta dissertação, optou-se por não discutir qual a lógica apropriada para os operadores deônticos, aspecto que tem sido objecto de intenso e interessante debate (veja-se e.g. [Meyer & Wieringa 93b; Jones & Sergot 94; Brown & Carmo 96; Δeon'98]).

O poder expressivo adicional obtido com a introdução do operador H_X pode ainda ser salientado pelo facto de se obter nesta lógica - com os operadores E_X e G_X - sete posições-de-acção para um agente, isto se se considerar o esquema (H_X-D):

(A1) E_XA
(A2) $E_X\neg A$
(A3) $G_XA\wedge\neg E_XA$
(A4) $G_X\neg A\wedge\neg E_X\neg A$
(A5) $H_XA\wedge\neg G_XA$
(A6) $H_X\neg A\wedge\neg G_X\neg A$
(A7) $\neg H_XA\wedge\neg H_X\neg A$

em que neste caso (A7) representa "x encontra-se passivo relativamente a A", no sentido em que nem sequer tentou obter, quer A, quer ¬A. Esta parece ser uma representação mais adequada da "passividade" de um agente relativamente a A.

No entanto, sem a adopção do "critério de racionalidade" imposto pelo esquema (H_X-D), obtém-se treze posições.

De modo a ilustrar as capacidades de inferência da lógica proposta, considere-se o seguinte exemplo de uma organização com três agentes *a*, *b* e *c*, onde os agentes têm apenas as capacidades indicadas na Figura 7-1. Assume-se ainda que o agente *a* tem influência/poder efectivo sobre os agentes *b* e *c*, relativamente a *p1* e *p2*, respectivamente.

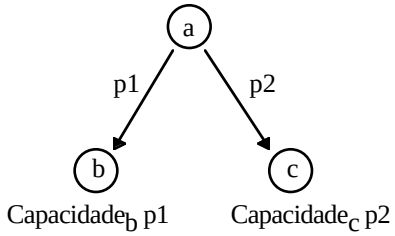


Figura 7-1: Diagrama de uma organização: ilustração das capacidades de inferência

Nesta estrutura organizacional simples, existe apenas uma possibilidade para que *a* assegure *p1*∧*p2*: *a* deve exercer uma influência com sucesso nos agentes *b* e *c* para assegurarem *p1* e *p2*, respectivamente. De facto, os axiomas-esquemas e as regras de inferência propostas para os operadores E e G permitem a seguinte dedução:

$\{E_aG_bp1, E_aG_cp2\} \vdash G_a(p1 \wedge p2)$

Considere-se agora a situação em que o agente *a* exerce influência nos agentes *b* e *c* para assegurarem *p1* e *p2*, mas em que o agente *c* não obtém sucesso na obtenção de *p2*, ao passo que o agente *b* obtém sucesso na obtenção de *p1*. Pode afirmar-se neste caso que, na sua tentativa para obter *p1∧p2*, o agente *a* apenas assegura *p1*. E de facto a dedução seguinte também é suportada pela lógica proposta:

$\{E_aG_bp1, E_aH_cp2\} \vdash G_ap1 \wedge H_a(p1 \wedge p2)$

Por último, considere-se a seguinte variante do cenário anterior: o agente *c* não executou qualquer acção. Este caso gera conclusões idênticas ao caso anterior, apesar de haver agora uma diferença relativamente à influência exercida pelo agente *a* no agente *c*: o agente *a* tentou que o agente *c* assegurasse *p2*, mas não teve sucesso. Existe apenas um possível candidato para representar esta tentativa de influência por parte do agente *a* em termos das combinações dos operadores de acção anteriormente introduzidos: H_aG_cp2. A dedução seguinte também é suportada pela lógica proposta:

$\{E_aG_bp1, H_aG_cp2, \neg H_cp2\} \vdash G_ap1 \wedge H_a(p1 \wedge p2)$

As hipóteses utilizadas nas deduções anteriores representam as situações mencionadas de acordo com o nível de abstracção considerado, e.g. expressando a influência com sucesso do agente *a* sobre o agente *c* através da expressão E_aG_cp2 e expressando a influência sem sucesso do agente *a* sobre o agente *c* através da expressão H_aG_cp2. Repare-se que E_aG_cp2 e H_aG_cp2 representam o que se verificou *à posteriori* como resultado das acções do agente *a*, embora nas situações anteriores se pudesse considerar que o agente *a* agiu do mesmo modo. Se nesta organização for assumido que o agente *a* exercita as suas influências através da *atribuição de responsabilidades*, essa suposição levaria a propor os seguintes conjuntos de hipóteses como uma descrição mais detalhada das situações anteriores: $\{E_aOG_bp1, E_aOG_cp2, E_bp1, E_cp2\}$, $\{E_aOG_bp1, E_aOG_cp2, E_bp1, H_cp2, \neg E_cp2\}$, $\{E_aOG_bp1, E_aOG_cp2, E_bp1, \neg H_cp2\}$. No entanto, a lógica proposta não permite obter as mesmas conclusões anteriores destes conjuntos de hipóteses, nem mesmo no caso de se assumir o princípio

(Atrib.resp') $E_xOG_yA \rightarrow H_xG_yA$

reflectindo a ideia “atribuições de responsabilidades são casos particulares de influências não necessariamente com sucesso”.

A razão de ser deste problema reside, mais uma vez, no facto de não existir nenhum princípio de “transmissão da acção” relacionando os actos de influência com as outras acções na “cadeia de influências com sucesso” suportando uma “acção indirecta”.

Se for aceite, por exemplo, o princípio

(Trans.inf') $(H_xG_yA \wedge G_yA) \rightarrow G_xA$

para representar a “transmissão da acção” envolvida em muitas organizações institucionalizadas, então este princípio (e o princípio (Atrib.resp')) permitirá a dedução das conclusões esperadas das descrições mais concretas apresentadas acima, i.e.:

$\{E_aOG_bp1, E_aOG_cp2, E_bp1, E_cp2\} \vdash G_a(p1 \wedge p2)$
$\{E_aOG_bp1, E_aOG_cp2, E_bp1, H_cp2, \neg E_cp2\} \vdash G_ap1 \wedge H_a(p1 \wedge p2)$
$\{E_aOG_bp1, E_aOG_cp2, E_bp1, \neg H_cp2\} \vdash G_ap1 \wedge H_a(p1 \wedge p2)$

Repare-se que a fórmula H_xG_yA desempenha no princípio (Trans.inf') o mesmo papel que a fórmula ${}_xI_yA$ desempenha no princípio (Trans.inf) proposto na página 75. Ambas representam “influências não necessariamente com sucesso” de x para y. Aliás, foi o facto de o operador ${}_xI_y$ relacionar dois agentes que sugeriu que se investigasse um conceito de acção “mais primitivo” que mencionasse apenas um agente e que permitisse representar uma influência da forma ${}_xI_yA$. Um candidato natural para representar ${}_xI_yA$ é obviamente H_xG_yA, embora exija maior reflexão.

Outros princípios de “transmissão da acção” podem também ser discutidos neste contexto, por forma a suportar deduções a partir de descrições mais concretas de actividades organizacionais. No entanto, como foi dito anteriormente, qualquer princípio que seja proposto depende sempre de hipóteses adicionais acerca de cada organização particular, pelo que não podem ser considerados válidos em geral.

7.2 Caracterização de Princípios de Transmissão da Acção

Como foi dito anteriormente, os princípios de “transmissão da acção” podem variar de organização para organização. Contudo, adoptou-se o princípio (Trans.inf) como princípio válido da lógica $Lact_N$ para possibilitar o tipo de análise de organizações proposta no capítulo 5, o que

inviabilizou a análise de organizações utilizando outros princípios de "transmissão da acção". Uma solução possível para este problema consiste naturalmente em adoptar diferentes lógicas com base em diferentes princípios de "transmissão da acção". No entanto, esta opção também não é apropriada para descrever casos em que organizações interagem de acordo com diferentes princípios, ou mesmo casos em que as acções de um ou mais agentes podem estar sujeitas a princípios diferentes (e.g. nos casos em que esses agentes interagem no âmbito de duas ou mais organizações).

A ideia de se considerar "sistemas de significado" como um elemento importante de análise de organizações, adoptada recentemente por investigadores na área da Teoria das Instituições (veja-se, e.g. [Scott & Meyer 94]), vem no entanto sugerir uma solução para este problema. Esses autores defendem que agentes (individuais ou colectivos) e as suas acções (individuais ou colectivas) podem ser vistos como construções sociais cujo significado depende de elementos simbólicos de vários tipos (representacionais, constitutivos ou normativos).

Adoptando esta perspectiva, pode dizer-se a adopção de um particular princípio de "transmissão da acção" por parte de uma organização reflecte como a organização interpreta a importância das acções executadas pelos seus agentes, i.e. como a organização interpreta o significado dessas acções, ou seja como essas acções "contam" para a organização.

Considere-se, por exemplo, os dois princípios de "transmissão da acção" seguintes (anteriormente discutidos no capítulo 4):

(Trans.atrib.resp1) $(E_xOG_yA \wedge G_yA) \rightarrow G_xA$
(Trans.não.importa) $(OG_xA \wedge A) \rightarrow G_xA$

Uma organização *o* utilizando o princípio (Trans.atrib.resp1) "valoriza" essencialmente as influências exercidas por atribuições de responsabilidades. Neste caso, as situações caracterizadas por $E_xOG_yA \wedge G_yA$ "contam para a organização *o* como" G_xA. Já no caso em que uma organização *o* adopta o princípio (Trans.não.importa), esta apenas centra a sua atenção nos resultados esperados, não valorizando qualquer acção em particular. Neste caso, as situações caracterizadas por $OG_xA \wedge A$ "contam para a organização *o* como" G_xA.

Para representar as ideias anteriores, adopta-se aqui um operador condicional $\Rightarrow_o$, onde expressões da forma $A\Rightarrow_oB$ são lidas por "para a organização o, A conta como B", adaptando neste contexto a ideia apresentada em [Jones & Sergot 96][107] e aí formalmente caracterizada.

Os princípios de "transmissão da acção" anteriores passam agora a poder ser representados da seguinte maneira:

(Trans.atrib.resp1') $(E_xOG_yA\wedge G_yA)\Rightarrow_oG_xA$
(Trans.não.importa') $(OG_xA\wedge A)\Rightarrow_oG_xA$

A utilidade do operador $\Rightarrow_o$ não se resume apenas à caracterização de princípios de "transmissão da acção". Este operador pode também ser utilizado para representar tarefas abstractas e o modo como uma particular organização *o* "assume" a decomposição dessas tarefas. Por exemplo, uma expressão da forma $((A\wedge B)\Rightarrow_oC)$ - representando "$A\wedge B$" conta para a organização *o* como um modo de obter C" - pode ser utilizada para representar uma possível decomposição de C, e uma expressão da forma $(A\Rightarrow_oC)\wedge(B\Rightarrow_oC)$ pode ser utilizada para representar dois modos alternativos para obter C.

Adopta-se para o operador $\Rightarrow_o$ uma lógica condicional do tipo CE (de acordo com a classificação de Chellas)[108], e com os axiomas-esquemas adicionais:

(CC) $((A\Rightarrow_oB)\wedge(A\Rightarrow_oC))\rightarrow(A\Rightarrow_o(B\wedge C))$
(CA) $((A\Rightarrow_oB)\wedge(C\Rightarrow_oB))\rightarrow((A\vee C)\Rightarrow_oB)$

Para uma discussão detalhada dos princípios lógicos a adoptar para o operador $\Rightarrow_o$, bem como a sua caracterização semântica, veja-se [Jones & Sergot 96].

---

[107] Em [Jones & Sergot 96], é proposta a leitura "de acordo com a instituição *o*, A conta como B" para expressões da forma $A\Rightarrow_oB$. De modo a tratar múltiplas instituições, Jones e Sergot introduzem adicionalmente um operador modal proposicional D_o do tipo rKD, onde expressões da forma D_oA são lidas por "na instituição *o* vigora A". Impõem ainda o esquema $(A\Rightarrow_oB)\rightarrow D_o(A\rightarrow B)$ como relacionamento adequado entre estes operadores.

[108] De acordo com [Chellas 80], uma lógica condicional (relativamente ao operador $\Rightarrow$) do tipo CE é uma lógica fechada para as seguintes regras: (rCEA) de $A\leftrightarrow A'$, infere-se $(A\Rightarrow B)\leftrightarrow(A'\Rightarrow B)$; e (rCEC) de $B\leftrightarrow B'$, infere-se $(A\Rightarrow B)\leftrightarrow(A\Rightarrow B')$.

Veja-se agora como a introdução deste operador pode suportar a caracterização de diferentes organizações utilizando diferentes "sistemas de significado" e como podem ser deduzidas interpretações neste contexto.

O operador anterior permite caracterizar diferentes "regras" de interpretação por parte de diferentes organizações, e.g. através de fórmulas da forma $(A \Rightarrow_{o1} B) \wedge (A \Rightarrow_{o2} C)$, sendo *o1* e *o2* diferentes organizações. Possibilita ainda que essas "regras" sejam conflituosas. Considere-se, e.g. fórmulas da forma $(A \Rightarrow_{o1} B) \wedge (A \Rightarrow_{o2} \neg B)$. Mas como é que podem ser deduzidas interpretações neste contexto a partir das "regras" anteriores ?

Repare-se que, intuitivamente, se se verifica $A \Rightarrow_o B$, então para a organização o, se A é verdade também B é verdade. Adaptando neste contexto a mesma estratégia apresentada em [Jones & Sergot 96], introduz-se aqui um operador modal proposicional (relativizado a organizações) D_o, em que expressões da forma $D_o A$ podem ser lidas por "A é reconhecido pela organização *o*". Tal como [Jones & Sergot 96], adopta-se como lógica deste operador um sistema de lógica modal normal do tipo rKD, i.e. um sistema lógico com os seguintes axiomas-esquema e regra de inferência:

(D_o-K)	$D_o(A \rightarrow B) \rightarrow (D_o A \rightarrow D_o B)$
(D_o-D)	$D_o A \rightarrow \neg D_o \neg A$
(D_o-rN)	de A, infere-se $D_o A$

Adicionalmente, adoptam-se os seguintes axiomas-esquema:

($\Rightarrow_o$D)	$A \Rightarrow_o B \rightarrow D_o(A \rightarrow B)$ [109]
(Trans.sign1)	$A \Rightarrow_o B \rightarrow D_o(E_X A \rightarrow E_X B)$
(Trans.sign2)	$A \Rightarrow_o B \rightarrow D_o(G_X A \rightarrow G_X B)$

($\Rightarrow_o$D) representa intuitivamente que " se A conta como B na organização *o*, então se *o* reconhece que A é verdade também reconhece que B é verdade". Os esquemas (Trans.sign1) e (Trans.sign2) reflectem o facto de se considerar aqui a relação "conta como" para representar decomposição de tarefas.

[109] Mas repare-se que é fundamental não se ter $A \Rightarrow_o B \rightarrow (A \rightarrow B)$, uma vez que se pretende considerar a representação simultânea de organizações com políticas contraditórias face ao significado de A. Pela mesma razão não se considera o esquema (D_o-T).

Para ilustrar as capacidades de inferência adicionalmente obtida pela introdução das noções anteriores, considere-se o seguinte exemplo de uma organização *o1* com quatro agentes *a, b, c* e *d*, onde estes agentes têm apenas as capacidades indicadas na Figura 7-2, e onde o agente *a* tem a responsabilidade de *q* (que pode ser decomposta por *p1*, *p2* e *p3*). Assume-se também que o agente *a* tem poder efectivo (através da atribuição de responsabilidades) sobre o agente *b* relativamente a *q* e *b* tem poder efectivo sobre os agentes *c* e *d* relativamente a *p1* e *p2*, respectivamente. Adicionalmente, assume-se também que a organização *o1* adopta o princípio de "transmissão da acção" caracterizado pelo esquema (Trans.atrib.resp1') - i.e. $(E_x OG_y A \wedge G_y A) \Rightarrow_{o1} G_x A$ - e que todas as acções dos agentes *a, b, c , d* são reconhecidas pela organização *o1*.

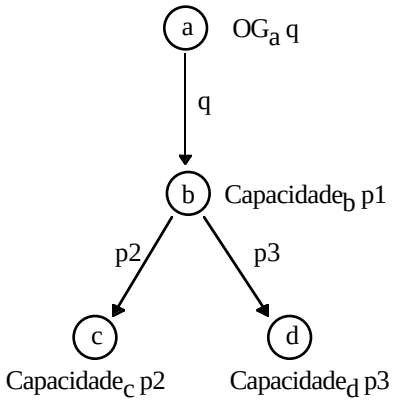


Figura 7-2: Diagrama da organização o1: ilustração das capacidades de inferência.

O sistema lógico proposto permite a dedução seguinte:

$$\Gamma \vdash D_{o1}(OG_a q \wedge G_a q)$$

com $\Gamma = \{(E_x OG_y A \wedge G_y A) \Rightarrow_{o1} G_x A: A \in Form(L)\} \cup \{((p1 \wedge p2 \wedge p3) \Rightarrow_{o1} q), D_{o1}(E_a OG_b q \wedge E_b OG_c p2 \wedge E_b OG_d p3 \wedge E_b p1 \wedge E_c p2 \wedge E_d p3), D_{o1} OG_a q\}$. Portanto, nestas circunstâncias, conclui-se que é reconhecido pela organização *o1* que o agente *a* cumpre a sua responsabilidade.

Suponha-se agora outra organização *o2*, e que esta adopta o princípio de "transmissão da acção" (Trans.não.importa') - i.e. $(OG_x A \wedge A) \Rightarrow_{o2} G_x A$. Considere-se o exemplo da organização descrita na Figura 7-2, mas desta vez aplicado a *o2*. O sistema lógico proposto suporta a seguinte dedução:

$\Gamma \vdash D_{o2}(OG_a q \wedge G_a q)$

com $\Gamma = \{(OG_x A \wedge A) \Rightarrow_{o2} G_x A: A \in Form(L)\} \cup \{((p1 \wedge p2 \wedge p3) \Rightarrow_{o2} q), D_{o2} OG_a q, D_{o2}(E_b p1 \wedge E_c p2 \wedge E_d p3)\}$. Esta conclusão não poderia ser obtida na organização *o1* com o princípio de "transmissão da acção" (Trans.atrib.resp1').

Os exemplos anteriores ilustram como esta lógica pode ser utilizada para tornar explícito o facto de que aquilo que "conta como" para a organização *o2* como modos de o agente *a* cumprir a sua responsabilidade pode não ser interpretado da mesma maneira pela organização *o1*.

Estes exemplos simples também sugerem como a utilização destas ferramentas formais podem ser utilizadas para detectar possíveis incompatibilidades entre diferentes organizações relativamente às políticas ou regulamentações que estas adoptam.

As potencialidades deste tipo de representações serão objecto de investigação futura. Por exemplo, será interessante considerar representações da forma

(P1) $(OG_x A \wedge G_x A) \Rightarrow_o C1$
(P2) $(OG_x A \wedge \neg G_x A \wedge H_x A) \Rightarrow_o C2$
(P3) $(OG_x A \wedge \neg H_x A) \Rightarrow_o C3$

para distinguir diferentes políticas adoptadas por uma organização *o* acerca do "cumprimento" ou "não cumprimento" de responsabilidades, em que C1, C2 e C3 podem representar recompensas, sanções, ou mesmo novas responsabilidades. Por exemplo, se uma organização *o* adopta que o "não cumprimento" de uma responsabilidade tem o mesmo tipo de consequências, independentemente do grau de participação do agente x, então nesse caso dever-se-á considerar para C2 e C3 fórmulas equivalentes.

8. Conclusões e Trabalho Futuro

Finaliza-se esta dissertação apresentando uma síntese do trabalho apresentado, realçando as suas contribuições e identificando perspectivas de trabalho futuro.

O trabalho exposto nesta dissertação foi motivado pela necessidade de investigar modelos organizacionais explícitos onde os conceitos organizacionais relevantes fossem formalmente caracterizados e permitissem a análise sistemática de organizações. Adoptou-se a perspectiva dos *sistemas normativos*, segundo a qual a caracterização de uma organização consiste fundamentalmente na caracterização normativa da interacção entre agentes relevantes para a organização. De entre vários conceitos organizacionais úteis para descrever esta interacção escolheu-se o conceito de *responsabilidade organizacional* e procedeu-se ao estudo de um inter-relacionamento preciso, e suficientemente abstracto, entre estas responsabilidades e as formas de interacção entre agentes disponíveis em cada organização. Isto orientou a investigação para a discussão de noções de acção envolvidas no conceito de *responsabilidade organizacional* e ao estudo de lógicas modais de acção adequadas à sua caracterização formal. Neste âmbito, foi discutida a utilidade de um distinção clara (inexistente em contribuições anteriores) entre a noção de *acção directa* (caracterizada pelo operador E_X) e a noção de *acção indirecta* (caracterizada pelo operador G_X), utilizando-se esta última, em combinação com um apropriado operador de obrigação O, na caracterização do conceito de *responsabilidade organizacional*, representando assim expressões da forma "o agente x é responsável por A" por OG_XA (i.e. "é obrigatório que o agente x assegure A"). Paralelamente, salientou-se ainda o poder expressivo adicional obtido pela introdução dos operadores E_X e G_X, nomeadamente na caracterização de conceitos de *controlo interpessoal* e na caracterização de *posições-de-acção para um agente*.

Procedeu-se entretanto ao estudo sintáctico-axiomático dos conceitos de acção mencionados e dos seus inter-relacionamentos, por forma a propor um sistema lógico capaz de suportar deduções com o adequado grau de abstracção necessário à análise de actividades organizacionais envolvendo múltiplos agentes, bem como o relacionamento destas com a satisfação de *responsabilidades organizacionais*. A necessidade de suportar deduções a partir da representação detalhada de actividades organizacionais e seu relacionamento com o nível de abstracção introduzido pelo conceito de *acção indirecta* levou ainda à introdução da noção adicional de *acção directa de influência* (caracterizada pelo operador $_XI_Y$) e à discussão de *princípios de transmissão de acção* (dos quais se escolheu um como princípio lógico a adoptar na análise de organizações). Passou-se então à caracterização detalhada do sistema lógico $Lact_N$ (utilizando os operadores E_X, G_X e $_XI_Y$,

para x, y = 1, ..., N) e ao estudo das suas propriedades (incluindo a sua correcção e completude face à classe de modelos relevante).

Após o estudo anterior, abordou-se a utilização dos conceitos anteriormente caracterizados no âmbito da especificação e análise de organizações. Foi proposta uma estrutura organizacional na qual cada organização é descrita como um conjunto de agentes com capacidades de executar tarefas predefinidas e com poderes efectivos (também estes predefinidos) para responsabilizar/influenciar outros agentes para executar tarefas. Os aspectos de análise de organizações foram então abordados com base nos conceitos de acção e dos seus inter-relacionamentos (embora implícitos) com o conceito de *responsabilidade organizacional*. Embora simples, o modelo de organização proposto permite suportar alguns aspectos de análise cujo interesse potencial abrange aspectos relacionados com o desenho de organizações e a análise das suas propriedades: análise da atribuição de responsabilidades organizacionais, análise da distribuição de tarefas, e suporte a sistemas periciais que guiem o acesso a utentes de uma organização. Estes aspectos de análise foram suportados através da resposta aos seguintes tipos de questões, mais simples,

(Q1) pode o agente x assegurar A?

(Q2) quais os agentes que podem assegurar A?

(Q3) que deve o agente x fazer (directamente) de modo a assegurar A?

respostas essas formalizadas com base nas deduções obtidas pela lógica Lact_{N} a partir dos diferentes conjuntos de "actos directos" (*comportamentos possíveis*) estritamente de acordo com as capacidades e os poderes especificados para cada um dos agentes da organização em análise. As formalizações propostas para as respostas às questões anteriores foram ainda discutidas com base nas intuições fornecidas pelo menor modelo mínimo canónico para a lógica Lact_{N}.

Abordou-se de seguida a automação da lógica Lact_{N} com o objectivo de suportar computacionalmente as respostas às questões anteriores. Foi apresentado um método de prova automática - o sistema TLact_{N} - baseado em *tableaux semânticos* para a lógica Lact_{N}, discutidas as suas propriedades e os critérios adoptados na sua implementação. Foram ainda apresentados vários sistemas de tableaux para automatizar as deduções de alguns sistemas de lógica (uni)modal clássica proposicional não-normal para os quais não existem actualmente métodos de prova. Trata-se de outra contribuição importante desta dissertação, uma vez que se verificou a inexistência de métodos para automatizar lógicas modais clássicas não-normais (como é o caso da lógica de acção desenvolvida neste trabalho). Contudo, repare-se que embora se tenha provado a correcção dos sistemas de tableaux propostos, não foi possível provar a sua completude.

Passou-se então à discussão da automação da análise de organizações. A ausência do resultado de completude do sistema TLact_{N} exigiu que se estudassem formas de garantir que o conjunto de *comportamentos possíveis* de uma organização fosse correctamente gerado computacionalmente (recorde-se que cada *comportamento possível* foi caracterizado em termos da coerência na lógica

$Lact_N$). Provou-se entretanto um resultado adicional que permitiu arranjar uma solução de compromisso para este problema, consistindo em "mover" o cálculo da coerência da lógica $Lact_N$ para a lógica proposicional (onde os tableaux são completos).

A análise de organizações proposta é actualmente suportada por uma "bancada" implementada em Prolog (veja-se o código no Anexo 2 e exemplos da sua utilização na secção 6.4). Esta "bancada" permite ilustrar como a utilização de um modelo organizacional explícito pode ser empregue na análise sistemática de organizações.

Finalizou-se este trabalho com a discussão de algumas extensões à lógica $Lact_N$, através da introdução do conceito de *tentativa* e de um operador condicional para caracterizar formalmente os princípios de *transmissão de acção*. Como se viu, estas extensões permitem introduzir um poder expressivo adicional na caracterização da interacção entre agentes em organizações e analisar com mais detalhe as condições de cumprimento de *responsabilidades organizacionais*.

A aplicação da estrutura organizacional proposta à descrição e análise de organizações é obviamente limitada. Apesar de simples, a estrutura organizacional proposta deve ser vista como uma estrutura básica facilmente extensível a outras estruturas organizacionais mais elaboradas, com outros conceitos capazes de aumentar as potencialidades de análise de organizações. Vislumbram-se duas vias complementares para o fazer e que serão alvo de investigação futura:

1. Extensões utilizando mais conceitos primitivos capazes de enriquecer a descrição de uma organização;
2. Extensões considerando novas abstracções capazes de permitir uma especificação rápida de uma organização à custa de conceitos mais primitivos;

Relativamente à via de investigação (1), é prioritário considerar a introdução da componente deôntica na estrutura organizacional. Este será o primeiro passo a dar na direcção da caracterização explícita de *responsabilidades organizacionais* e que permitirá a análise da sub-responsabilização e da sobre-responsabilização dos vários agentes da organização. Mais ainda, a introdução da componente deôntica permitirá também outros tipos úteis de análise no âmbito do processo de desenho de organizações (veja-se a este respeito os trabalhos em [Ramos & Fiadeiro 97, 98] baseados na teoria do diagnóstico de Reiter [Reiter 87]).

Outro aspecto prioritário é o de considerar extensões utilizando o operador condicional $\Rightarrow_o$ (apresentado na secção 7.2) para representar as regras de interpretação de uma organização *o*. Como se viu, a introdução deste operador permitirá representar diferentes princípios de *transmissão da acção*, potenciando portanto a análise simultânea de várias organizações (ou mesmo de vários órgãos da mesma organização, mas utilizando diferentes regras de interpretação). Para além disso, este operador permitirá representar as formas como uma organização assume a decomposição de tarefas e as diferentes vias para as obter.

Ainda no âmbito da via de investigação (1), urge desenvolver uma teoria de agentes colectivos que permita suportar a caracterização de entidades organizacionais constituídas por dois ou mais agentes, e.g. *conselhos*, *assembleias*, *departamentos*, etc. Mas como lidar com noções normativas neste contexto? Por exemplo, o que significa dizer que "o conselho de administração (de uma organização) é responsável por A"? E como determinar a satisfação de tais responsabilidades a partir de uma representação detalhada de actividades organizacionais descritas pelas acções de cada um dos agentes individuais? A resolução deste problema passa pela caracterização de agentes colectivos, de acções colectivas e de princípios de *transmissão da acção* capazes de relacionar acções individuais e colectivas.

Nesta dissertação as organizações foram especificadas e analisadas numa perspectiva institucional, considerando que as formas de interacção entre agentes estão predefinidas em cada organização. Uma outra perspectiva de análise seria a de considerar os aspectos subjectivos que determinam a interacção entre agentes por forma a analisar as relações sociais emergentes numa organização (e.g. o poder de influência obtido por um agente, resultante da sua anterior colaboração com outros agentes). Neste caso, na descrição de organizações, seria essencial considerar outros conceitos, tais como crenças, intenções e objectivos de cada agente da organização (veja-se [Castelfranchi 90, 96; Falcone & Castelfranchi 97]).

Relativamente à via de investigação (2), é necessário investir em linguagens de especificação com notações adicionais, definidas à custa dos conceitos primitivos considerados na estrutura organizacional, que facilitem a especificação de organizações mais complexas. A título ilustrativo, a introdução da notação $x>_{\Phi}y$ poderia ser utilizada para descrever casos em que existe um canal de influência de x para y relativamente a qualquer tarefa de uma classe Φ. É necessário também considerar linguagens de especificação que utilizem noções normalmente utilizadas em organizações. Dever-se-á portanto investir na caracterização desses conceitos, utilizando os conceitos primitivos como blocos básicos para a sua caracterização.

A abordagem proposta não permite especificar e analisar muitos aspectos das organizações. Muito está ainda por fazer neste contexto, quer na caracterização formal de outros conceitos fundamentais, quer no enriquecimento das especificações de organizações com outros aspectos relevantes dos seus processos e estrutura, de modo a obter uma abordagem útil para especificar e analisar organizações concretas.

O trabalho apresentado nesta dissertação foi também orientado pela convicção de que a utilização de técnicas da lógica formal - e especialmente das ferramentas providenciadas pela Lógica Modal - podem dar uma valiosa contribuição no âmbito da especificação e análise de organizações. Apesar deste trabalho se encontrar ainda num estágio inicial relativamente ao domínio de aplicação adoptado, os resultados alcançados ilustram claramente o interesse desta orientação.

Anexos

Anexo 1: Implementação do sistema dedutivo $TLact_N$

```
/*
        ================================
        LactN Tableaux Program

        Propositional operators: neg, and, or, imp, eqv.
        Modal operators: e, g, i.
        ================================
*/

:- op(140, fy, neg).
:- op(160, xfy, [and, or, imp, eqv]).

/*
        conjunctive(X) :- X is an alpha formula.
*/

conjunctive( ( _ and _ ) ).
conjunctive( ( _ eqv _ ) ).
conjunctive( neg( _ or _ ) ).
conjunctive( neg( _ imp _ ) ).

/*
        disjunctive(X) :- X is a betha formula.
*/

disjunctive( ( _ or _ )).
disjunctive( ( _ imp _ ) ).
disjunctive( neg( _ and _ ) ).
disjunctive( neg( _ eqv _ ) ).

/*
        unary(X) :- X is a double negation or a negated constant.
*/

unary(neg neg _ ).
unary(neg true).
unary(neg false).

/*
        components(X, Y, Z) :- Y and Z are the components of formula X.
*/

components(X and Y, X, Y).
```

```
components(neg(X and Y), neg X, neg Y).
components(X or Y, X, Y).
components(neg(X or Y), neg X, neg Y).
components(X imp Y, neg X, Y).
components(neg(X imp Y),  X, neg Y).
components(X eqv Y, X imp Y, Y imp X).
components(neg(X eqv Y), neg(X imp Y), neg(Y imp X)).

/*
        component(X, Y) :- Y is the component of the unary formula X.
*/

component(neg neg X, X).
component(neg true, false).
component(neg false, true).

/*
        theorem(X) :-  create a complete tableaux expansion for neg X and test if it is closed.
*/

theorem(X) :- expansion(neg X, Y,complete), closed(Y).

/*
         expansion(X, Y, T) :-  Y is the result of the expansion (of type T) of X.

                T = complete (complete dual clause form expansion);
                T = reduced (propositional dual clause form expansion);
                T = clausepl (propositional clause form expansion).
*/

expansion(X, Y, clausepl) :- !, expand0([[X]], Y).
expansion(X, Y, Type) :- Type \= clausepl, expand1([[X]], Y, Type).

/*
        expand0(Old, New) :- New is the result of applying singlestep0 as many times as possible,
                                starting with Old.
*/

expand0(Con, Newcon) :- singlestep0(Con, Temp), !, expand0(Temp, Newcon).
expand0(Clause, Clause).

/*
        singlestep0(Old, New) :- New is the result of applying a singlestep0 of the expansion
                                process to Old.
*/

singlestep0([Dijunction | Rest], New) :-                /* rules: Rnegneg, RnegTrue, RnegFalse */
        memberx(Formula, Dijunction),
        unary(Formula),
        component(Formula, Newformula),
        remove(Formula, Dijunction, Temporary),
        Newdijunction = [Newformula | Temporary],
        New = [Newdijunction | Rest].
```

```
singlestep0([Dijunction | Rest], New) :-                /* rule: Ralpha */
        memberx(Alpha, Dijunction),
        conjunctive(Alpha),
        components(Alpha, Alphaone, Alphatwo),
        remove(Alpha, Dijunction, Temporary),
        Newdisone = [Alphaone | Temporary],
        Newdistwo = [Alphatwo | Temporary],
        New = [Newdisone, Newdistwo | Rest].

singlestep0([Dijunction | Rest], New) :-                /* rule: Rbeta */
        memberx(Beta, Dijunction),
        disjunctive(Beta),
        components(Beta, Betaone, Betatwo),
        remove(Beta, Dijunction, Temporary),
        Newdijunction = [Betaone, Betatwo | Temporary],
        New = [Newdijunction | Rest].

singlestep0([Disjunction | Rest], [Disjunction | Newrest]) :- singlestep0(Rest, Newrest).

/*
        expand1(Old, New, T) :- New is the result of applying singlestep1 (of type T) as many
                                times as possible, starting with Old.
*/

expand1(Dis, Newdis, T) :- singlestep1(Dis, Temp, T), !, expand1(Temp, Newdis, T).
expand1(Dualclause, Dualclause, T).

/*
        singlestep1(Old, New, T) :- New is the result of applying a singlestep1 (of type T) of the
                                expansion process to Old.
*/

singlestep1([Conjunction | Rest], New, T) :-            /* rules: Rnegneg, RnegTrue, RnegFalse */
        memberx(Formula, Conjunction),
        unary(Formula),
        component(Formula, Newformula),
        remove(Formula, Conjunction, Temporary),
        Newconjunction = [Newformula | Temporary],
        New = [Newconjunction | Rest].

singlestep1([Conjunction | Rest], New, T) :-            /* rule: Ralpha */
        memberx(Alpha, Conjunction),
        conjunctive(Alpha),
        components(Alpha, Alphaone, Alphatwo),
        remove(Alpha, Conjunction, Temporary),
        Newconjunction = [Alphaone, Alphatwo | Temporary],
        New = [Newconjunction | Rest].

singlestep1([Conjunction | Rest], New, T) :-            /* rule: Rbeta */
        memberx(Beta, Conjunction),
        disjunctive(Beta),
        components(Beta, Betaone, Betatwo),
        remove(Beta, Conjunction, Temporary),
        Newconone = [Betaone | Temporary],
```

```
        Newcontwo = [Betatwo | Temporary],
        New = [Newconone, Newcontwo | Rest].

singlestep1([Conjunction | Rest], New, T) :-            /* rule: REiimpGi */
        memberx(e(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(e(Index, Argformula), Conjunction, Temporary),
        if_then_else(   T = complete,
                Newcon = [op_e(Index, Argexpansion), g(Index, Argformula) | Temporary],
                Newcon = [op_e(Index, Argexpansion) | Temporary]),
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-            /* rule: RTGi */
        memberx(g(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(g(Index, Argformula), Conjunction, Temporary),
        if_then_else(   T = complete,
                        Newcon = [op_g(Index, Argexpansion), Argformula | Temporary],
                        Newcon = [op_g(Index, Argexpansion) | Temporary]),
        New = [Newcon | Rest].

/* mark remaining operators */

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg e(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg e(Index, Argformula), Conjunction, Temporary),
        Newcon = [neg op_e(Index, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg g(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg g(Index, Argformula), Conjunction, Temporary),
        Newcon = [neg op_g(Index, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(i(I, J, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(i(I, J, Argformula), Conjunction, Temporary),
        Newcon = [op_i(I, J, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg i(I, J, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg i(I, J, Argformula), Conjunction, Temporary),
        Newcon = [neg op_i(I, J, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], [Conjunction | Newrest], T) :- singlestep1(Rest, Newrest, T).
```

```
/*
        closed(Tableau) :- every branch of Tableau contains a contradiction.
*/

closed([Branch | Rest]) :-
        memberx(false, Branch), !,
        closed(Rest).

closed([Branch | Rest]) :-
        memberx(X, Branch),
        memberx(neg X, Branch), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rule: RrEEi */
        memberx(op_e(I, X), Branch),
        memberx(neg op_e(I, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rule: RrEGi */
        memberx(op_g(I, X), Branch),
        memberx(neg op_g(I, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rule: RrEiIj */
        memberx(op_i(I, J, X), Branch),
        memberx(neg op_i(I, J, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rules: RNoGi, RNoiIj */
        (memberx(op_g(I, X), Branch); memberx(op_i(I, J, X), Branch)),
        convert_dual(X, Convx),
        theorem(Convx), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rule: RNoFiIj */
        memberx(op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        theorem(neg Convx), !,
        closed(Rest).

closed([Branch | Rest]) :-                              /* rule: RDiIj */
        memberx(op_i(I, J, X), Branch),
        remove(op_i(I, J, X), Branch, Branch2),
        memberx(op_i(I, J, Y), Branch2),
        convert_dual(X, Convx),
```

```
        convert_dual(Y, Convy),
        theorem(Convx eqv neg Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                          /* rule: RDCiIj */
        memberx(op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        expansion(neg Convx, NegXexpansion, clausepl),
        NegXexpansion = [A, B | L],
        set_partitions(NegXexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_i(I, J, Y), Branch), X \= Y,
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                          /* rule: RC1Ei */
        memberx(neg op_e(I, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_e(I, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_e(I, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_e(I, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_e(I, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_e(I, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                          /* rule: RC1Gi */
        memberx(neg op_g(I, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
```

```
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_g(I, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_g(I, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_g(I, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_g(I, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_g(I, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                              /* rule: RC1iIj */
        memberx(neg op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_i(I, J, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                              /* rules: RiIj&GjimpGi, RC2Gi */
        memberx(neg op_g(I, X), Branch),
        remove(neg op_g(I, X), Branch, Newbranch1),
        (       ( memberx( op_i(I, J, Y), Newbranch1),
                  convert_dual(X, Convx),
                  convert_dual(Y, Convy),
                  theorem(Convx eqv Convy),
                  remove(op_i(I, J, Y), Newbranch1, Newbranch2),
                  closed([[neg op_g(J, X) | Newbranch2]])
                );
                ( convert_dual(X, Convx),
                  expansion(Convx, Xexpansion, clausepl),
```

```
                Xexpansion = [A, B | L],
                set_partitions(Xexpansion, Set_partitions),
                memberx([X1, Y1], Set_partitions),
                convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
                expansion(Convx1, X1expansion, reduced),
                closed([[neg op_g(I, X1expansion) | Newbranch1]]),
                expansion(Convy1, Y1expansion, reduced),
                closed([[neg op_g(I, Y1expansion) | Newbranch1]]))
        ), !, closed(Rest).

closed([Branch | Rest]) :-                      /* rules: REiEjimpEiGj, RnoQEiEj */
        memberx(op_e(I, X), Branch),            /* rules: REiGjimpiIj, REredEi */
        convert_dual(X, Convx1),
        expansion(Convx1, Xexpansion1, clausepl),
        simplify(Xexpansion1, Simpler_Xexpansion1),
        memberx(Dis, Simpler_Xexpansion1),
        (       ( memberx(e(J, Y), Dis),
                  convert_clause(Simpler_Xexpansion1, Convx2),
                  theorem(Convx2 eqv e(J, Y)),
                  remove(op_e(I, X), Branch, Newbranch),
                  expansion(Y, Yexpansion, reduced),
                  ( closed([[op_e(I,[[op_g(J, Yexpansion)]]) | Newbranch]]);
                    ( I \= J, closed([[ neg op_e(I,Yexpansion) | Newbranch]]))), !
                );
                ( memberx(g(J, Y), Dis),
                  convert_clause(Simpler_Xexpansion1, Convx2),
                  theorem(Convx2 eqv g(J, Y)),
                  remove(op_e(I, X), Branch, Newbranch),
                  expansion(Y, Yexpansion, reduced),
                  closed([[op_i(I, J, Yexpansion) | Newbranch]]), !
                )
        ), closed(Rest).

closed([Branch | Rest]) :-                      /* rules: RQGiGj, RGiEjimpGiGj */
        memberx(op_g(I, X), Branch),            /* rules: REredGi */
        convert_dual(X, Convx1),
        expansion(Convx1, Xexpansion1, clausepl),
        simplify(Xexpansion1, Simpler_Xexpansion1),
        memberx(Dis, Simpler_Xexpansion1),
         (      ( memberx(e(J, Y), Dis),
                  convert_clause(Simpler_Xexpansion1, Convx2),
                  theorem(Convx2 eqv e(J, Y)),
                  remove(op_g(I, X), Branch, Newbranch),
                  expansion(Y, Yexpansion, reduced),
                  closed([[op_g(I,[[op_g(J, Yexpansion)]]) | Newbranch]]), !
                );
                ( memberx(g(J, Y), Dis),
                  convert_clause(Simpler_Xexpansion1, Convx2),
                  theorem(Convx2 eqv g(J, Y)),
                  remove(op_g(I, X), Branch, Newbranch),
                  expansion(Y, Yexpansion, reduced),
                  closed([[op_g(I,Yexpansion) | Newbranch]]), !
                )
        ), closed(Rest).
```

```
closed([]).

/*
        convert_dual(X, Y):-  Y is the Lact formula that corresponds to X (dual clause form
                              notation).
*/

convert_dual([], false).
convert_dual([X], Y) :- conv_and(X,Y).
convert_dual([X,Y | L], W or Z) :- conv_and(X,W), convert_dual([Y | L], Z).

conv_and([],true).
conv_and([neg X1], neg X2) :- !, conv_and([X1], X2).
conv_and([op_e(I, L1)], e(I, L2)) :- !, convert_dual(L1, L2).
conv_and([op_g(I, L1)], g(I, L2)) :- !, convert_dual(L1, L2).
conv_and([op_i(I1, I2, L1)], i(I1, I2, L2)) :- !, convert_dual(L1, L2).
conv_and([X], X).
conv_and([X1,Y | Z], X2 and W) :- conv_and([X1], X2), conv_and([Y | Z],W).

/*
        convert_clause(X, Y):- Y is the Lact formula that corresponds to X (clause form notation).
*/

convert_clause([], true).
convert_clause([X], Y) :- conv_or(X,Y).
convert_clause([X,Y | L], W and Z) :- conv_or(X,W), convert_clause([Y | L], Z).

conv_or([],false).
conv_or([neg X1], neg X2) :- !, conv_or([X1], X2).
conv_or([op_e(I, L1)], e(I, L2)) :- !, convert_clause(L1, L2).
conv_or([op_g(I, L1)], g(I, L2)) :- !, convert_clause(L1, L2).
conv_or([op_i(I1, I2, L1)], i(I1, I2, L2)) :- !, convert_clause(L1, L2).
conv_or([X], X).
conv_or([X1,Y | Z], X2 or W) :- conv_or([X1], X2), conv_or([Y | Z],W).

/*
    simplify(X, Y) :- X is simplified to Y, by removing "false" in each dijunction on X, and
                          eliminating on X every dijunction with "B or neg B" or true.
*/

simplify(Con, Newcon) :- simplify_step(Con, Temp), !, simplify(Temp, Newcon).
simplify(Con, Con).

simplify_step([Dis | Rest], Newcon) :-
        memberx(false, Dis), remove(false, Dis, Newdis), Newcon = [Newdis | Rest].
simplify_step([Dis | Rest], Newcon) :-
        memberx(true, Dis), Newcon = Rest.
simplify_step([Dis | Rest], Newcon) :-
        memberx(X, Dis), memberx(neg X, Dis), Newcon = Rest.
simplify_step([Dis | Rest], [Dis | Newrest]) :- simplify_step(Rest, Newrest).
```

```
/*
        ================================
        Program auxiliary predicates
        ================================
*/

memberx(X, [X | _]).
memberx(X, [_ | T]):- memberx(X, T).

remove(X, [], []).
remove(X, [X | T1], T2) :- remove(X, T1, T2).
remove(X, [Y | T1], [Y | T2]) :- X \= Y, remove(X, T1, T2).

union([], Y, Y).
union([X|L1], Y,L2) :- memberx(X, L2), remove(X, L2, LT), union(L1, Y, LT).

partition(X, Y, Z) :- union(X,Y,Z), X \= [], Y \= [].

set_partitions(S,P) :- setof([X, Y], partition(X,Y,S), Temp), bag_to_set_partitions(Temp, P).

bag_to_set_partitions([[X1, X2]], [[X1, X2]]).
bag_to_set_partitions([[X1, X2], Z | L], S) :-
        memberx([Y1, Y2], [Z | L]),
        ((equal_set(X1, Y1), equal_set(X2, Y2)); (equal_set(X1, Y2), equal_set(X2, Y1))), !,
        bag_to_set_partitions([ Z | L], S).
bag_to_set_partitions([[X1, X2], Z | L], [[X1, X2] | S]) :- bag_to_set_partitions([ Z | L], S).

equal_set([], []).
equal_set([X|A], B) :-
        memberx(X, B), remove(X, A, A1), remove(X, B, B1), equal_set(A1, B1).

if_then_else(P, Q, R) :- P, !, Q.
if_then_else(P, Q, R) :- R.
```

Anexo 2: Código da Bancada

```
/*
        ================================
        Organization bed
        ================================
*/

:- op(140, fy, neg).
:- op(160, xfy, [and, or, imp, eqv]).

/*
        Main menu.
*/

main :- reconsult(organizations),
        repeat, nl, write('main_menu: new, open, list, del, help, exit ? -> '),
        read(Option), nl, main_com(Option).

main_com(new) :-
        write('  new org_id -> '), read(Org_id), nl,
        not(org(Org_id,Org_desc)),
        assert(org(Org_id, [])), assert(behaviour(Org_id, true)),
        assert(behaviour_list(Org_id, yes)),
        org_menu(Org_id), !, fail.
main_com(new) :-
        !, write('  invalid org_id: there is another organization with that name.'), nl, fail.

main_com(open) :-
        write('  predefined org_id -> '), read(Org_id), nl,
        org(Org_id, Org_desc), org_menu(Org_id), !, fail.
main_com(open) :-
        !, write('  invalid org_id: there is no predefined organization with that name.'), nl, fail.

main_com(list) :-
        !, write('  list of predefined organizations:'), nl,
        org(Org_id, Org_desc), nl,
        write('    Org_id            : '), write(Org_id), nl,
        write('    Org_desc: '), write(Org_desc), nl, fail.

main_com(del) :-
        write('  organization to be deleted: org_id -> '), read(Org_id), nl,
        retract(org(Org_id,Org_desc)), !,
        retract(behaviour_list(Org_id,YN)),
        behaviour(Org_id, Behaviour),
        retract(behaviour(Org_id, Behaviour)), fail.
main_com(del) :-
        !, write('  invalid org_id: there is no predefined organization with that name.'), nl, fail.
```

```
main_com(help) :- !,
        write('  new : define new organization;'), nl,
        write('  open: open predefined organization;'), nl,
        write('  list   : list of predefined organizations;'), nl,
        write('  del   : delete predefined organization;'), nl,
        write('  exit  : exit program and save defined organizations.'), nl, fail.

main_com(exit) :- !, tell(organizations), save_organizations.

main_com( _ ) :- write('  invalid command!'), nl, !, fail.

save_organizations :-
        org(Org_id, Org_desc), write(org(Org_id, Org_desc)), write('.'), nl,
        retract(org(Org_id, Org_desc)), fail.
save_organizations :-
        nl, nl,
        behaviour_list(Org_id, YN ),
        write(behaviour_list(Org_id, YN )), write('.'), nl,
        retract(behaviour_list(Org_id, YN )),
        behaviour(Org_id, Behaviour ),
        write(behaviour(Org_id, Behaviour )), write('.'), nl,
        retract(behaviour(Org_id, Behaviour )), fail.
save_organizations :- told.

/*
        Org menu.
*/

org_menu(Org_id) :-
        repeat, nl, write(Org_id), write(' : desc, modify, analyse, help, exit ? -> '),
        read(Option), nl, org_com(Org_id, Option).

org_com(Org_id, desc) :-
        !, write('  Org_desc:'), nl,
        org(Org_id, Org_desc), write('     '), write(Org_desc), nl, fail.

org_com(Org_id, modify) :- modify_menu(Org_id), !, fail.

org_com(Org_id, analyse) :- analyse_menu(Org_id), !, fail.

org_com( _ , help) :- !,
        write('  desc      : current organization description;'), nl,
        write('  modify : modify current organization;'), nl,
        write('  analyse : analyse current organization;'), nl,
        write('  exit       : return main menu.'), nl, fail.

org_com( _, exit) :- !.

org_com( _ , _ ) :- write('  invalid command!'), nl, !, fail.

/*
        Modify menu.
*/
```

```
modify_menu(Org_id) :-
        repeat, nl, write('modify('), write(Org_id), write(') : add_item, del_item, exit ? -> '),
        read(Option), nl, modify_com(Org_id, Option).

modify_com(Org_id, add_item) :-
        !, write('  item to be added -> '), read(Formula), nl, check_item(Formula),
        retract(org(Org_id, Org_desc)), assert(org(Org_id, [Formula | Org_desc])), fail.

modify_com(Org_id, del_item) :-
        !, write('  item to be deleted -> '), read(Formula), nl, check_item(Formula),
        retract(org(Org_id, Org_desc)), remove(Formula, Org_desc, Neworg_desc),
        assert(org(Org_id, Neworg_desc)), fail.

modify_com(Org_id, exit) :-
        !, retract(behaviour_list(Org_id, _ )), assert(behaviour_list(Org_id, no)),
        delete_behaviours(Org_id).

modify_com( _ , _ ) :- write('  invalid command!'), nl, !, fail.

/*
        Analyse menu.
*/

analyse_menu(Org_id) :-
        repeat, nl, write('analyse('), write(Org_id),
        write(') : behaviours, can, whocan, how, help, exit ? -> '),
        read(Option), nl, analyse_com(Org_id, Option).

analyse_com(Org_id, behaviours) :-
        write('  list of possible behaviours:'), nl,
        generate_behaviours(Org_id),
        behaviour(Org_id, Behaviour),
        write('     '), write(Behaviour), nl, fail.
analyse_com( _, behaviours) :- !, fail.

analyse_com(Org_id, can) :-
        write('  formula -> '),  read(Formula), nl, check(Formula),
        generate_behaviours(Org_id),
        minimal_behaviours(Org_id, Formula, Minimal_behaviours),
        Minimal_behaviours = [X | Tail], !,
        write('    YES! Consider, e.g. the possible behaviour: '), nl,
        write('     '), write(X), nl, fail.
analyse_com(Org_id, can) :-
        write('    NO! There is no possible behaviour that satisfies this formula.'), nl, !, fail.

analyse_com(Org_id, whocan) :-
        !, write('  formula -> '),  read(Formula), nl, checkwho(Formula),
        write('  frontier -> '),  read(Frontier), nl, checkfrontier(Frontier),
        generate_behaviours(Org_id),
        agents_who_can(Org_id, Formula, Frontier, Agents),
        write('     who = '), write(Agents), nl, fail.
```

```
analyse_com(Org_id, how) :-
        write('  formula -> '),  read(Formula), nl, checkhow(Formula),
        generate_behaviours(Org_id),
        minimal_behaviours(Org_id, Formula, Minimal_behaviours),
        Minimal_behaviours = [X | Tail], !,
        arg(1, Formula, Agent),
        agent_minimal_acts(Agent, Minimal_behaviours, Mandatory_acts, Optional_acts),
        write('    Agent '), write(Agent), write(' must do:'), nl,
        write('       mandatory acts: '),
        if_then_else(   Mandatory_acts = true,
                        write('No mandatory acts!'),
                        write(Mandatory_acts) ), nl,
        write('       optional acts: '),
        if_then_else(   Optional_acts = true,
                        write('No optional acts!'),
                        write(Optional_acts) ), nl, fail.
analyse_com(Org_id, how) :-
        write('    NO! There is no possible behaviour.'), nl, !, fail.

analyse_com( _ , help) :- !,
        write('  behaviours : list of possible behaviours of the current organization;'), nl,
        write('  can            : analyse possibility in the current organization;'), nl,
        write('  whocan      : analyse who can do or ensure in the current organization;'), nl,
        write('  how          : analyse what must an agent do to ensure in the current organization;'),
        nl, write('  exit            : return previous menu.'), nl, fail.

analyse_com( _ , exit) :- !.

analyse_com( _ , _ ) :- write('  invalid command!'), nl, fail.

/*
        consistency(Actlist) :- consistency teste for acts whith PL formulas.
*/

consistency(Actlist) :- condition1(Actlist), condition2(Actlist), condition3(Actlist).

condition1(Actlist) :-
        findall(Arg, memberx(cap(I, Arg), Actlist), Arglist),
        actlist_to_behaviour(Arglist, Convarglist),
        not(theorem(neg Convarglist)).

condition2(Actlist) :-
        findall((I,J), memberx(c(I, J, _), Actlist), Pairlist),
        test_each_pair(Pairlist, Actlist).

test_each_pair([(I,J) | L], Actlist) :-
        findall(Arg, memberx(c(I, J, Arg), Actlist), Arglist),
        actlist_to_behaviour(Arglist, Convarglist), !,
        not(theorem(neg Convarglist)),
        test_each_pair(L, Actlist).
test_each_pair([], _).

condition3([A | L]) :-
        item_arg(A, Arg), !, not(theorem(Arg)), condition3(L).
```

```
condition3([]).

/*
        Auxiliary predicates.
*/

minimal_behaviours(Org_id, Formula, Minimal_behaviours) :-
        behaviours_to_analise(Org_id),
        reject_not_minimal(Org_id, Formula),
        setof(Behaviour, temp_behaviour(Org_id, Behaviour), Minimal_behaviours), !,
        delete_temp_behaviours(Org_id).
minimal_behaviours(Org_id, Formula, []).

reject_not_minimal(Org_id, Formula) :-
        temp_behaviour(Org_id, Behaviour),
        not(theorem(Behaviour imp Formula)),
        retract(temp_behaviour(Org_id, Behaviour)), fail.
reject_not_minimal(Org_id, Formula) :-
        temp_behaviour(Org_id, Behaviour1),
        temp_behaviour(Org_id, Behaviour2),
        Behaviour1 \= Behaviour2,
        theorem(Behaviour1 imp Behaviour2),
        retract(temp_behaviour(Org_id, Behaviour1)), fail.
reject_not_minimal( _ , _ ).

behaviours_to_analise(Org_id) :-
        behaviour(Org_id, Behaviour),
        assert(temp_behaviour(Org_id, Behaviour)), fail.
behaviours_to_analise( _ ).

delete_temp_behaviours(Org_id) :-
        temp_behaviour(Org_id, Behaviour),
        retract(temp_behaviour(Org_id, Behaviour)), fail.
delete_temp_behaviours( _ ).

generate_behaviours(Org_id) :- behaviour_list(Org_id, yes), !.
generate_behaviours(Org_id) :-
        retract(behaviour_list(Org_id, _ )),
        delete_behaviours(Org_id), assert(behaviour(Org_id, true)),
        org(Org_id, Org_desc), Org_desc \= [],
        insert_if_possible_behaviour(Org_id, Org_desc),
        set_partitions(Org_desc, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        insert_if_possible_behaviour(Org_id, X1),
        insert_if_possible_behaviour(Org_id, Y1),
        fail.
generate_behaviours(Org_id) :- assert(behaviour_list(Org_id, yes)).

delete_behaviours(Org_id) :-
        behaviour(Org_id, Behaviour),
        retract(behaviour(Org_id, Behaviour)),
        fail.
delete_behaviours( _ ).
```

```
insert_if_possible_behaviour(Org_id, Actlist) :-
        consistency(Actlist),
        actlist_to_behaviour(Actlist, Behaviour),
        assert(behaviour(Org_id, Behaviour)), !.
        insert_if_possible_behaviour(Org_id, Actlist).

actlist_to_behaviour([], true).
actlist_to_behaviour([X1], X2) :- act_to_behaviour(X1, X2).
actlist_to_behaviour([X,Y | L], W and Z) :-
        act_to_behaviour(X, W), actlist_to_behaviour([Y | L], Z).

act_to_behaviour(cap(I, Argformula), e(I, Argformula)) :- !.
act_to_behaviour(c(I, J, Argformula), i(I, J, Argformula)) :- !.
act_to_behaviour(X, X).

agent_minimal_acts(Agent, Minimal_behaviours, Mandatory_acts, Optional_acts) :-
        direct_acts(Agent, Minimal_behaviours, Agentbehaviours),
        list_intersection(Agentbehaviours, Mandatory_acts_aux),
        convert_dual([Mandatory_acts_aux], Mandatory_acts),
        list_without_set(Agentbehaviours, Mandatory_acts_aux, Optional_acts_aux),
        convert_dual(Optional_acts_aux, Optional_acts).

direct_acts(Agent, [], []).
direct_acts(Agent, [B | L1], [Bexpansion2 | L2]) :-
        expansion(B, [Bexpansion1], reduced),
        remove_other_acts(Agent, Bexpansion1, Bexpansion2),
        direct_acts(Agent, L1, L2).

remove_other_acts(Agent, [], []).
remove_other_acts(Agent, [X | T1], [X | T2]) :-
        arg(1, X, Agent), !, remove_other_acts(Agent, T1, T2).
remove_other_acts(Agent, [X | T1], T2) :-
        remove_other_acts(Agent, T1, T2).

agents_who_can( _ , _ , [], []) :- !.
agents_who_can(Org_id, Formula, [A | Tailfrontier], [A | Agentstailfrontier]) :-
        subs(who, A, Formula, Subsformula),
        behaviour(Org_id, Behaviour), theorem(Behaviour imp Subsformula), !,
        agents_who_can(Org_id, Formula, Tailfrontier, Agentstailfrontier).
agents_who_can(Org_id, Formula, [A | Tailfrontier], Agentstailfrontier) :-
        agents_who_can(Org_id, Formula, Tailfrontier, Agentstailfrontier).

check_item(cap(_, Argformula)) :- !, check(Argformula).
check_item(c(_, _, Argformula)) :- !, check(Argformula).
check_item( _ ) :- write('  invalid item!'), nl, fail.

check(Formula).                                    /* no further checks available */

checkwho(g(who, Argformula)) :- !, check(Formula).
checkwho(e(who, Argformula)) :- !, check(Formula).
checkwho( _ ) :- write('  invalid formula!'), nl, fail.

checkhow(g(Ag, Argformula)) :- !, check(Formula).
checkhow( _ ) :- write('  invalid formula!'), nl, fail.
```

```
checkfrontier(Frontier) :- Frontier = [A | L], !.
checkfrontier(Frontier) :- write('  invalid frontier!'), nl, fail.

subs(who, A, g(who, Argformula), g(A, Argformula)).
subs(who, A, e(who, Argformula), e(A, Argformula)).

item_arg(cap(I, Argformula), Argformula).
item_arg(c(I, J, Argformula), Argformula).

behaviour( _ , _ ) :- fail.

/*
        ================================
        LactN Tableaux Program

        Propositional operators: neg, and, or, imp, eqv.
        Modal operators: e, g, i.
        ================================
*/

/*
        conjunctive(X) :- X is an alpha formula.
*/

conjunctive( ( _ and _ ) ).
conjunctive( ( _ eqv _ ) ).
conjunctive( neg( _ or _ ) ).
conjunctive( neg( _ imp _ ) ).

/*
        disjunctive(X) :- X is a betha formula.
*/

disjunctive( ( _ or _ )).
disjunctive( ( _ imp _ ) ).
disjunctive( neg( _ and _ ) ).
disjunctive( neg( _ eqv _ ) ).

/*
        unary(X) :- X is a double negation or a negated constant.
*/

unary(neg neg _ ).
unary(neg true).
unary(neg false).

/*
        components(X, Y, Z) :- Y and Z are the components of formula X.
*/

components(X and Y, X, Y).
components(neg(X and Y), neg X, neg Y).
components(X or Y, X, Y).
```

```
components(neg(X or Y), neg X, neg Y).
components(X imp Y, neg X, Y).
components(neg(X imp Y),  X, neg Y).
components(X eqv Y, X imp Y, Y imp X).
components(neg(X eqv Y), neg(X imp Y), neg(Y imp X)).

/*
        component(X, Y) :- Y is the component of the unary formula X.
*/

component(neg neg X, X).
component(neg true, false).
component(neg false, true).

/*
        theorem(X) :-  create a complete tableaux expansion for neg X and test if it is closed.
*/

theorem(X) :- expansion(neg X, Y,complete), closed(Y).

/*
         expansion(X, Y, T) :-  Y is the result of the expansion (of type T) of X.

                T = complete (complete dual clause form expansion);
                T = reduced (propositional dual clause form expansion);
                T = clausepl (propositional clause form expansion).
*/

expansion(X, Y, clausepl) :- !, expand0([[X]], Y).
expansion(X, Y, Type) :- Type \= clausepl, expand1([[X]], Y, Type).

/*
        expand0(Old, New) :- New is the result of applying singlestep0 as many times as possible,
                                starting with Old.
*/

expand0(Con, Newcon) :- singlestep0(Con, Temp), !, expand0(Temp, Newcon).
expand0(Clause, Clause).

/*
        singlestep0(Old, New) :- New is the result of applying a singlestep0 of the expansion
                                process to Old.
*/

singlestep0([Dijunction | Rest], New) :-                /* rules: Rnegneg, RnegTrue, RnegFalse */
        memberx(Formula, Dijunction),
        unary(Formula),
        component(Formula, Newformula),
        remove(Formula, Dijunction, Temporary),
        Newdijunction = [Newformula | Temporary],
        New = [Newdijunction | Rest].

singlestep0([Dijunction | Rest], New) :-                /* rule: Ralpha */
        memberx(Alpha, Dijunction),
```

```
        conjunctive(Alpha),
        components(Alpha, Alphaone, Alphatwo),
        remove(Alpha, Dijunction, Temporary),
        Newdisone = [Alphaone | Temporary],
        Newdistwo = [Alphatwo | Temporary],
        New = [Newdisone, Newdistwo | Rest].

singlestep0([Dijunction | Rest], New) :-            /* rule: Rbeta */
        memberx(Beta, Dijunction),
        disjunctive(Beta),
        components(Beta, Betaone, Betatwo),
        remove(Beta, Dijunction, Temporary),
        Newdijunction = [Betaone, Betatwo | Temporary],
        New = [Newdijunction | Rest].

singlestep0([Disjunction | Rest], [Disjunction | Newrest]) :- singlestep0(Rest, Newrest).

/*
        expand1(Old, New, T) :- New is the result of applying singlestep1 (of type T) as many
                                times as possible, starting with Old.
*/

expand1(Dis, Newdis, T) :- singlestep1(Dis, Temp, T), !, expand1(Temp, Newdis, T).
expand1(Dualclause, Dualclause, T).

/*
        singlestep1(Old, New, T) :- New is the result of applying a singlestep1 (of type T) of the
                                expansion process to Old.
*/

singlestep1([Conjunction | Rest], New, T) :-        /* rules: Rnegneg, RnegTrue, RnegFalse */
        memberx(Formula, Conjunction),
        unary(Formula),
        component(Formula, Newformula),
        remove(Formula, Conjunction, Temporary),
        Newconjunction = [Newformula | Temporary],
        New = [Newconjunction | Rest].

singlestep1([Conjunction | Rest], New, T) :-        /* rule: Ralpha */
        memberx(Alpha, Conjunction),
        conjunctive(Alpha),
        components(Alpha, Alphaone, Alphatwo),
        remove(Alpha, Conjunction, Temporary),
        Newconjunction = [Alphaone, Alphatwo | Temporary],
        New = [Newconjunction | Rest].

singlestep1([Conjunction | Rest], New, T) :-        /* rule: Rbeta */
        memberx(Beta, Conjunction),
        disjunctive(Beta),
        components(Beta, Betaone, Betatwo),
        remove(Beta, Conjunction, Temporary),
        Newconone = [Betaone | Temporary],
        Newcontwo = [Betatwo | Temporary],
        New = [Newconone, Newcontwo | Rest].
```

```
singlestep1([Conjunction | Rest], New, T) :-            /* rule: REiimpGi */
        memberx(e(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(e(Index, Argformula), Conjunction, Temporary),
        if_then_else(   T = complete,
                Newcon = [op_e(Index, Argexpansion), g(Index, Argformula) | Temporary],
                Newcon = [op_e(Index, Argexpansion) | Temporary]),
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-            /* rule: RTGi */
        memberx(g(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(g(Index, Argformula), Conjunction, Temporary),
        if_then_else(   T = complete,
                        Newcon = [op_g(Index, Argexpansion), Argformula | Temporary],
                        Newcon = [op_g(Index, Argexpansion) | Temporary]),
        New = [Newcon | Rest].

/* mark remaining operators */

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg e(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg e(Index, Argformula), Conjunction, Temporary),
        Newcon = [neg op_e(Index, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg g(Index, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg g(Index, Argformula), Conjunction, Temporary),
        Newcon = [neg op_g(Index, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(i(I, J, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(i(I, J, Argformula), Conjunction, Temporary),
        Newcon = [op_i(I, J, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], New, T) :-
        memberx(neg i(I, J, Argformula), Conjunction),
        expansion(Argformula, Argexpansion, reduced),
        remove(neg i(I, J, Argformula), Conjunction, Temporary),
        Newcon = [neg op_i(I, J, Argexpansion) | Temporary],
        New = [Newcon | Rest].

singlestep1([Conjunction | Rest], [Conjunction | Newrest], T) :- singlestep1(Rest, Newrest, T).

/*
        closed(Tableau) :- every branch of Tableau contains a contradiction.
*/
```

```
closed([Branch | Rest]) :-
        memberx(false, Branch), !,
        closed(Rest).

closed([Branch | Rest]) :-
        memberx(X, Branch),
        memberx(neg X, Branch), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rule: RrEEi */
        memberx(op_e(I, X), Branch),
        memberx(neg op_e(I, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rule: RrEGi */
        memberx(op_g(I, X), Branch),
        memberx(neg op_g(I, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rule: RrEiIj */
        memberx(op_i(I, J, X), Branch),
        memberx(neg op_i(I, J, Y), Branch),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv Convy), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rules: RNoGi, RNoiIj */
        (memberx(op_g(I, X), Branch); memberx(op_i(I, J, X), Branch)),
        convert_dual(X, Convx),
        theorem(Convx), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rule: RNoFiIj */
        memberx(op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        theorem(neg Convx), !,
        closed(Rest).

closed([Branch | Rest]) :-                      /* rule: RDiIj */
        memberx(op_i(I, J, X), Branch),
        remove(op_i(I, J, X), Branch, Branch2),
        memberx(op_i(I, J, Y), Branch2),
        convert_dual(X, Convx),
        convert_dual(Y, Convy),
        theorem(Convx eqv neg Convy), !,
        closed(Rest).
```

```
closed([Branch | Rest]) :-                          /* rule: RDCiIj */
        memberx(op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        expansion(neg Convx, NegXexpansion, clausepl),
        NegXexpansion = [A, B | L],
        set_partitions(NegXexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_i(I, J, Y), Branch), X \= Y,
        convert_dual(Y, Convy),
        if_then_else(    theorem(Convy eqv Convx1),
                        ( remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                          /* rule: RC1Ei */
        memberx(neg op_e(I, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_e(I, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(    theorem(Convy eqv Convx1),
                        ( remove(neg op_e(I, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_e(I, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_e(I, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_e(I, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                          /* rule: RC1Gi */
        memberx(neg op_g(I, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
```

```
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_g(I, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_g(I, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_g(I, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_g(I, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_g(I, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                              /* rule: RC1iIj */
        memberx(neg op_i(I, J, X), Branch),
        convert_dual(X, Convx),
        expansion(Convx, Xexpansion, clausepl),
        Xexpansion = [A, B | L],
        set_partitions(Xexpansion, Set_partitions),
        memberx([X1, Y1], Set_partitions),
        convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
        memberx(op_i(I, J, Y), Branch),
        convert_dual(Y, Convy),
        if_then_else(   theorem(Convy eqv Convx1),
                        ( remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convy1, Y1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, Y1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest)),
                        ( theorem(Convy eqv Convy1),
                          remove(neg op_i(I, J, X), Branch, Newbranch1),
                          expansion(Convx1, X1expansion, reduced),
                          Newbranch2 = [ neg op_i(I, J, X1expansion) | Newbranch1],
                          closed([Newbranch2]), !,
                          closed(Rest))).

closed([Branch | Rest]) :-                              /* rules: RiIj&GjimpGi, RC2Gi */
        memberx(neg op_g(I, X), Branch),
        remove(neg op_g(I, X), Branch, Newbranch1),
        (       ( memberx( op_i(I, J, Y), Newbranch1),
                  convert_dual(X, Convx),
                  convert_dual(Y, Convy),
                  theorem(Convx eqv Convy),
                  remove(op_i(I, J, Y), Newbranch1, Newbranch2),
                  closed([[neg op_g(J, X) | Newbranch2]])
                );
                ( convert_dual(X, Convx),
                  expansion(Convx, Xexpansion, clausepl),
                  Xexpansion = [A, B | L],
                  set_partitions(Xexpansion, Set_partitions),
                  memberx([X1, Y1], Set_partitions),
```

```
                convert_clause(X1, Convx1), convert_clause(Y1, Convy1),
                expansion(Convx1, X1expansion, reduced),
                closed([[neg op_g(I, X1expansion) | Newbranch1]]),
                expansion(Convy1, Y1expansion, reduced),
                closed([[neg op_g(I, Y1expansion) | Newbranch1]]))
        ), !, closed(Rest).

closed([]).

/*
        convert_dual(X, Y):-  Y is the Lact formula that corresponds to X (dual clause form
                              notation).
*/

convert_dual([], false).
convert_dual([X], Y) :- conv_and(X,Y).
convert_dual([X,Y | L], W or Z) :- conv_and(X,W), convert_dual([Y | L], Z).

conv_and([],true).
conv_and([neg X1], neg X2) :- !, conv_and([X1], X2).
conv_and([op_e(I, L1)], e(I, L2)) :- !, convert_dual(L1, L2).
conv_and([op_g(I, L1)], g(I, L2)) :- !, convert_dual(L1, L2).
conv_and([op_i(I1, I2, L1)], i(I1, I2, L2)) :- !, convert_dual(L1, L2).
conv_and([X], X).
conv_and([X1,Y | Z], X2 and W) :- conv_and([X1], X2), conv_and([Y | Z],W).

/*
        convert_clause(X, Y):- Y is the Lact formula that corresponds to X (clause form notation).
*/

convert_clause([], true).
convert_clause([X], Y) :- conv_or(X,Y).
convert_clause([X,Y | L], W and Z) :- conv_or(X,W), convert_clause([Y | L], Z).

conv_or([],false).
conv_or([neg X1], neg X2) :- !, conv_or([X1], X2).
conv_or([op_e(I, L1)], e(I, L2)) :- !, convert_clause(L1, L2).
conv_or([op_g(I, L1)], g(I, L2)) :- !, convert_clause(L1, L2).
conv_or([op_i(I1, I2, L1)], i(I1, I2, L2)) :- !, convert_clause(L1, L2).
conv_or([X], X).
conv_or([X1,Y | Z], X2 or W) :- conv_or([X1], X2), conv_or([Y | Z],W).

/*
        ================================
        Program auxiliary predicates
        ================================
*/

memberx(X, [X | _]).
memberx(X, [_ | T]):- memberx(X, T).

remove(X, [], []).
remove(X, [X | T1], T2) :- remove(X, T1, T2).
remove(X, [Y | T1], [Y | T2]) :- X \= Y, remove(X, T1, T2).
```

```
intersection([], Y, []).
intersection([X | R], Y, [X | Z]) :- memberx(X, Y), !, intersection(R, Y, Z).
intersection([X | R], Y, Z) :- intersection(R, Y, Z).

union([], Y, Y).
union([X|L1], Y,L2) :- memberx(X, L2), remove(X, L2, LT), union(L1, Y, LT).

set_without_set(X, [], X).
set_without_set(X, [Y | L], Z) :- memberx(Y, X), !, remove(Y, X, W), set_without_set(W, L, Z).
set_without_set(X, [Y | L], Z) :- set_without_set(X, L, Z).

list_intersection([X], X).
list_intersection([X, Y | L], Z) :- intersection(X, Y, W), W \= [], !, list_intersection([W | L], Z).
list_intersection([X, Y | L], []).

list_without_set([X], Y, [Z]) :- set_without_set(X, Y, Z).
list_without_set([X1, X2 | L1], Y, [Z | L2]) :-
        set_without_set(X1, Y, Z), Z \= [], !, list_without_set([X2 | L1], Y, L2).
list_without_set([X1, X2 | L1], Y, L2) :- list_without_set([X2 | L1], Y, L2).

partition(X, Y, Z) :- union(X,Y,Z), X \= [], Y \= [].

set_partitions(S,P) :- setof([X, Y], partition(X,Y,S), Temp), bag_to_set_partitions(Temp, P).

bag_to_set_partitions([[X1, X2]], [[X1, X2]]).
bag_to_set_partitions([[X1, X2], Z | L], S) :-
        memberx([Y1, Y2], [Z | L]),
        ((equal_set(X1, Y1), equal_set(X2, Y2)); (equal_set(X1, Y2), equal_set(X2, Y1))), !,
        bag_to_set_partitions([ Z | L], S).
bag_to_set_partitions([[X1, X2], Z | L], [[X1, X2] | S]) :- bag_to_set_partitions([ Z | L], S).

equal_set([], []).
equal_set([X|A], B) :-
        memberx(X, B), remove(X, A, A1), remove(X, B, B1), equal_set(A1, B1).

if_then_else(P, Q, R) :- P, !, Q.
if_then_else(P, Q, R) :- R.
```