Formalização de Resultados Teóricos em Assistente de Provas

Maria Julia Dias Lima e Cláudia Nalon

Departamento de Ciência da Computação, Universidade de Brasília, Campus Universitário Darcy Ribeiro, Asa Norte, 70910-900, Brasília, DF, Brasil
majuhdl@gmail.com e nalon@unb.br

Resumo Formalização da sintaxe e da semântica da lógica multimodal utilizando assistente de provas Coq, com objetivo de automatização do processo, possibilitando a reutilização dos resultados obtidos.

Palavras-Chave: Lógicas não-clássicas · lógica modal · formalização · sistemas interativos de provas

1 Introdução

Lógica Modal é uma lógica que utiliza operadores modais na construção de sua linguagem [3]. Essa linguagem é utilizada no desenvolvimento de argumentos matemáticos, com base em teoremas, axiomas, criação de modelos, sistemas, provas matemáticas e suas análises.

Tal lógica consiste na utilização de operadores modais, de forma que se possa considerar como situações têm sua verdade ou falsidade alterada se consideramos diferentes situações para essas argumentações. Podemos dividir essa lógica em monomodal e multimodal, relacionando com a quantidade de operadores modais utilizados na construção da lógica (respectivamente, apenas um operador ou mais de um).

Com o objetivo de formalizar a lógica multimodal, foram criadas no provador interativo de teoremas Coq [2] definições dos principais pontos desse tipo de sistema. O Coq utiliza um cálculo construtivo indutivo. A partir dele, são construídas provas utilizando-se de táticas e teoremas já definidos e provados no processo de criação do assistente. Essas definições e as provas se encontram em bibliotecas que são importadas da base do sistema, sendo utilizadas como auxiliares no processo de provas, com o objetivo de facilitar e reutilizar os itens que já foram mostrados previamente no assistente. Tal processo de reutilização permite que seja possível também reutilizar o que está sendo criado nesse projeto em outros projetos futuros, similarmente ao que fazemos com as bibliotecas do assistente.

As definições formalizadas neste trabalho são baseadas em [1]. Inicialmente, foi definida a sintaxe, constituída das definições relacionadas a como são formadas as fórmulas. Nesse trabalho, a sintaxe é constituída das definições de fórmula bem formada, forma normal negada para as fórmulas, tamanho de fórmulas bem

formadas, igualdade sintática e simplificação de fórmulas. Além disso, foi mostrado que a função de transformação de fórmulas em sua forma normal negada é correta e que nenhuma fórmula possui tamanho zero. Com relação à semântica, foi definida a satisfatibilidade das fórmulas bem formadas em suas versão local.

Todos esses resultados foram mostrados em função da definição da semântica, definida através de estruturas (ou modelos) de Kripke [3], utilizando as definições teóricas de mundo, relações e as funções que relacionam sintaxe e seu significado nesta estrutura.

O objetivo deste trabalho é a verificação da correção dessas definições e funções, mostrando que as transformações aplicadas à sintaxe preservam significaco. Isso tudo deve ser feito de forma automatizada, utilizando a ferramenta de assistente de provas Coq [2].

Esse artigo é estruturado da seguinte maneira: na próxima seção, temos as definições formais que são utilizadas nesse projeto; na Seção3, são apresentados os resultados que foram obtidos pela formalização dos conceitos escolhidos no assistente de provas, Coq; e, por fim, concluímos na Seção 4.

2 A lógica modal

Os operadores modais são símbolos que dão às sentenças uma qualificação diferente da original, similar à função de advérbios em uma frase, expandindo o significado do que tínhamos inicialmente. Nesse sentido, podemos caracterizá-la como a expansão de lógicas que não consideram essas qualificações, como a clássica.

Essa lógica possui uma sintaxe e uma semântica. Consideramos aqui a lógica multimodal, com um ou mais operadores, sendo eles o de necessidade e seu dual, de possibilidade. Provaremos alguns resultado sobre essas definições na próxima seção.

Definimos sua sintaxe, ou seja, o formato como suas fórmulas devem ser escritas e quais operadores e símbolos utilizaremos. A sintaxe da lógica modal é formada pelo que chamamos de fórmulas bem formadas, FBFs. Primeiramente, definimos o conjunto de proposições P, o conjunto de constantes C e os utilizamos para definir as FBFs.

Definição 1 *Denotamos por* $C = \{\mathbf{true}, \mathbf{false}\}$ *o conjunto de* constantes.

Definição 2 *Denotamos por* P *o conjunto enumerável de* símbolos proposicionais*, em que* $P = \{p, q, r, ...\}$*, com elementos indexados ou não.*

Definição 3 *Denotamos por agente a um elemento do conjunto* $An = \{1, ..., n\}$*, no qual* $n \in \mathbb{N}$*.*

Definição 4 *O conjunto das fórmulas bem formadas,* FBF*s, é dado indutivamente a partir do conjunto* P*,* C *e* An*, conforme segue:*

- φ *está em* FBF*, se* $\varphi \in C$*,*

- φ *está em* FBF*, se* $\varphi \in P$*,*
- $\neg\varphi$ *está em* FBF*, se* $\varphi \in \mathsf{FBF}$*,*
- $[a]\varphi$ *está em* FBF*, se* $\varphi \in \mathsf{FBF}$ *e* $a \in A$*,*
- $(\varphi \vee \psi)$*,* $(\varphi \wedge \psi)$ *e* $(\varphi \rightarrow \psi)$ *estão em* FBF*, se* φ *e* $\psi \in \mathsf{FBF}$*s.*

Uma das noções que utilizamos é a de tamanho de uma fórmula, que é dado pela soma da quantidade de todos os símbolos proposicionais e operadores lógicos da fórmula, exceto por parênteses.

Definição 5 *O tamanho de uma fórmula é definido recursivamente por* $f : \mathsf{FBF} \longrightarrow \mathbb{N}$*:*

- $f(\varphi) = 1$*, se* $\varphi \in P$ *ou* $\varphi \in C$
- $f(\neg\varphi) = 1 + f(\varphi)$
- $f([a]\varphi) = f(\langle a\rangle\varphi) = a + f(\varphi)$
- $f((\varphi \wedge \psi)) = f((\varphi \vee \psi)) = f((\varphi \rightarrow \psi)) = 1 + f(\varphi) + f(\psi)$

Utilizamos a Forma Normal Negada (FNN) como uma forma de padronização do formato das fórmulas, com o objetivo de facilitar a leitura de sua sintaxe, além de reduzir os seus operadores somente aos de $\neg, \wedge, \vee, [a], \langle a\rangle$.

Definição 6 *A Forma Normal Negada tem o formato de uma fórmula com seus operadores reduzidos a* $\neg$*, aplicado somente a símbolos proposicionais, e aos operadores:* $\wedge, \vee, [a], \langle a\rangle$*.*

Definição 7 *A Forma Normal Negada de uma fórmula é dada pela função* $g : \mathsf{FBF} \longrightarrow \mathsf{FBF}$*, na qual* $\varphi, \psi \in \mathsf{FBF}$ *e* $p \in P$*:*

$g(p) = p$
$g(\mathbf{true}) = \mathbf{true}$
$g(\mathbf{false}) = \mathbf{false}$
$g(\neg p) = \neg p$
$g(\neg\mathbf{true}) = \mathbf{false}$
$g(\neg\mathbf{false}) = \mathbf{true}$
$g(\neg\neg\varphi) = g(\varphi)$
$g(\neg(\varphi \vee \psi)) = g(\neg\varphi) \wedge g(\neg\psi)$
$g(\neg(\varphi \wedge \psi)) = g(\neg\varphi) \vee g(\neg\psi)$
$g(\neg(\varphi \rightarrow \psi)) = g(\varphi) \wedge g(\neg\psi)$
$g(\neg[a]\varphi) = \langle a\rangle g(\neg\varphi)$
$g(\neg\langle a\rangle\varphi) = [a]g(\neg\varphi)$
$g(\varphi \vee \psi) = g(\varphi) \vee g(\psi)$
$g(\varphi \wedge \psi) = g(\varphi) \wedge g(\psi)$
$g(\varphi \rightarrow \psi) = g(\neg\varphi) \vee g(\psi)$
$g([a]\varphi) = [a]g(\varphi)$
$g(\langle a\rangle\varphi) = \langle a\rangle g(\varphi)$

Para a semântica, é utilizada a noção de modelos de Kripke [3] para a avaliação das fórmulas. Esses modelos são utilizados na definição da semântica dos operadores modais porque utilizamos a noção de mundo e das relações entre eles para definir a satisfatibilidade das fórmulas, particularmente nas que utilizam os operadores modais.

Definição 8 *Um* frame *é uma tupla* $\langle W, R_1, \ldots, R_n\rangle$*, no qual* W *é um conjunto não-vazio (de mundos possíveis) e cada* R_a *é uma relação em* W*.*

Definição 9 *Um* modelo *é uma tupla* $\langle F, \pi\rangle$. W, *no qual* F *é um* frame *e* $\pi : W \longrightarrow (\mathsf{FBF} \rightarrow \{true, false\})$ *é a função de avaliação com relação a esse* frame.

Para simplificar notação, no restante do texto nós escrevemos $\langle W, , R_1, \ldots, R_n, \pi\rangle$ ao invés de $\langle\langle W, , R_1, \ldots, R_n\rangle, \pi\rangle$. A semântica é a definição da satisfatibilidade de uma fórmula bem formada, em um determinado modelo, podendo ser ela definida de forma local ou global. Primeiramente, é preciso definir satisfatibilidade de uma fórmula em um mundo.

Definição 10 *A relação de satisfatibilidade de uma fórmula na lógica multimodal em um modelo* $M = \langle W, R, \pi\rangle$ *e um mundo* $w \in W$, *é dada pela seguinte relação:*

- $\langle M, w\rangle \models$ **true**
- $\langle M, w\rangle \not\models$ **false**
- $\langle M, w\rangle \models p$ *se, e somente se,* $\pi(w, p) = true, p \in P$;
- $\langle M, w\rangle \models \neg\varphi$ *se e somente se* $\langle M, w\rangle \not\models \varphi$
- $\langle M, w\rangle \models \varphi \wedge \psi$ *se, e somente se,* $\langle M, w\rangle \models \varphi$ *e* $\langle M, w\rangle \models \psi$;
- $\langle M, w\rangle \models \varphi \vee \psi$ *se, e somente se,* $\langle M, w\rangle \models \varphi$ *ou* $\langle M, w\rangle \models \psi$;
- $\langle M, w\rangle \models \varphi \rightarrow \psi$ *se, e somente se,* $\langle M, w\rangle \not\models \varphi$ *ou* $\langle M, w\rangle \models \psi$;
- $\langle M, w\rangle \models [a]\varphi$ *se, e somente se,* $\forall w, w' \in W, wRw' \rightarrow \langle M, w\rangle \models \varphi$;
- $\langle M, w\rangle \models \langle a\rangle\varphi$ *se, e somente se* $\exists w, w' \in W, wRw'$ *e* $\langle M, w\rangle \models \varphi$;

Podemos definir a satisfatibilidade localmente, ou seja, em função da existência de pelo menos um mundo em que a fórmula seja satisfeita.

Definição 11 *Uma fórmula* φ *é* satisfatível localmente *se, e somente se, existe um modelo* $M = \langle W, R_1, \ldots, R_n, \pi\rangle$ *e existe um mundo* $w \in W$ *tal que* $\langle M, w\rangle \models \varphi$.

A semântica é utilizada para que possamos identificar os significados de cada símbolo da fórmula, tornando possível definições como equivalência entre fórmulas, além de provas de correção e completude, e de consistência para a lógica e o cálculo que escolhermos utilizar.

3 Resultados

Como resultados, obtivemos as formalizações no Coq das definições acima, além de provas sobre a correção da transformação fornecida pelas funções de transformação de FBF em FNN e de equivalência semântica entre elas. A formalização e demais documentos referentes a este projeto podem ser encontrados em http://cic.unb.br/~nalon/#software.

3.1 Sintaxe

Inicialmente, foram definidos os conceitos de conjunto de símbolos proposicionais, *Props*, como um conjunto equivalente ao de $\mathbb{N}$ (já que P é enumerável). Além disso, o conjunto de constantes, *Const*, foi definido como contendo os símbolos *tt* e *ff*, que correspondem, na linguagem do provador às constantes **true** e **false**, respectivamente.

```
Definition Props := nat.
Inductive Const: Set := tt | ff.
```

Definiu-se uma variável n, que foi utilizada para limitar o conunto de agentes. Na definição seguinte, se relaciona um dado agente a um natural, ou seja, faz a relação de que existe um natural equivalente a esse número que está presente no conjunto. Por fim, a última definição dá exatamente a relação de que um agente pode ser transformado em um número natural.

```
Variable n : nat.

 Definition Agents :=
   {a:nat | (a ≤ n)}.

 Definition Agents_to_Nat (a:Agents) : nat :=
match a with
| exist _ m _ ⇒ m
end.

 Coercion Agents_to_Nat: Agents >-> nat.
```

A definição de fórmula bem-formada foi feita de forma indutiva, em função do tipo da fórmula, de forma similar à desenvolvida na teoria, como mostrado abaixo. Por exemplo, a linha "$PropsF : Props \longrightarrow formula$" diz que todo símbolo proposicional é uma fórmula. Já a linha "$Not : formula \longrightarrow formula$" diz que a negação de uma fórmula é uma fórmula. Ou seja, temos que a última parte mostra que temos uma fórmula e as partes anteriores definem do que é formada essa fórmula.

```
Inductive formula : Type :=
  | PropsF : Props → formula
  | ConstF : Const → formula
  | Not : formula → formula
  | And : formula → formula → formula
  | Or : formula → formula → formula
  | Imp : formula → formula → formula
  | Box : Agents → formula → formula
  | Diamond : Agents → formula → formula.
```

Definimos também a função de tamanho de fórmula, de forma similar a como definimos na teoria, recursivamente, mostrada abaixo. Ela recebe uma fórmula e retorna o seu tamanho. Abaixo, a notação `match` indica que a avaliação do argumento da função é feita por casamento de padrões; o padrão _ corresponde a todos os não anteriormente especificados.

```
Fixpoint size (f: formula) : nat :=
  match f with
  | PropsF p ⇒ 1
  | ConstF _ ⇒ 1
  | Not g ⇒ 1 + size g
  | And f g ⇒ 1 + (size f) + (size g)
  | Or f g ⇒ 1 + (size f) + (size g)
  | Imp f g ⇒ 1 + (size f) + (size g)
  | Box a f ⇒ 1 + (size f)
  | Diamond a f ⇒ 1 + (size f)
  end.
```

Definimos indutivamente a árvore sintática da fórmula, uma construção similar à anterior, mas que cria uma árvore para a formatação da fórmula. A diferença se dá apenas no formato de disposição do conteúdo, mas o objetivo de significado é o mesmo. O primeiro item dessa árvore é uma folha, que é formada de uma fórmula e um número, indicando a profundidade modal dessa folha na árvore. Então, temos os operadores unários e binários, que são formados de uma fórmula e sua profundidade na árvore e um ou dois outros "galhos"da árvore, do mesmo tipo árvore, formando, assim, uma recursão, com um ou dois filhos, respectivamente.

```
Inductive tree: Type :=
  | leaf (f:formula) (n:nat)
  | binary_op (f:formula) (n:nat) (t1 t2: tree)
  | unary_op (f:formula) (n:nat) (t1: tree).
```

Definimos abaixo uma função recursiva de formação dessas árvores, onde os itens de uma fórmula que são indivisíveis, ou seja, o *PropsF* e *ConstF*, são colocados como folhas e o n é o número inicial mínimo. Para o operador *Not*, fazemos uma recursão na árvore, criando um nó com a fórmula com Not e uma recursão com apenas o elemento interno. Para os casos de *And*, *Or* e *Imp*, a lógica é similar, porém com dois operadores, e, no caso, dois novos ramos. Nos casos de *Box* e *Diamond*, fazemos de forma similar ao *Not*, porém é adicionado 1 ao valor inicial inicial, pois o nível modal aumenta ao se encontrar uma fórmula modal nessa árvore.

```
Fixpoint insert (v:formula) (n:nat) : tree :=
  match v with
```

```
| PropsF p ⇒ leaf (PropsF p) n
| ConstF ff ⇒ leaf (ConstF ff) n
| ConstF tt ⇒ leaf (ConstF tt) n
| Not p ⇒ unary_op (Not p) n (insert p n)
| And p q ⇒ binary_op (And p q) n (insert p n) (insert q n)
| Or p q ⇒ binary_op (Or p q) n (insert p n) (insert q n)
| Imp p q ⇒ binary_op (Imp p q) n (insert p n) (insert q n)
| Box a p ⇒ unary_op (Box a p) n (insert p (add 1 n))
| Diamond a p ⇒ unary_op (Diamond a p) n (insert p (add 1 n))
end.
```

3.2 Semântica

Nas definições abaixo, extraídas da entrada para o Coq, utilizamos um novo conceito, o conceito de tipos. Um tipo é, basicamente, uma coleção de objetos. No Coq, todos os tipos são habitados, podendo ser interpretados, portanto, como conjuntos não-vazios. O tipo é utilizado para a nossa definição de mundos porque, como visto, o conjunto dos mundos não pode ser vazio na nossa teoria.

Definimos um tipo chamado *W*, para representar os mundos da estrutura modal.

```
Definition W := Type.
```

A seguinte definição é a da relação de um agente particular sobre o conjunto de mundos.

```
Definition Ra (A:Ensemble W) := Agents → W → W → Prop.
```

Definimos um caminho abaixo. Nessa definição, dividimos em um caminho vazio e um caminho com elementos. O vazio define um caminho como sendo uma combinação de que existe uma relação de um mundo *w1* com um mundo *w2*. Na segunda parte da definição, temos que um caminho de *w1* para *w2* existe se temos uma relação de *w1* com algum mundo *v* e que existe um caminho de *v* para *w2*. Ou seja, a primeira definição temos uma relação direta de *w1* para *w2* e na segunda temos uma relação com itens intermediários, saindo de *w1* e eventualmente, computando os próximos nós da relação de forma sucessiva, chegando ao final em *w2*.

```
Inductive Path (A:Ensemble W) (w1 w2:W) (R:Ra A) : Prop:=
  | empty : ∀ a, (R a (In W A w1) (In W A w2)) → Path A w1 w2 R
  | app: ∀ a v, (R a (In W A w1) (In W A v)) → (Path A v w2 R) → Path
A w1 w2 R.
```

Foram definidas propriedades relacionadas ao resultado de caminho. A primeira delas é sobre relações que sejam acíclicas, ou seja, não podem ser repetidos elementos que já estão no caminho. Na segunda, definimos o que significa um mundo tem pai, ou seja, tem uma conexão acima dele. Na terceira, definimos raiz como sendo um mundo sem ter um pai. Por fim, definimos a unicidade de raiz. Isso define as propriedades necessárias para que depois a definição de modelo em formato de árvore possa ser formalizada.

```
Definition Ra_acyclic (A:Ensemble W) (R:Ra A) := ∀ (w:W), ¬ Path A w w R.
Definition has_parent (A:Ensemble W) (R:Ra A) (w1:W) := ∃ a w, R a (In W A w) (In W A w1).
Definition is_root (A:Ensemble W) (R:Ra A) (w1:W) := ∃ a w3, R a (In W A w1) (In W A w3) ∧ ~(has_parent A R w1).
  Definition unique_root (A:Ensemble W) (R:Ra A) := ∃ w1, (is_root A R w1) ∧ ∀ a, ~(a = w1) → ~(is_root A R a).
```

Definimos, então, um *frame*, também como um tipo. Nessa definição, utilizamos o conceito de conjunto de mundos W, definido acima. Além disso, temos R, um conjunto de relações binárias. Os símbolos % `type` indicam que essa construção deve ser considerada pelo assistente de provas também como um tipo. Essa definição diz que temos um *Frame*, que é um tipo, no qual temos um conjunto de mundos e um conjunto de relações binárias, tais que as relações são definidas entre dois desses mundos.

```
Definition Frame (A:Ensemble W) : Type := (Ensemble W×Ra A).
```

A definição da função de avaliação é feita como na Seção 3.1: dado um mundo W e um símbolo proposicional (nesse caso, um dos símbolos pertencentes ao conjunto *Props*), retorna verdadeiro ou falso (como definidos pelo tipo *bool* em Coq).

```
Definition pi := W → Props → bool.
```

Por fim, a definição de modelo é dada pelas triplas ordenadas de mundos, W, relações, R, e função de avaliação, pi:

```
Definition Model (A:Ensemble W) : Type :=    (Frame A× pi) %type.
```

A definição de satisfatibilidade é feita recursivamente para cada formato de fórmula. Para um símbolo proposicional p, a função retorna True (respectivamente, False) se a função pi o avalia para verdadeiro (respectivamente, para falso). No caso de fórmulas complexas, aplica-se recursão. Por exemplo, para a fórmula $\varphi \vee \psi$, recursão é aplicada às suas subfórmulas φ e ψ. Note na definição abaixo que, na notação do provador, *Or* é o operador na linguagem-objeto; o operador $\vee$ simboliza a operação na metalinguagem.

```
Fixpoint sat (A:Ensemble W) (M:Model A) (w:W) (f:formula): Prop :=
 let R := snd(fst(M)) in
 let pi:= snd M in
  match f with
  | ConstF tt ⇒ True
  | ConstF ff ⇒ False
  | PropsF p ⇒ if (pi w p) then True else False
  | Not g ⇒ ~(sat A M w g)
  | And g h ⇒ (sat A M w g) ∧ (sat A M w h)
  | Or g h ⇒ (sat A M w g) ∨ (sat A M w h)
  | Imp g h ⇒ ~(sat A M w g) ∨ (sat A M w h)
  | Box a g ⇒ ∀ w':W, ((R a) (In W A w) (In W A w')) → (sat A M w' g)
  | Diamond a g ⇒ ∃ w':W, ((R a) (In W A w) (In W A w')) ∧ (sat A M w'
g)
  end.
```

A definição de satisfatibilidade local também segue a apresentação da Seção 3.2. Uma fórmula é localmente satisfatível, se existe um modelo e um mundo nesse modelo que satisfaçam a fórmula recebida como entrada.

```
Definition locally_satisfatible (A:Ensemble W) (f:formula): Prop := ∃
(M:Model A), ∃ w:W, sat A M w f.
```

Duas fórmulas são semanticamente equivalentes se sua avaliação é a mesma em todos os modelos e mundos:

```
Definition eq_semantica (A:Ensemble W) (f g : formula) : Prop :=
  ∀ (M:Model A) (w:W), sat A M w f = sat A M w g.
```

```
Definition tree_locally_satisfatible (A:Ensemble W) (f:formula): Prop := ∃
(M:Model A), ∃ (w:W), M_tree_like A M ∧ is_root A (snd(fst M)) w ∧ sat A
M w f.
```

```
Definition pointed_model (A:Ensemble W) (w:W) (M:Model A) := (In W
                                    (fst (fst M)) w).
```

3.3 Forma Normal Negada

A função abaixo define fórmulas na Forma Normal Negada (FNN). Tal função recebe uma fórmula em FBF e retorna verdadeiro ou falso, ou seja, se a fórmula está na FNN ou não. Lembrando, o padrão _ corresponde a todos os não anteriormente especificados. Portanto, quando o operador de negação *Not* é aplicado a símbolos proposicionais, a função retorna verdadeiro; aplicado a qualquer outra fórmula, resulta em falso.

```
Fixpoint is_NNF (f:formula): Prop :=
  match f with
  | PropsF p ⇒ True
  | ConstF ff ⇒ True
  | ConstF tt ⇒ True
  | Not (PropsF p) ⇒ True
  | Not _ ⇒ False
  | And f g ⇒ (is_NNF f ) → (is_NNF g)
  | Or f g ⇒ (is_NNF f ) → (is_NNF g)
  | Imp _ _ ⇒ False
  | Box a f ⇒ (is_NNF f)
  | Diamond a f ⇒ (is_NNF f)
  end.
```

Definimos a função indutiva de transformação de uma fórmula em sua FNN no formato abaixo, preservando as características teóricas. A função recebe uma fórmula em FBF como entrada e retorna outra fórmula em FBF. É importante observar que o Coq só aceita a formalização de funções totais e terminantes. Observa-se da definição abaixo que a função é de fato total. Entretanto, o assistente de prova não consegue encontrar automaticamente as condições de terminação. O motivo é que o resultado da função é aplicado a fórmulas que não são subfórmula da entrada. Por exemplo, da Definição 7, o resultado da aplicação a $\neg(\varphi \vee \psi)$ é o resultado da recursão sobre $\neg\varphi$ e $\neg\psi$. Estas últimas não são subfórmulas de $\neg(\varphi \vee \psi)$ e, portanto, o assistente não consegue extrair o princípio de indução adequado e provar automaticamente sua terminação. Entretanto, é claramente observável que a recursão ocorre em argumentos de tamanhos menores do que o original. Por isso, esta função foi definida de forma mais geral (utilizando `Function` ao invés de `Fixpoint`) e adicionando-se que o cálculo da terminação é feita em relação ao tamanho da fórmula. Isto é expresso pela anotação {`measure` *size f*} na seguinte definição.

```
Function NNF (f:formula) {measure size}: formula :=
  match f with
  | PropsF p ⇒ PropsF p
  | ConstF ff ⇒ ConstF ff
  | ConstF tt ⇒ ConstF tt
  | Not (PropsF p) ⇒ Not (PropsF p)
  | Not (ConstF ff) ⇒ ConstF tt
  | Not (ConstF tt) ⇒ ConstF ff
  | Not (Not f) ⇒ NNF f
  | Not (And f g) ⇒ Or (NNF (Not f)) (NNF (Not g))
  | Not (Or f g) ⇒ And (NNF (Not f)) (NNF (Not g))
  | Not (Imp f g) ⇒ And (NNF f) (NNF (Not g))
  | Not (Box a f) ⇒ Diamond a (NNF (Not f))
  | Not (Diamond a f) ⇒ Box a (NNF (Not f))
  | And f g ⇒ And (NNF f) (NNF g)
```

```
| Or f g ⇒ Or (NNF f) (NNF g)
| Imp f g ⇒ Or (NNF (Not f)) (NNF g)
| Box a f ⇒ Box a (NNF f)
| Diamond a f ⇒ Diamond a (NNF f)
end.
```

A prova de terminação é feita por indução no tamanho da fórmula, sendo que a maior parte dos casos é obtida automaticamente através de simplificação pelo assistente de prova, utilizando aritmética linear inteira. Para o caso da implicação, foi necessário utilizar fatos sobre desigualde entre inteiros, mas o restante da prova é obtida automaticamente também com o auxílio da implementação dos procedimentos de prova, no Coq, para a aritmética linear inteira.

Além de totalidade e terminação, é essencial que provemos que a função *NNF* é correta, isto é, mostrar que o resultado da sua aplicação está, de fato, no formato de FNN. O próximo lema mostra a correção da função:

```
Theorem NNF_is_NNF (f:formula) :
  is_NNF (NNF f).
```

A prova do teorema acima é feita por indução sobre o resultado da aplicação de (*NNF f*), que se baseia na medida de complexidade (no tamanho da fórmula) dada acima. A prova é obtida automaticamente pelo provador.

O teorema seguinte mostra que a função de transformação de uma fórmula na sua forma normal negada preserva seu valor semântico.

```
Theorem eq_NNF_f (A:Ensemble W) (f : formula) :
  ∀ (M:Model A) (w:W), sat A M w f ↔ sat A M w (NNF f).
```

A prova foi feita por indução em (*FNN f*), assim como na prova de terminação da aplicação da função *FNN* apresentada na seção anterior. Para os casos-base, isto é constantes, símbolos proposicionais ou negações de símbolos proposicionais, as provas são simples e seguem diretamente da definição de satisfatibilidade para estas fórmulas (porque a fórmula e sua forma normal negada coincidem). A hipótese de indução é de que a propriedade vale para fórmulas de tamanho menor que a fórmula que está sendo analisada. Para fórmulas complexas, são aplicadas as hipóteses e a conclusão segue em cada caso com poucas linhas de prova. Por exemplo, precisamos mostrar que, dados um modelo M e um mundo w, o resultado da aplicação de *sat M w* para a fórmula *Not* (*Not Props p*) é exatamente o mesmo que *sat M w* (*Props p*), na qual esta última fórmula é o resultado da aplicação *NNF Not* (*Not Props p*).

4 Conclusão

Neste trabalho, apresentamos uma formalização da lógica multimodal no assistente de provas Coq, baseada em teoremas e definições teóricas. Assim, mostramos ser possível a formalização de conceitos matemáticos teóricos de sintaxe

e semântica, relacionados à lógica monomodal na linguagem computacional do assistente de provas. Em um próximo projeto de pesquisa, essa lógica será expandida para a multimodal, completando, assim, essa etapa de formalizações.

Referências

1. Cláudia Nalon, Clare Dixon, and Ullrich Hustadt. 2019. Modal Resolution: Proofs, Layers, and Refinements. ACM Trans. Comput. Logic 20, 4, Article 23 (August 2019), 38 pages. https://doi.org/10.1145/3331448
2. The Coq Proof Assistant, version 8.12.2 (December 2020) https://doi.org/10.5281/zenodo.4501022
3. Melvin Fitting, Richard L. Mendelsohn: First-Order Modal Logic. Springer Science+Business Media, Dordrecht (1998) https://doi.org/10.1007/978-94-011-5292-1