UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SILVIA PIMPÃO VASQUEZ

IMPLEMENTAÇÃO DE UM SISTEMA PARA MODELAGEM EPISTÊMICA DE CONHECIMENTOS E VALIDAÇÃO DE SEGURANÇA EM PROTOCOLOS DE COMUNICAÇÃO

RIO DE JANEIRO
2020

SILVIA PIMPÃO VASQUEZ

IMPLEMENTAÇÃO DE UM SISTEMA PARA MODELAGEM EPISTÊMICA DE CONHECIMENTOS E VALIDAÇÃO DE SEGURANÇA EM PROTOCOLOS DE COMUNICAÇÃO

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Mário Roberto Folhadela Benevides

RIO DE JANEIRO

2020

CIP - Catalogação na Publicação

V335i

Vasquez, Silvia Pimpão
Implementação de um sistema para modelagem epistêmica de conhecimentos e validação de segurança em protocolos de comunicação / Silvia Pimpão Vasquez. -- Rio de Janeiro, 2020.
59 f.

Orientador: Mário Roberto Folhadela Benevides.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2020.

1. Lógica epistêmica. 2. Segurança. 3. Protocolos. 4. Modelagem de conhecimento. 5. Verificador de modelos. I. Benevides, Mário Roberto Folhadela, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

SILVIA PIMPÃO VASQUEZ

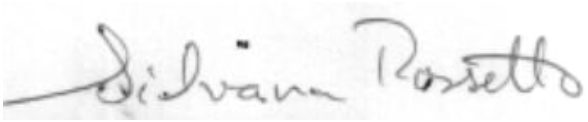
IMPLEMENTAÇÃO DE UM SISTEMA PARA MODELAGEM EPISTÊMICA DE CONHECIMENTOS E VALIDAÇÃO DE SEGURANÇA EM PROTOCOLOS DE COMUNICAÇÃO

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 15 de Dezembro de 2020

BANCA EXAMINADORA:

Prof. Mário Roberto Folhadela Benevides
Ph.D (DCC-IM-UFRJ)



Profª. Silvana Rossetto
D.Sc (DCC-IM-UFRJ)

Prof. Bruno Lopes Vieira
D.Sc (DCC-IC-UFF)

RESUMO

Com o crescente uso da Internet, é cada vez maior o volume de informações trafegadas entre pessoas do mundo todo. Essas informações muitas das vezes são sigilosas e espera-se que os meios de comunicação utilizados online adotem as medidas necessárias para garantir a integridade e a privacidade das mesmas. Torna-se relevante neste cenário a existência de métodos que avaliem a segurança tanto de protocolos de comunicação já amplamente adotados quanto de protocolos ainda em fase de concepção. Neste trabalho, é apresentado um programa que permite a modelagem e a avaliação da segurança em um dado protocolo de comunicação de forma visual, através de uma interface disponibilizada online. São utilizadas metodologias provenientes da Lógica Epistêmica Multi-Agente que possibilitam a modelagem de conhecimentos e crenças de determinados agentes. Para avaliar a implementação do programa, foram modelados três protocolos de comunicação que utilizam algoritmo de chave pública e privada apresentados no trabalho de Dolev e Yao. Também foi realizada uma modelagem adicional baseada no famoso problema lógico das crianças com lama na testa. Os resultados obtidos nas modelagens realizadas conferem com o esperado, demonstrando que o programa cumpre com os objetivos pretendidos de forma correta.

Palavras-chave: Lógica Epistêmica Multi-Agente. Segurança. Protocolo. Verificador.

ABSTRACT

The volume of information exchanged between people in the world through Internet becomes larger every day. Frequently, this information exchanged is private and it's expected that the comunication protocols used online follow the necessary methods to ensure its integrity and privacy. In this scenario, methodologies that can verify security of protocols both widely used and still in conception phase becomes relevant. In this work, it's presented a software application capable of visually modeling and evaluating security of a communication protocol through an interface available online. Knowledge and beliefs of known agents are modeled using methodologies from Multi-Agent Epistemic Logic area. Three public and private keys communication protocols presented in Dolev and Yao's work are modeled to check correctness of the sotfware application presented. An additional model based on the famous muddy children logic puzzle is also presented. In all presented models, the obtained results comply correctly with the expectations for this work.

Keywords: Multi-Agent Epistemic Logic. Security. Protocol. Checker.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Ao longo das últimas décadas, a Internet cresceu de forma que cada vez mais pessoas possuem meios de acessar e interagir através dela. Nem sempre essas pessoas possuem boas intenções, sendo necessário garantir a segurança e a privacidade das informações, principalmente as sensíveis, trafegadas através da rede para que não caiam em mãos erradas.

Uma forma de garantir essa segurança é através de algoritmos de criptografia, que codificam as mensagens originais de forma que elas não sejam trafegadas em texto claro e que somente os destinatários pretendidos possuam alguma forma de decodificá-las. Porém, apenas utilizar um método criptográfico seguro pode não ser suficiente. Em seu trabalho, (DOLEV; YAO, 1983) provaram que um intruso é capaz de descobrir o conteúdo de uma mensagem trafegada entre dois agentes mesmo que as mensagens trocadas possuam uma criptografia segura, pois a lógica do protocolo em si não provia segurança.

É possível ainda analisar a segurança de um protocolo através da modelagem de conhecimentos de cada um dos agentes envolvidos na comunicação, avaliando se os mesmos chegam a adquirir conhecimento sobre o conteúdo em texto claro das mensagens criptografadas trocadas. Essa modelagem de conhecimentos pode ser realizada através de conceitos provenientes de uma das sub-áreas de lógica denominada lógica epistêmica. Nela, é possível especificar quais conhecimentos cada um dos agentes possui e quais crenças eles acreditam ser verdadeiras, porém sem ter a certeza.

Essas crenças e conhecimentos são dinâmicas e sofrem mudanças conforme ações são realizadas, como por exemplo o envio de uma mensagem de um agente para outro. Essas ações são modeladas como anúncios, onde todos ou alguns agentes tomam conhecimento sobre a veracidade ou não de novas informações. Isso possivelmente leva à perda das dúvidas que alguns agentes poderiam anteriormente ter e, consequentemente, alteram a modelagem de conhecimento inicialmente elaborada.

De posse do modelo e da definição de ações realizadas pelos agentes, é possível verificar se há comprometimento da privacidade dos agentes se um intruso na comunicação, em algum momento, toma conhecimento sobre o conteúdo original da mensagem trafegada. Essa verificação é feita através do envio de perguntas lógicas ao modelo.

Este trabalho se propõe a analisar, através da modelagem epistêmica de conhecimentos dos agentes, se um dado protocolo provê privacidade para a comunicação entre agentes legítimos. Foi desenvolvido um programa[1] capaz de gerar modelos epistêmicos, alterar conhecimentos e validar informações. Três modelagens são apresentadas em detalhes, baseadas nos três exemplos de protocolos de chave pública e privada discutidos no trabalho de (DOLEV; YAO, 1983). Nas três modelagens, chega-se às mesmas conclusões apresen-

[1] Programa disponível em: https://spimpaov.github.io/tcc/

tadas no trabalho original quanto à segurança de cada protocolo, o que serve como uma prova de corretude para o programa e suas funcionalidades.

Uma modelagem adicional foi realizada para este trabalho a fim de mostrar que o programa implementado pode ser utilizado na resolução de problemas mais gerais na área da lógica epistêmica. Foi modelado o conhecido problema lógico comumente chamado de 'Crianças com lama na testa' (PLAZZA, 1989) (FAGIN et al., 2003).

1.1 OBJETIVO

Como objetivo principal, o trabalho pretende se mostrar como uma ferramenta útil na concepção e avaliação de protocolos de comunicação seguros. Modelando esses protocolos logicamente, é possível obter garantias teóricas em relação à segurança que eles providenciam, como demonstrado em (DOLEV; YAO, 1983). Porém, os modelos utilizados para fazer essa avaliação são em sua maioria pouco visuais e de difícil entendimento. O programa implementado se mostra como uma alternativa para tornar essa validação lógica mais viável e visual. Foi desenvolvida uma interface com a qual é possível interagir através de mouse e teclado para criar, editar e apagar os grafos que representam os modelos de conhecimento dos agentes.

Como objetivo adicional, o programa também visa auxiliar no estudo e visualização de modelagens epistêmicas de forma mais geral. Para demonstrar que isso é uma possibilidade no programa, o conhecido problema lógico chamado 'Crianças com lama na testa' foi modelado e resolvido utilizando a implementação apresentada (PLAZZA, 1989) (FAGIN et al., 2003).

1.2 ORGANIZAÇÃO DO TEXTO

O restante do trabalho está organizado da seguinte forma: o Capítulo 2 traz explicações sobre conceitos necessários para o entendimento do trabalho provenientes do campo da lógica e de modelagem de conhecimentos, bem como a reprodução dos três exemplos discutidos em (DOLEV; YAO, 1983). O Capítulo 3 comenta sobre a implementação do programa, com um manual de uso para a interface. O Capítulo 4 apresenta os modelos implementados e o Capítulo 5, por fim, traz considerações finais e ideias para trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo são apresentados os conceitos necessários para o entendimento da proposta de solução apresentada neste trabalho. Na Seção 2.1, são apresentados conceitos pertinentes para as modelagens e validações lógicas que serão apresentados. Para a Seção 2.2, são apresentadas formas possíveis de representar alterações nas modelagens feitas. Um detalhamento maior sobre o trabalho de (DOLEV; YAO, 1983) é realizado na Seção 2.3. Por fim, é feita uma analise comparativa com trabalhos relacionados.

2.1 LÓGICA EPISTÊMICA MULTI-AGENTE

A Lógica Epistêmica Multi-Agente é uma sub-área de Lógica que busca modelar os conhecimentos e crenças de determinados agentes. Essa modelagem se baseia na existência de diversos mundos ou estados possíveis, onde afirmações sobre o conhecimento geral dos agentes dependem diretamente do conhecimento que possuem em cada um desses mundos possíveis (FAGIN et al., 2003).

Adaptando o exemplo de (FAGIN et al., 2003), se está chovendo no Rio de Janeiro, um agente carioca que esteja no local pode afirmar com certeza que está chovendo, pois ele está presenciando o acontecimento com os próprios olhos. Em outras palavras, pode-se afirmar que o agente carioca *sabe* que está chovendo. Entretanto, se o carioca for questionado sobre o tempo em São Paulo, ele não poderá afirmar com certeza, pois existem vários mundos possíveis (chuva, sol, nublado etc.) e a confirmação não pode ser feita da mesma forma como antes (observação do tempo com os próprios olhos). Em outras palavras, o agente carioca *não sabe* se está chovendo em São Paulo, porém *acredita* na possibilidade, já que estar chovendo em São Paulo é um dos mundos possíveis. Em notação lógica, as certezas do agente carioca são representadas como K de *knows* e as crenças são representadas como B de *beliefs*.

Também chamada de lógica $S5_M$, por ser uma evolução da lógica $S4_M$ considerando vários agentes com conhecimentos próprios, a Lógica Epistêmica Multi-Agente possui diversas aplicações, como em protocolos e em teoria dos jogos (FAGIN et al., 2003) (DITMARSCH; HOEK; KOOI, 2007).

2.1.1 Linguagem

A linguagem para a Lógica Epistêmica Multi-Agente pode ser definida como: Φ representando o conjunto enumerável de símbolos proposicionais, A representando o conjunto finito de agentes, $\neg$ e $\wedge$ representando os conectivos *booleanos* NOT e AND, respectivamente, K_a como operador de conhecimento para cada agente a existente e B_a como operador de crenças para cada agente a existente (DITMARSCH; HOEK; KOOI, 2007).

As fórmulas são definidas seguindo o padrão abaixo:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid K_a\varphi \mid B_a\varphi$$

onde φ, φ_1 e φ_2 são expressões lógicas, $p \in \Phi$, $a \in A$ e $\top$ é uma abreviação para $\neg\varphi \vee \varphi$ (DITMARSCH; HOEK; KOOI, 2007) (FAGIN et al., 2003).

Algumas equivalências podem ser consideradas a fim de facilitar a notação de algumas expressões (DITMARSCH; HOEK; KOOI, 2007):

$$\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$$

$$\neg\top \equiv \neg(\neg\varphi \vee \varphi) \equiv \bot$$

$$B_a\varphi \equiv \neg K_a \neg\varphi$$

Retomando o exemplo dado anteriormente, se o agente carioca for representado como c, a chuva no Rio de Janeiro como $ChuvaRJ$ e a chuva em São Paulo como $ChuvaSP$, é possível representar os conhecimentos do carioca na linguagem epistêmica da seguinte forma:

$$K_c ChuvaRJ$$

$$B_c ChuvaSP$$

2.1.2 Modelo

Um modelo epistêmico é definido como $M = (S, R_a, V)$, onde S é um conjunto não-vazio de estados, R é um conjunto de relações binárias entre estados contidos em S para todo $a \in A$ e V é uma função de valoração $V : \Phi \to 2^S$. A função de valoração é a responsável por fazer a tradução de um símbolo proposicional qualquer $p \in \Phi$ para *true* ou *false*. Um símbolo proposicional p é dado como *true* em um determinado estado s se $V(p)$ contém s. Caso contrário, p é dado como *false*.

Dado um modelo epistêmico $M = (S, R_a, V)$, $s \in S$ e $a \in A$, a noção de satisfação $M, s \vDash \varphi$ é definida como (DITMARSCH; HOEK; KOOI, 2007):

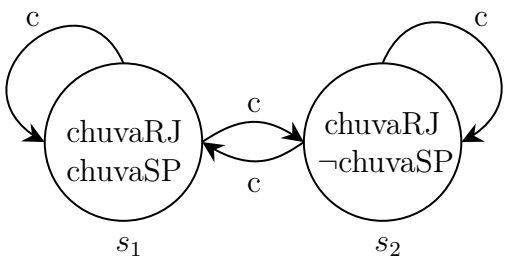
- $M, s \vDash p$ se e somente se $s \in V(p)$
- $M, s \vDash \neg\varphi$ se e somente se $M, s \nvDash \varphi$
- $M, s \vDash \varphi \wedge \psi$ se e somente se $M, s \vDash \varphi$ e $M, s \vDash \psi$
- $M, s \vDash \varphi \to \psi$ se e somente se $M, s \nvDash \varphi$ ou $M, s \vDash \psi$
- $M, s \vDash K_a\varphi$ se e somente se, para todo $s' \in S$, se sR_as' então $M, s' \vDash \varphi$
- $M, s \vDash B_a\varphi$ se e somente se existe um $s' \in S$ tal que sR_as' e $M, s' \vDash \varphi$

É possível designar um modelo epistêmico para representar os conhecimentos do agente carioca sobre o tempo no Rio de Janeiro e São Paulo. Considerando c como o agente carioca, $chuvaRJ$ como a chuva no Rio de Janeiro, $chuvaSP$ como a chuva em São Paulo e $\neg chuvaSP$ como qualquer outro tempo possível em São Paulo que não seja chuva, é possível descrever um modelo M como:

- $M = (S, R_c, V)$
  - $S = \{s_1, s_2\}$
  - $R_c = \{(s_1, s_1), (s_2, s_2), (s_1, s_2), (s_2, s_1)\}$
  - $V(chuvaRJ) = \{s_1, s_2\}$
  - $V(chuvaSP) = \{s_1\}$

A representação gráfica do modelo M descrito pode ser vista na Figura 1.

Figura 1 – Modelo com os conhecimentos acerta do tempo no Rio de Janeiro e em São Paulo pelo agente carioca.



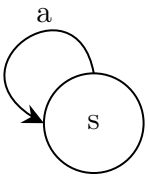
2.1.3 Frame

Em Lógica Epistêmica, um *frame* é uma tupla $F = (S, R_a)$, onde S e R_a seguem a mesma definição vista na representação de modelos. É possível descrever um modelo epistêmico baseado em um *frame* F da seguinte forma: $M = (F, V)$. V é uma função de valoração descrita da mesma forma como na definição de modelo epistêmico dada anteriormente. Se uma fórmula φ é verdadeira para um dado *frame* F, isso significa que φ é verdadeiro para todo modelo epistêmico baseado em F (FAGIN et al., 2003).

Algumas relações binárias entre estados são particularmente interessantes no estudo de *frames*. É o caso das relações de reflexividade, transitividade e simetria. A reflexividade se dá quando um modelo é tal que todos os seus estados S tenham relação consigo mesmo por um agente qualquer $a \in A$ (FAGIN et al., 2003):

$$sR_as, \forall s \in S$$

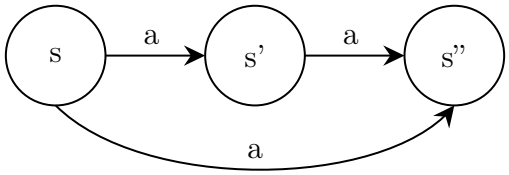
Figura 2 – Relação reflexiva.



A relação de transitividade se dá quando um modelo é tal que, se um estado s possui relação com um estado s' e s' possui relação com um estado s'' por um agente qualquer a, então s também possui relação com o estado s'' por a (FAGIN et al., 2003):

$$sR_as' \wedge s'R_as'' \rightarrow sR_as'', \forall s, s', s'' \in S$$

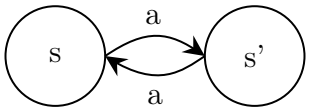
Figura 3 – Relação transitiva.



A simetria, por sua vez, ocorre quando um modelo é tal que, se um estado s possui relação com um estado s' por um agente qualquer a, então s' também possui relação com s por a (FAGIN et al., 2003):

$$sR_as' \rightarrow s'R_as, \forall s, s' \in S$$

Figura 4 – Relação simétrica.



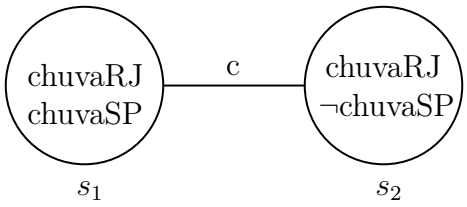
Com base nessas relações binárias, é possível construir *frames* onde cada um dos estados presentes cumprem as especificações de um determinado tipo de relação. Por exemplo, um *frame* é dito reflexivo se para todo $s \in S$ a relação sR_as acontece para todo agente $a \in A$. Isso é útil pois permite a construção de axiomas para um determinado *frame* e, consequentemente, para um modelo baseado nesse *frame*. Por exemplo, a expressão $K_a\varphi \rightarrow \varphi$ é verdadeira no *frame* reflexivo, já que a expressão é válida para todos os estados. Logo, ela é também verdadeira em todo modelo baseado no *frame* reflexivo (DITMARSCH; HOEK; KOOI, 2007).

Todas as expressões mostradas abaixo são fórmulas válidas para *frames* reflexivos, transitivos e simétricos:

- $K_a\varphi \rightarrow \varphi$
- $K_a(\varphi \rightarrow \psi) \rightarrow (K_a\varphi \rightarrow K_a\psi)$
- $K_a\varphi \rightarrow K_aK_a\varphi$
- $\neg K_a\varphi \rightarrow K_a\neg K_a\varphi$

Retomando o exemplo do agente carioca, é possível simplificar o desenho do modelo (Figura 1) transformando as relações simétricas observadas em linhas ao invés de setas e ocultando as relações reflexivas, como mostrado no modelo redesenhado da Figura 5. Nessa versão simplificada, é comum descrever cada linha como uma representação das dúvidas de um determinado agente. Nesse exemplo, poderia ser dito que o agente c está em dúvida sobre qual dentre os estados s_1 e s_2 é o estado verdadeiro, ou seja, qual representa fielmente a realidade sobre o tempo de Rio de Janeiro e São Paulo.

Figura 5 – Modelo simplificado com os conhecimentos acerta do tempo no Rio de Janeiro e em São Paulo pelo agente carioca.



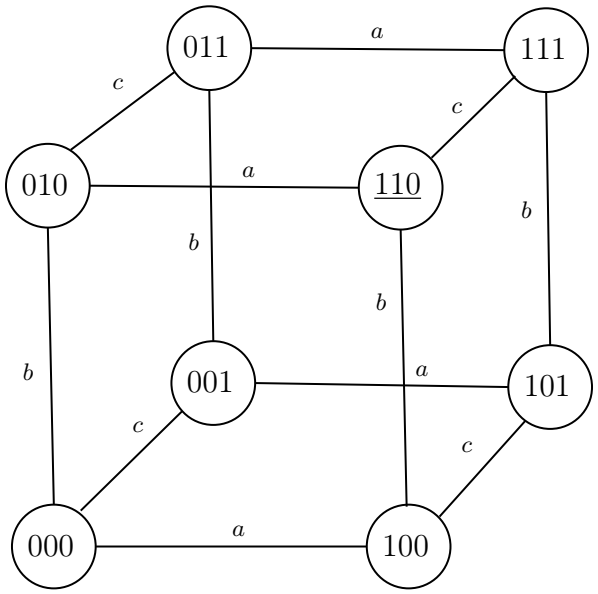
2.1.4 Exemplo: Crianças com Lama na Testa

Modelagens epistêmicas são bastante utilizadas na resolução de problemas lógicos. Um exemplo bastante conhecido na área é o 'Crianças com lama na testa'. Nele, n crianças precisam descobrir se têm ou não a testa suja de lama. Cada criança consegue ver o estado da testa dos colegas mas não consegue ver e nem pode sentir se está com a própria testa suja ou não (PLAZZA, 1989) (FAGIN et al., 2003).

Supondo que n seja 3 e apelidando as crianças de Ana, Beto e Carla, é possível representar as dúvidas desses três agentes através de um modelo epistêmico, exibido na Figura 6. Cada estado é composto de três números, cada um podendo ser 0 ou 1. O número 0 indica que a criança não possui lama na testa e 1 indica que ela possui. A ordem do número indica a qual criança a informação se refere, considerando a ordem Ana, Beto, Carla. Assim, o estado 100 representa o mundo possível onde apenas Ana possui lama na testa; no estado 101, Ana e Carla possuem lama na testa, mas Beto não.

O estado 110 é suposto como o estado que representa a realidade na brincadeira, isto é, Ana e Beto possuem lama na testa e Carla não.

Figura 6 – Modelo epistêmico para o problema das crianças com lama na testa.



As arestas representam as dúvidas que os agentes possuem entre estados. Por exemplo, o agente a (representando Ana) possui dúvida entre os estados 110 e 010 pois o estado da testa dos colegas permanece o mesmo, somente o estado da testa de a permanece desconhecido por ela, já que ela pode estar suja em 110 ou limpa em 010. De forma generalizada, todos os agentes apenas possuem dúvidas entre nós do grafo onde a única variação é a testa do próprio agente, já que essa é a única informação desconhecida.

Para resolver esse problema lógico, é preciso determinar ações que resultam em alterações no conhecimento original modelado, impactando diretamente no desenho do grafo. Geralmente essas alterações estão relacionadas ao ganho de conhecimento por parte de alguns ou de todos os agentes, o que resulta na perda das dúvidas e, consequentemente, na poda de arestas ou estados no grafo.

2.2 MODELANDO MUDANÇAS NO CONHECIMENTO

Algumas extensões da Lógica Epistêmica Multi-Agente foram desenvolvidas a fim de representar mudanças ocorridas dentre os conhecimentos dos agentes. Essas extensões compõem uma modalidade chamada Lógica Epistêmica Dinâmica. Dentro dessa área, mudanças no conhecimento são chamadas de anúncios e podem ocorrer de diversas formas, como por exemplo publicamente, privadamente para algum agente, ou privadamente para um subgrupo dos agentes. Diversos tipos de anúncios já foram discutidos em trabalhos como (GERBRANDY; GROENEVELD, 1997).

2.2.1 Anúncios Públicos

A Lógica de Anúncios Públicos é uma extensão à Logica Epistêmica Multi-Agente que considera a ocorrência de anúncios realizados publicamente. Isto é, todos os agentes tomam ciência do que foi informado pelo anúncio no momento em que ele ocorre (PLAZZA, 1989).

Retornando ao exemplo anterior sobre o agente carioca e suas dúvidas com relação ao tempo, um exemplo de anúncio público que alteraria os conhecimentos do agente c poderia ser um amigo que more em São Paulo lhe informar sobre o estado chuvoso do tempo. A partir do momento que o agente c toma conhecimento de que está de fato chovendo em São Paulo assim como chove no Rio de Janeiro, o modelo sobre seus conhecimentos se altera e passa a considerar essa nova condição.

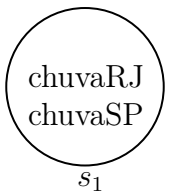
Em outras palavras, todos os estados em que $chuvaSP$ não é verdadeiro são cortados do modelo, sobrando apenas os estados que satisfazem o que foi proferido pelo anúncio público, isto é, onde $chuvaSP$ é verdadeiro. O modelo resultante após esse corte pode ser visto na Figura 7, onde pode-se notar que o estado s_2 foi cortado, restando apenas o estado s_1. O modelo só possui um único estado porque o agente não possui mais dúvidas.

A linguagem da Lógica de Anúncios Públicos é semelhante à da Lógica Epistêmica Multi-Agente, seguindo o padrão abaixo:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid K_a\varphi \mid B_a\varphi \mid [\varphi_1]\varphi_2$$

onde φ, φ_1 e φ_2 são expressões lógicas, $p \in \Phi$, $a \in A$ e $\top$ é uma abreviação para $\neg\varphi \vee \varphi$ (DITMARSCH; HOEK; KOOI, 2007).

Figura 7 – Modelo atualizado representando os conhecimentos do agente carioca com relação ao tempo no Rio de Janeiro e em São Paulo.



A única novidade é a parte $[\varphi_1]\varphi_2$, que pode ser lida como a pergunta "após anúncio de φ_1, é verdade que φ_2?". Assim, o anúncio de que está chovendo em São Paulo poderia ser escrito como $[chuvaSP]$. Considerando o modelo sem cortes descrito anteriormente na Figura 5, a pergunta $[chuvaSP]chuvaRJ \wedge chuvaSP$ retorna $true$ justamente porque o operador $[chuvaSP]$ se encarrega de cortar todos os estados em que $chuvaSP$ é falso, avaliando o restante da expressão ($chuvaRJ \wedge chuvaSP$) apenas nos estados que sobraram após esse corte.

A noção de satisfação da Lógica de Anúncios Públicos é também semelhante a da Lógica Epistêmica Multi-Agente, havendo apenas a adição da seguinte consideração:

- $M, s \vDash [\varphi]\psi$ se e somente se $M, s \vDash \varphi$ implica em $M|_\varphi, s \vDash \psi$

  onde $M|_\varphi = (S', R', V')$ é definido como:

  - $S' = [\varphi]_M$
  - $R'_a = R_a \cap ([\varphi]_M \times [\varphi]_M)$
  - $V'_p = V_p \cap [\varphi]_M$

(DITMARSCH; HOEK; KOOI, 2007)

2.2.2 Anúncios Privados

Os anúncios privados no campo da lógica epistêmica geralmente são representados como ações epistêmicas, onde se deseja modelar alguma ação realizada que concede novas informações a um agente ou grupo de agentes específico. Os anúncios públicos, anteriormente discutidos, nada mais são do que modelos de ação onde, para todos os agentes, a mesma informação é anunciada (DITMARSCH; HOEK; KOOI, 2007) (EIJCK, 2004).

Anúncios privados são mais complexos pois lidam com um conjunto restrito de agentes e há a possibilidade dos demais agentes estarem cientes sobre essas ações ou não. Quando um ou mais agentes obtêm determinada informação como verdadeira, os demais agentes podem saber que o anúncio privado foi feito ou não. Caso não saibam, o anúncio é dito secreto (BENTHEM; EIJCK; KOOI, 2006).

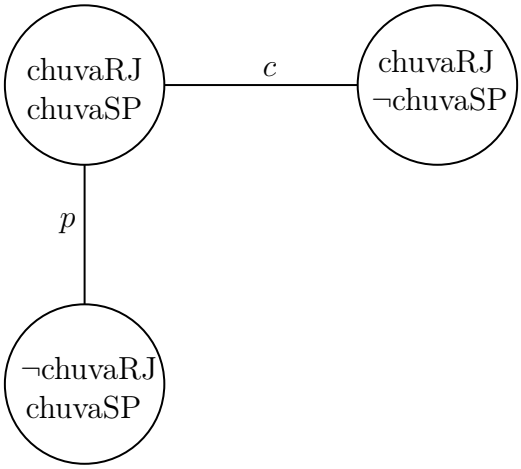
Para este trabalho, foi utilizado um modelo de ação baseado na funcionalidade do comando *info* do trabalho DEMO, descrito em (EIJCK, 2004) e (EIJCK, 2007). Nesse tipo de anúncio privado um agente ou grupo de agentes se torna ciente do valor de determinada informação de forma transparente. Em termos lógicos, esse anúncio é descrito como $(\varphi \rightarrow C_B\varphi) \wedge (\neg\varphi \rightarrow C_B\neg\varphi)$, onde C_B representa o conhecimento de um agente ou grupo de agentes B. A transparência se dá pois todos os agentes sabem que B recebeu essa informação.

Assim, se é anunciado a um dado agente a que a informação p é verdadeira, a saberá distinguir estados da modelagem onde p é verdadeiro de estados onde p é falso, e vice-versa. Isso é representado no grafo através do corte de arestas de dúvidas de a entre esses estados. Entretanto, após o anúncio, a ainda terá dúvida entre estados vizinhos onde p é verdadeiro em ambos, ou onde p é falso em ambos. Ou seja, as arestas entre esses estados se mantêm.

Para entender melhor como esse anúncio funciona na prática, o exemplo do agente carioca será revisitado. Agora, imagine um modelo que considere não apenas o conhecimento de um agente carioca c mas também os conhecimentos de um agente paulista p. O

carioca sabe que está chovendo no Rio de Janeiro, mas não sabe o tempo de São Paulo. O paulista, ao contrário, sabe que chove em São Paulo, porém não sabe o tempo do Rio de Janeiro. Essa situação está representada na modelagem epistêmica apresentada na Figura 8.

Figura 8 – Modelo epistêmico representando os conhecimentos iniciais dos agentes c e p.



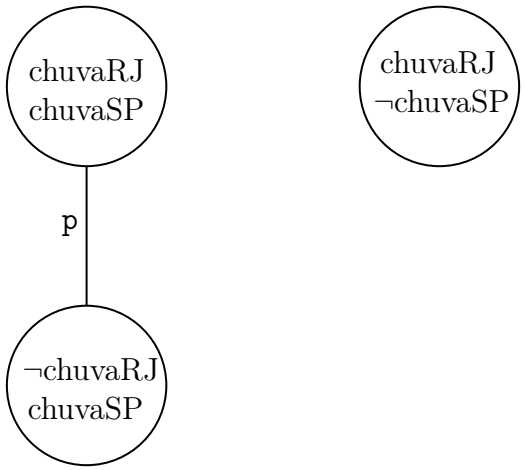
Pode-se exemplificar esse modelo de anúncio privado como a ação do agente carioca ver na televisão uma notícia sobre temporais em São Paulo. Nesse momento, o agente aprende que $chuvaSP$ é verdadeiro e a única aresta de dúvida do agente c presente na modelagem é cortada, pois agora o agente tem certeza sobre o estado do tempo em Rio de Janeiro e em São Paulo. Por outro lado, o agente paulista não assistiu a mesma notícia vista pelo agente carioca e não viu nenhuma outra notícia que informasse sobre o tempo no Rio de Janeiro. Portanto, as arestas de dúvida de p permanecem inalteradas, como mostrado na Figura 9. O ganho de conhecimento pelo agente c é descrito como um anúncio privado pois ele impacta os conhecimentos somente deste agente, enquanto os conhecimentos de p permanecem inalterados.

De forma geral, pode-se dizer que anúncios públicos cortam estados do grafo modelo enquanto anúncios privados cortam arestas. Isso se dá justamente porque anúncios públicos invalidam mundos possíveis, já que todos os agentes compartilham do mesmo ganho de conhecimento proveniente do que é anunciado. Anúncios privados, por sua vez, invalidam dúvidas, já que lidam com os conhecimentos específicos de cada agente (DITMARSCH; HOEK; KOOI, 2007).

2.2.3 Exemplo: Anúncios Públicos no Problema das Crianças com Lama na Testa

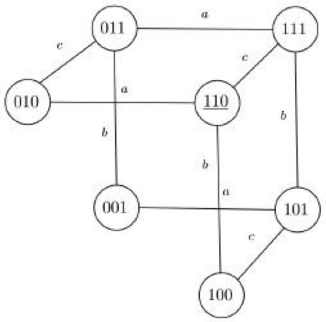
Retomando o exemplo das crianças com lama na testa, a solução poderia ser obtida através da realização de anúncios públicos para as crianças. Geralmente o pronunciador desses anúncios é representado como o pai ou um responsável por elas, e ele começa

Figura 9 – Conhecimentos de c e p após c receber um anúncio privado sobre o tempo em São Paulo.

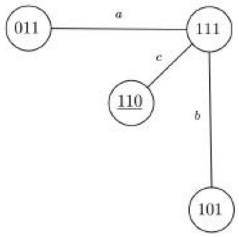


anunciando publicamente que existe pelo menos uma criança com lama na testa. Ao escutar isso, todas as crianças passam a saber que o estado onde nenhuma possui lama na testa (no grafo, o nó com valor 000) é impossível e, portanto, tanto o estado quanto as dúvidas relacionadas a ele são excluídas. O resultado pode ser observado na Figura 10a.

Em seguida, um novo anúncio público é feito: o responsável pergunta "alguém já sabe se tem lama na testa?" e nenhuma criança responde. A pergunta e a ausência de uma resposta implicam no anúncio público de que ninguém ainda sabe se tem lama na testa. O resultado disto é a exclusão dos estados onde apenas uma criança tem lama na testa, como mostrado na Figura 10b. Isso se dá pois, se fosse o caso de existir somente uma criança com lama na testa, ela teria respondido "sim" à pergunta do pai, já que ela sabe que existe pelo menos uma criança com lama na testa e todos os colegas que ela vê não têm a testa suja.



(a) O estado 000 deixa de ser um dos mundos possíveis na modelagem de conhecimento das crianças.



(b) Estados 010, 001 e 100 passam a ser impossíveis na modelagem.

(c) Somente o estado 110 permanece possível após o anúncio.

Figura 10 – Problema das crianças com lama na testa após realização dos anúncios públicos.

Na segunda vez que o responsável realiza a mesma pergunta, considerando o estado 110 como real, Ana e Beto já sabem que têm lama na testa. Eles sabem disso pois sabem que a esse ponto existem pelo menos duas pessoas com lama na testa, já que ninguém respondeu à pergunta anterior do responsável, e ambos veem somente um colega com a testa suja. Ou seja, a segunda criança com lama na testa só pode ser ela própria. Portanto, para o segundo questionamento do responsável, Ana e Beto respondem "sim".

Esse anúncio está representado na Figura 10c. Carla, ao ver os dois amigos responderem, agora sabe que ela mesma não possui lama na testa, pois se ela também tivesse, os amigos ainda não teriam certeza. Portanto, todas as dúvidas restantes são desfeitas e o estado real é o único mundo possível na modelagem de conhecimento das crianças.

2.3 MODELO DOLEV-YAO

Em (DOLEV; YAO, 1983), os autores propuseram um modelo para analisar a segurança de certas famílias de protocolos criptográficos de comunicação. A partir desse modelo, seria possível verificar se um certo tipo de protocolo provê privacidade para os agentes envolvidos, considerando a presença não só de agentes maliciosos passivos (que somente interceptam mensagens trafegadas) como também a de agentes maliciosos ativos (que personificam outros agentes e alteram o conteúdo das mensagens). A privacidade da comunicação é dita quebrada quando um agente malicioso obtém acesso às mensagens originais dos agentes legítimos (DOLEV; YAO, 1983).

Tomando uma certa família de protocolos como exemplo, o modelo Dolev-Yao se propõe a definir uma notação para essa família, um modelo de execução que define as capacidades dos agentes e as formas de troca de mensagens e uma linguagem formal que a família deve seguir a fim de garantir a segurança dos protocolos contidos nela (DOLEV; YAO, 1983).

Um dos modelos analisados por (DOLEV; YAO, 1983) é referente à família de protocolos que utilizam chaves pública e privada, onde cada participante X da comunicação possui duas funções:

- E_X: função de encriptação que utiliza a chave pública de X;
- D_X: função de decodificação que utiliza a chave privada de X.

(DOLEV; YAO, 1983) (RIVEST; SHAMIR; ADLEMAN, 1978)

É necessário que $E_X(D_X) = D_X(E_X)$. Dessa forma, se um usuário Y tem acesso a $E_X(M)$, isto é, a mensagem encriptada por X, Y não é capaz de obter acesso direto à mensagem M. Por outro lado, X obtém acesso à mensagem M pois $D_X(E_X(M)) = M$ (DOLEV; YAO, 1983) (DIFFIE; HELLMAN, 1976) (RIVEST; SHAMIR; ADLEMAN, 1978).

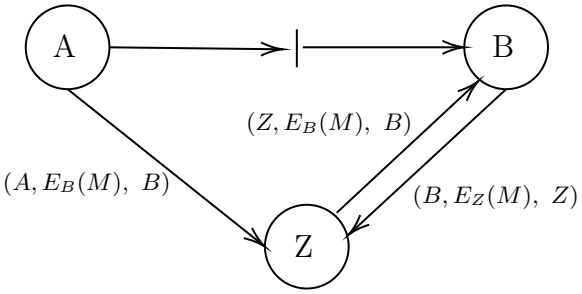
São descritos três exemplos dessa família de protocolos, a fim de avaliar se a privacidade dos agentes é mantida em cada caso (DOLEV; YAO, 1983).

2.3.1 Exemplo 1

Para o protocolo do exemplo 1 de (DOLEV; YAO, 1983), é adotado o formato (*remetente*, *texto*, *destinatário*) para as mensagens trafegadas. O fluxo de troca de mensagens segue como descrito a seguir:

1. A envia uma mensagem ao agente B com o formato $(A, E_B(M), B)$;
2. Essa mensagem é interceptada por um intruso Z, que a envia ao agente B se passando por A, alterando a mensagem para $(Z, E_B(M), B)$. É importante notar que, nesse momento, o conteúdo encriptado da mensagem $E_B(M)$ não pode ser alterado pois Z não possui meios para descriptografá-la;
3. O agente B, ao receber a mensagem, responde ao agente Z $(B, E_Z(M), Z)$;
4. O agente Z obtém acesso ao texto M pois $D_Z(E_Z(M)) = M$.

Figura 11 – Troca de mensagens do exemplo 1 de (DOLEV; YAO, 1983).



O fluxo é ilustrado na Figura 11. Esse modelo de protocolo de chave pública se mostra, portanto, inseguro (DOLEV; YAO, 1983).

2.3.2 Exemplo 2

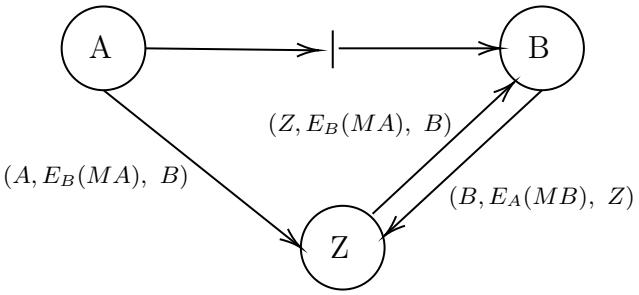
Uma possível adaptação poderia ser feita ao protocolo anterior de forma a torná-lo seguro: codificar o nome do remetente em conjunto com o texto da mensagem. Dessa forma, a troca de mensagens poderia ser reescrita para esse exemplo como:

1. A envia uma mensagem ao agente B com o formato $(A, E_B(MA), B)$, onde MA representa a concatenação do texto com o remetente da mensagem;
2. Essa mensagem é interceptada por um intruso Z, que a envia ao agente B se passando por A, enviando a mensagem $(Z, E_B(MA), B)$;

3. O agente B, ao receber a mensagem, responde ao agente Z $(B, E_A(MB), Z)$. B codifica a mensagem utilizando a chave pública de A pois esse é o agente que ele encontra ao descriptografar a mensagem recebida;

4. O agente Z não obtém acesso ao texto M pois o texto da mensagem foi criptografado utilizando a chave pública de A (agente concatenado ao texto criptografado) e não a de Z (agente destinatário da mensagem).

Figura 12 – Troca de mensagens do exemplo 2 de (DOLEV; YAO, 1983).



O fluxo é ilustrado na Figura 12. Nesse cenário, a privacidade da comunicação entre os agentes A e B não é comprometida e o modelo de protocolo se mostra seguro (DOLEV; YAO, 1983).

2.3.3 Exemplo 3

Pode-se pensar ainda numa tentativa adicional de incrementar a segurança do protocolo descrito através da adição de mais um passo de encriptação do texto contido na mensagem. Dessa forma, a troca ocorreria da seguinte forma:

1. A envia uma mensagem ao agente B com o formato $(A, E_B(E_B(M)A), B)$;

2. B responde A retornando a mensagem $(B, E_A(E_A(M)B), A)$;

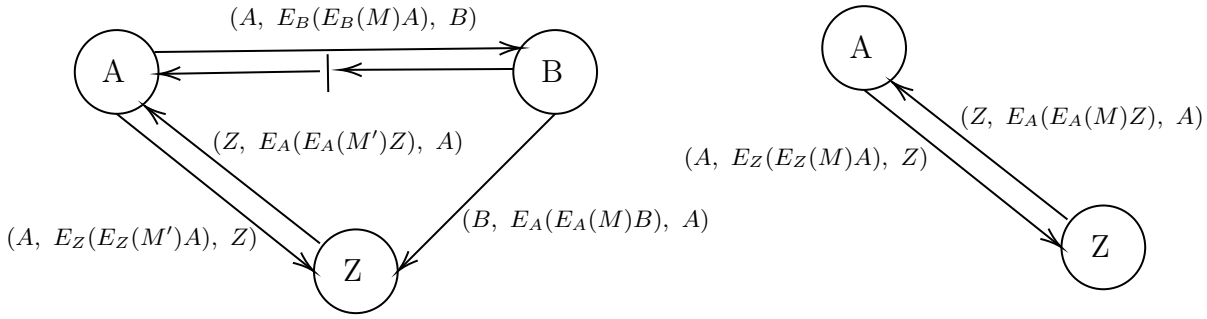
3. A mensagem enviada de B para A é interceptada por um agente intruso Z. Denotando o texto da mensagem $E_A(M)B$ como M', Z envia a mensagem $(A, E_A(E_A(M')Z), A)$ para A;

4. A responde Z enviando a mensagem $(A, E_Z(E_Z(M')A), Z)$;

5. Z consegue descriptografar o que recebeu e obter M'. Como $M' = E_A(M)B$, Z agora sabe $E_A(M)$. O intruso envia então uma nova mensagem a A com o conteúdo $(Z, E_A(E_A(M)Z), A)$;

6. A responde Z com a mensagem $(A, E_Z(E_Z(M)A), Z)$;

Figura 13 – Troca de mensagens do exemplo 3 de (DOLEV; YAO, 1983).



7. Z agora é capaz de obter M pois $D_Z(E_Z(E_Z(M)A)) = E_Z(M)A$ e $D_Z(E_Z(M)) = M$

O fluxo é ilustrado na Figura 13. Nesse exemplo, apesar da adição de mais uma operação de encriptação no texto trafegado pelas mensagens, a segurança da comunicação é comprometida (DOLEV; YAO, 1983).

2.4 TRABALHOS RELACIONADOS

Alguns trabalhos se dedicaram à analisar protocolos de comunicação e validar sua segurança, como o trabalho (DOLEV; YAO, 1983), que é um dos mais reconhecidos na área. Em (DOLEV; YAO, 1983), são apresentados alguns modelos formais para avaliar a segurança de famílias de protocolos. O modelo Dolev-Yao influenciou fortemente o trabalho aqui apresentado por utilizar exemplos de protocolos que poderiam ser modelados segundo a lógica epistêmica. A proposta de ambos os trabalhos é similar, porém a deste difere no sentido de que se deseja fazer uma análise visual, utilizando a modelagem de conhecimentos dos agentes.

Em (BENEVIDES; FERNANDEZ; OLIVEIRA, 2018), os autores apresentam uma extensão da lógica epistêmica multi-agente a fim de validar protocolos de segurança. Essa extensão é aplicada a protocolos reais e modernos como o *Kerberos* e o *Andrew Secure RPC*. Nesse trabalho, a ação de um agente A enviar uma mensagem a um agente B é descrita através de uma regra denominada $send_{[AB]}$, que se assemelha à abordagem realizada nesse trabalho, modelando a ação como um anúncio privado. Também há a definição de conhecimentos iniciais para os agentes em (BENEVIDES; FERNANDEZ; OLIVEIRA, 2018) baseado no conhecimento dos próprios sobre suas chaves privadas e públicas, o que foi também implementado para este trabalho, como será visto no Capítulo 3.

A implementação para o trabalho foi inspirada pelo programa DEMO (*Demo of Epistemic Modelling* ou *Dynamic Epistemic MOdelling*, como descrito no trabalho original), apresentado em (EIJCK, 2007). O DEMO é uma implementação escrita em linguagem Haskell com diversas funcionalidades para gerar modelos de conhecimento epistêmico e

validar informações sobre ele. O programa se assemelha ao deste trabalho pois ambos possibilitam a criação de tais modelos e a visualização gráfica deles. A diferença está na forma como isso é feito: no DEMO, é gerado um arquivo `.svg` que descreve o grafo gerado, enquanto neste trabalho a visualização está disponível *online* e é possível interagir diretamente com ela. Outra diferença está na proposta: o DEMO tem como objetivo ser uma ferramenta completa para resolução de problemas lógicos através de modelagem epistêmica e, portanto, possui mais funcionalidades. A implementação aqui realizada tem como objetivo principal a modelagem e validação de protocolos de comunicação, a fim de avaliar a segurança dos mesmos, de forma muito mais visualmente agradável e intuitiva.

Em (ROSCOE, 1995) é estudada uma forma de verificação de modelos com finalidade de validar a segurança de protocolos. Porém, a modelagem discutida não se baseia nos conceitos epistêmicos aqui apresentados e não leva em consideração a validação dos conhecimentos dos agentes envolvidos.

3 IMPLEMENTAÇÃO

Neste capítulo a implementação realizada para o trabalho é detalhada, com orientações de uso e explicações sobre todas as suas funcionalidades. Na Seção 3.1, é feita uma descrição geral do ambiente e de suas funcionalidades, apresentando a linguagem de programação e bibliotecas utilizadas. Explicações sobre como gerar grafos representando os modelos de conhecimento são apresentadas na Seção 3.2. A Seção 3.3 explica como definir um conjunto de conhecimentos iniciais para os agentes do modelo. A Seção 3.4 mostra como é feita a implementação dos anúncios no programa. Orientações para fazer perguntas ao modelo estão descritas na Seção 3.5. Por fim, a Seção 3.6 termina este capítulo com informações sobre como carregar exemplos já pré-cadastrados no programa.

3.1 DESCRIÇÃO DO AMBIENTE

Foi desenvolvida uma interface *web* para o trabalho utilizando a linguagem JavaScript e a biblioteca p5.js. A biblioteca p5.js oferece uma enorme gama de funcionalidades gráficas que foram utilizadas para a implementação do trabalho (MCCARTHY; REAS; FRY, 2015).

O código do programa é aberto e está disponível na plataforma Github através do seguinte endereço: https://github.com/spimpaov/tcc. A interface *web*, por sua vez, pode ser acessada em https://spimpaov.github.io/tcc. Essa interface possibilita a criação de modelos epistêmicos e a visualização de seus grafos correspondentes. Os grafos desses modelos são compostos por um conjunto de agentes e um conjunto de estados, onde cada estado possui determinadas proposições como verdadeiras ou falsas. De forma equivalente, o grafo também pode ser interpretado como um banco de dados, onde todos os agentes e conhecimentos adquiridos por eles se encontram mapeados.

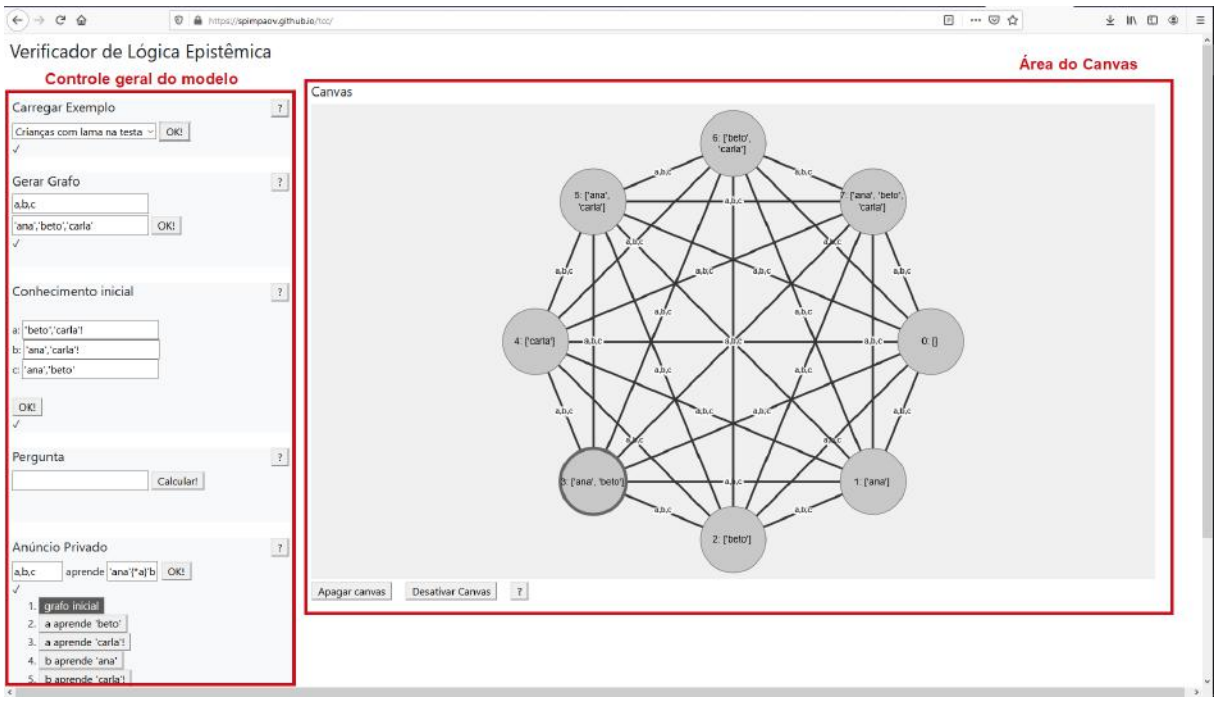
Uma vez que um modelo seja definido, é possível avaliar a veracidade de determinadas proposições lógicas nesse grafo através de perguntas, por exemplo, para saber se determinados agentes sabem que uma certa informação é verdadeira ou falsa. Também é possível realizar anúncios privados no formato "agente aprende proposição". Os anúncios são importantes pois é através deles que o estado atual do grafo pode ser modificado, possivelmente excluindo dúvidas que os agente antes possuíam. Além disso, também é possível definir os conhecimentos iniciais de determinados agentes, de forma que as proposições inseridas serão verdadeiras desde o princípio e perdurarão nas possíveis atualizações do grafo através dos anúncios.

A ideia é que seja possível modelar protocolos de comunicação e verificar se os mesmos proveem segurança e privacidade na comunicação entre agentes. Dessa forma, foram utilizados os exemplos de chave pública-privada de (DOLEV; YAO, 1983) apresentados

no Capítulo 2 a fim de checar a corretude do programa.

A interface é composta por duas partes principais (Figura 14): o Canvas (onde será exibido o grafo) e uma área para as demais funcionalidades do programa. Para todas as suas funcionalidades, o programa possui um botão correspondente com um símbolo de interrogação (’?’) que exibe na tela um texto de ajuda para aquela funcionalidade.

Figura 14 – Apresentação das duas áreas principais da interface do programa.



Um manual de utilização detalhado sobre o Canvas pode ser lido no Apêndice A. As demais funcionalidades da interface serão discutidas nas seções a seguir.

3.2 GERANDO O GRAFO

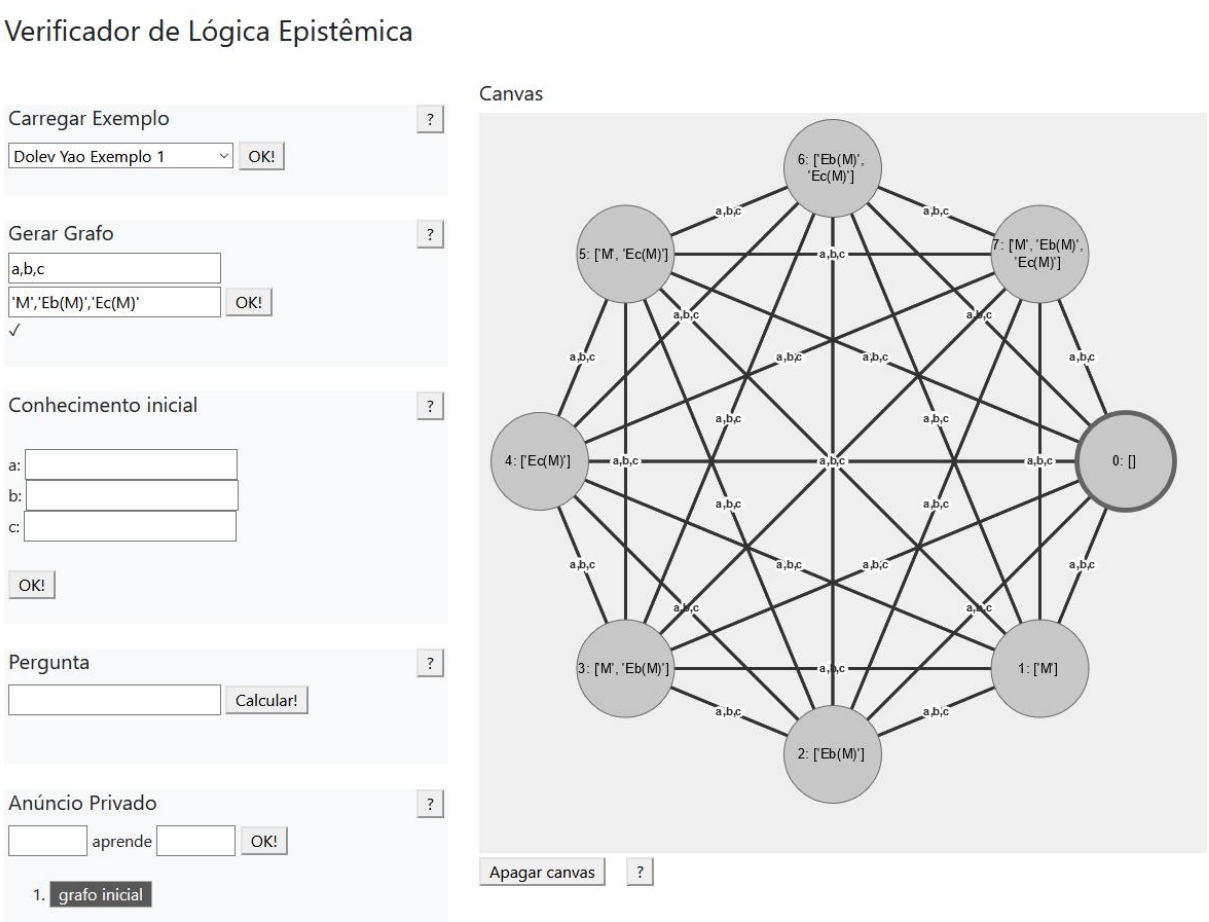
Pode-se definir um grafo para representar o modelo epistêmico do programa através de duas formas: manualmente criando estados e transições no Canvas ou gerando automaticamente através de uma lista de agentes e uma lista de proposições fornecidas ao programa. Para criar manualmente, basta seguir as instruções de criação de estados e transições contidas no manual de uso do Canvas apresentado no Apêndice A.

Para criar automaticamente, é preciso inserir na entrada `’Gerar Grafo’` do programa uma lista de agentes separados por vírgula (por exemplo, `a,b,c`) e uma lista de proposições também separadas por vírgula. Caso as proposições sejam formadas cada uma por mais de um caractere, devem ser delimitadas por aspas simples (por exemplo, `’M’,’Eb(M)’,’Ec(M)’`). Depois disso, basta clicar no botão `’OK!’` e o grafo será gerado automaticamente e exibido no Canvas.

O programa utiliza as proposições passadas como parâmetro para calcular todas as possibilidades de estados possíveis considerando que cada proposição pode ser verdadeira ou falsa. Se uma proposição é verdadeira num dado estado, ela aparece compondo a lista de proposições exibidas dentro do estado no Canvas. Se a proposição é falsa, ela apenas não é exibida. Além disso, no grafo gerado automaticamente, existem arestas de todos para todos os estados e com todos os agentes. Isso significa que, no grafo inicial gerado, todos os agentes possuem dúvida entre todos os estados.

Um exemplo de grafo gerado a partir dos agentes `a,b,c` e das proposições `'M', 'Eb(M)', 'Ec(M)'` pode ser visto na Figura 15.

Figura 15 – Grafo gerado automaticamente a partir de uma lista de agentes e proposições.



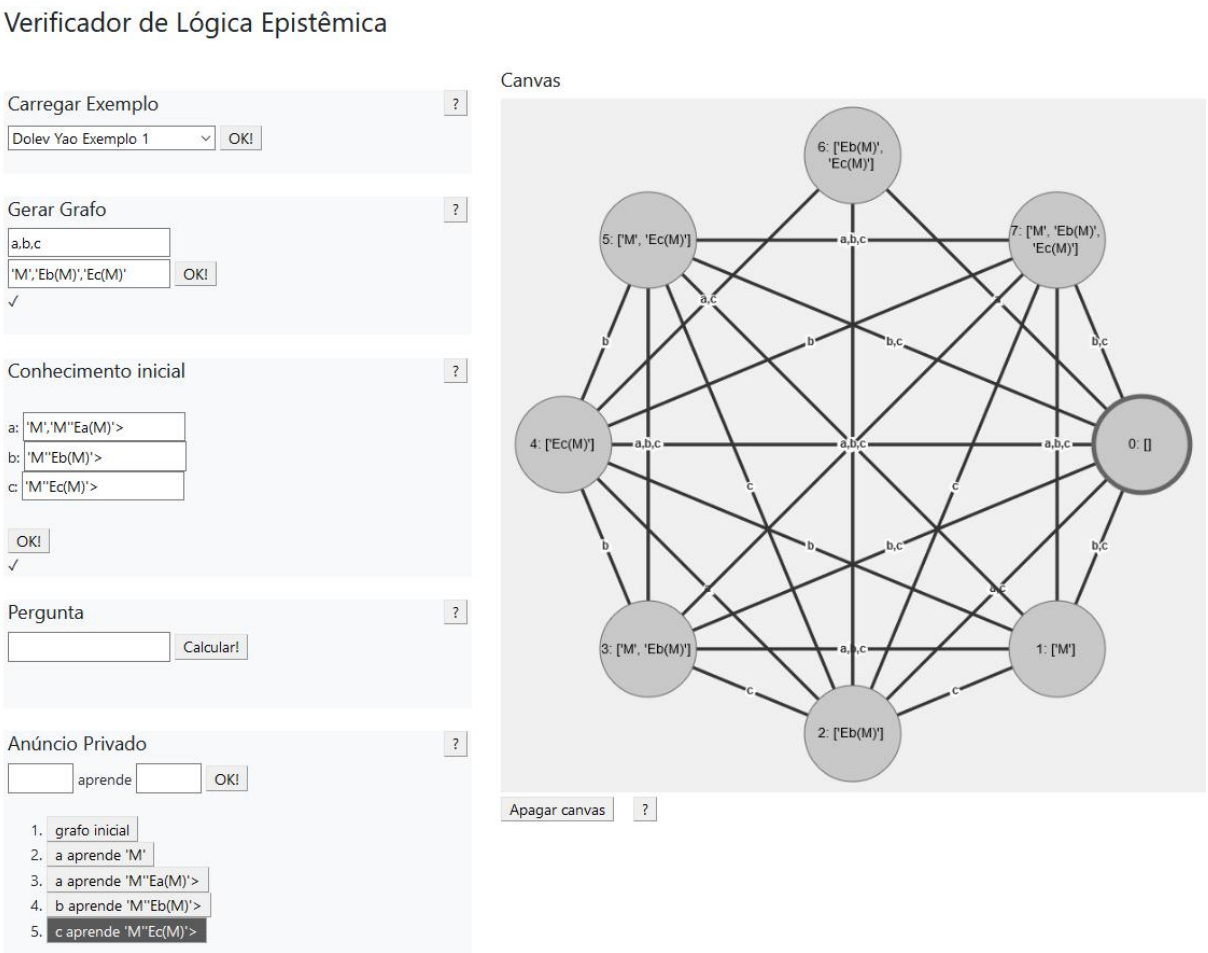
3.3 DEFININDO OS CONHECIMENTOS INICIAIS

É possível definir os conhecimento iniciais de cada agente para grafos gerados automaticamente via `'Gerar Grafo'` na interface. Logo após a geração automática do grafo, serão criados n caixas de texto na seção `'Conhecimento inicial'` da interface, onde n é o número de agentes passados para o programa. Essa funcionalidade permite que sejam definidos conhecimentos que serão verdadeiros para determinados agentes durante toda

a execução posterior do programa, independente dos anúncios feitos. Para os exemplos provenientes de (DOLEV; YAO, 1983), essa funcionalidade será útil para que um agente possa obter uma mensagem em texto claro a partir de uma mensagem encriptada com sua chave pública.

Um exemplo de uso desta funcionalidade pode ser observado na Figura 16. Nela, foi gerado um grafo com três agentes (`a,b,c`) e três proposições (`'M', 'Eb(M)', 'Ec(M)'`). Após este passo, três campos de texto, cada um referente a um agente, foram criados na seção `'Conhecimento Inicial'` da interface. Em cada uma dessas entradas, as expressões precisam estar separadas por vírgula e seguir a notação pós-fixada (mesma utilizada no campo `'Pergunta'` da interface, como será explicado na Seção 3.5).

Figura 16 – Conhecimento inicial definido e já computado pelo programa.



Para o agente `'a'`, foram definidos dois conhecimentos que são verdadeiros:

- `'M'`, pois `'a'` é o criador da mensagem e, portanto, sabe seu conteúdo original;
- `'M''Ea(M)'>`, que está, assim como o campo de `'Pergunta'` na interface, denotado em notação pós-fixada e significa $E_a(M) \rightarrow M$. Esse conhecimento inicial permite que um agente descriptografe mensagens criptografadas com sua chave pública.

Após definir os conhecimentos iniciais de cada agente enviando os dados ao programa, as dúvidas, representadas através de arestas no grafo, são atualizadas. Além disso, logo abaixo do campo `'Anúncio Privado'`, é possível observar uma atualização no histórico de execução do programa, que recebeu uma nova entrada para cada uma das expressões computadas.

3.4 REALIZANDO ANÚNCIOS

Como discutido na subseção 2.2.2, o modelo de anúncio privado implementado é descrito em termos lógicos como $(\varphi \rightarrow C_B\varphi) \wedge (\neg\varphi \rightarrow C_B\neg\varphi)$, onde C_B representa o conhecimento de um agente ou grupo de agentes B. Esse anúncio segue a execução do comando *info* dos trabalhos (EIJCK, 2007) e (EIJCK, 2004).

Na implementação realizada, se por exemplo `'a'` aprende `'M'`, `'a'` saberá distinguir estados onde `'M'` é verdadeiro de estados onde `'M'` é falso, e vice-versa. Porém, `'a'` ainda terá dúvida entre estados vizinhos onde `'M'` é verdadeiro em ambos, ou onde `'M'` é falso em ambos. Um pseudocódigo é mostrado no Algoritmo 1 para ilustrar a lógica implementada no código do programa quando um anúncio é feito.

Algoritmo 1: Pseudocódigo ilustrando a implementação dos anúncios

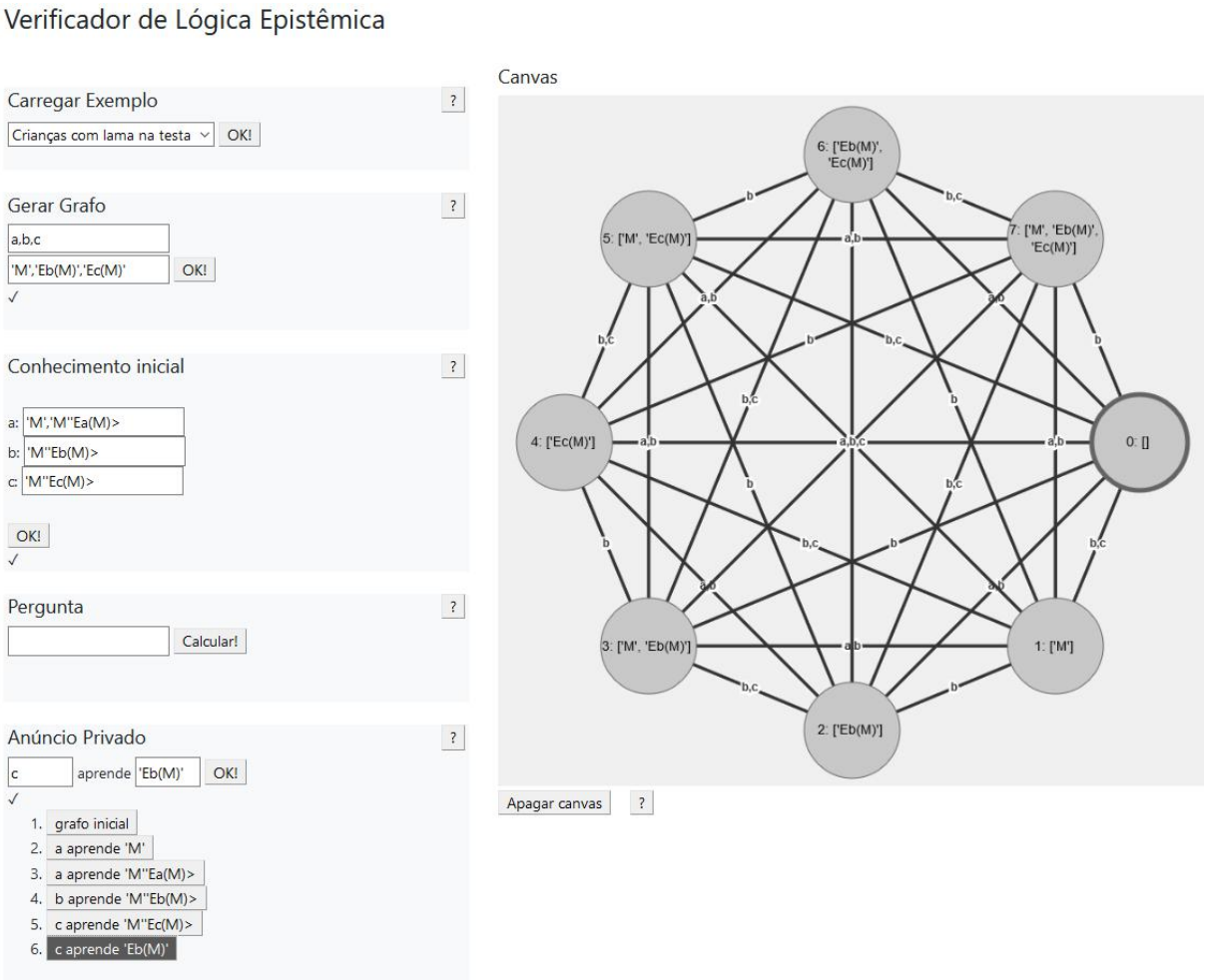
```
Entrada: Expressão anunciada (exp) e agente que escuta o anúncio (a)
Saída: Arestas marcadas que serão posteriormente excluídas do grafo
para cada estado s do grafo faça
    s_resultado ← calcula(exp, s);
    para cada estado v vizinho de s por a faça
        v_resultado ← calcula(exp, v);
        se s_resultado diferente de v_resultado então
            arestas_marcadas ← aresta entre s e v por a;
        fim
    fim
fim
retorna arestas_marcadas;
```

Esse anúncio é chamado de privado na interface do programa pois apenas alguns agentes recebem a informação anunciada, porém o anúncio não é secreto pois os demais agentes sabem que ele foi realizado. Isso é justificável para as modelagens feitas no trabalho, discutidas em detalhes no Capítulo 4, pois considera-se um cenário onde qualquer agente poderia estar visualizando o que é trafegado na rede por programas analisadores de pacote, ou *packet sniffers* como o programa *Wireshark* (OREBAUGH; RAMIREZ; BEALE, 2006).

Para fazer um anúncio privado, é preciso definir um agente ou conjunto de agentes e uma expressão para ser avaliada. Os valores são inseridos no campo 'Anúncio Privado' da interface. Na Figura 17, o agente `'c'` aprende uma nova informação `'Eb(M)'`. Ao fazer

esse anúncio, a aresta que interligava os estados `'1'` e `'2'` por `'c'` foi apagada, pois o agente agora consegue discernir esses estados. Porém, a aresta entre os estados `'2'` e `'3'`, por exemplo, ainda permanece após o anúncio pois ambos os estados possuem `'Eb(M)'` como informação verdadeira, e `'c'` ainda não consegue discernir entre eles.

Figura 17 – Anúncio privado sendo executado na interface.



Toda vez que um anúncio é feito, um novo botão é adicionado no histórico. É possível clicar nesses botões para visualizar estados anteriores do grafo, antes que modificações tenham sido feitas nele, seja através de anúncios ou da definição dos conhecimentos iniciais.

3.5 CONSTRUINDO A PERGUNTA

A lógica epistêmica se utiliza de diversos símbolos proposicionais derivados da lógica clássica para representar seus operadores. Como muitos desses símbolos não se encontram facilmente no teclado, o programa faz a tradução de determinados caracteres para que os mesmos sejam interpretados como operadores lógicos. A relação entre caracteres de entrada e operadores lógicos está mostrada na Tabela 1.

Tabela 1 – Relação de operadores lógicos para caracteres de entrada aceitos pelo programa.

Operador	Caractere
$\neg$	`!`
$\wedge$	`^`
$\vee$	`U`
$\rightarrow$	`>`
K_n	`{*n}`
B_n	`{?n}`
[ ]	`@`

A pergunta feita ao banco de dados é interpretada em notação pós-fixada, isto é, primeiro são passados os operandos e depois o operador que atuará sobre eles. Por exemplo, para validar a expressão $\neg P$ sobre o grafo definido, seria necessário passar como entrada a expressão `'P!'`, onde `'!'` é o caractere do teclado que representa o operador NOT. Uma relação entre expressões lógicas de exemplo e as correspondentes entradas para o programa é mostrada na Tabela 2.

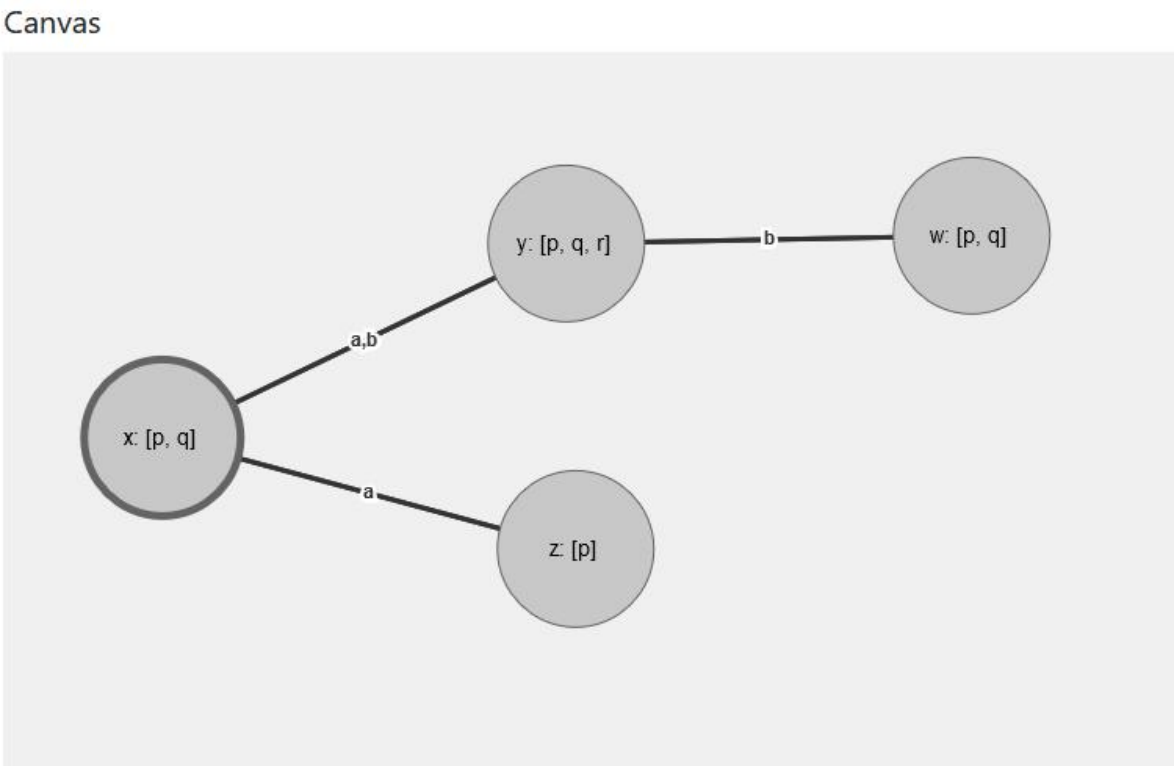
Tabela 2 – Relação de expressões lógicas para perguntas aceitas pelo programa.

Expressão	Pergunta
$\neg p$	`p!`
$p \wedge q$	`pq^`
$p \rightarrow q$	`qp>`
$(p \vee \neg r) \rightarrow q$	`qr!pU>`
$K_a(B_b p)$	`p{?b}{*a}`
$K_a p \rightarrow K_b(K_c p)$	`p{*c}{*b}p{*a}>`
$[p]q$	`qp@`
$[p](p \wedge q)$	`pq^p@`

É utilizada notação pós-fixada pois, dessa forma, torna-se muito mais fácil fazer a interpretação das expressões internamente. O programa mantém uma pilha para a entrada digitada no campo 'Pergunta' e itera de trás para a frente nela, identificando operadores e operandos. Cada operador sabe sobre quantos operandos deve atuar e, ao identificar um operador na pilha de entrada, o programa gera uma segunda pilha para armazenar o resultado das valorações dos operandos encontrados (*true* ou *false*) sobre a qual o operador identificado irá atuar. Caso um dos operandos também possua um operador, o processo se repete de forma recursiva. O programa parte sempre de um estado definido como raiz (por padrão, o estado de ID 0) para validar as perguntas.

Para entender melhor como as perguntas são interpretadas, considere o exemplo exibido na Figura 18. Neste exemplo, pode-se dizer que:

Figura 18 – Grafo de exemplo gerado manualmente no Canvas.



1. A informação `'q'` é verdadeira no estado `'x'`;
2. A partir do estado `'x'`, o agente `'a'` sabe que a informação `'p'` é verdadeira;
3. A partir do estado `'x'`, o agente `'a'` acredita que a informação `'r'` seja verdadeira.

O item 1 é comprovado após verificar que `'q'` está listado como uma das proposições verdadeiras para o estado `'x'`.

Antes de verificar o motivo que torna os itens subsequentes verdadeiros, é preciso considerar que um agente `'a'` só conhece transições que possuam `'a'` como agente, sendo todas as demais transições com outros agentes inexistentes para `'a'`. Em outras palavras, se torna impossível que `'a'`, partindo de um estado `'s1'`, conheça a existência de um estado `'s2'` se a transição entre esses estados não possui como um de seus possíveis agentes o agente `'a'`. Também é preciso recordar que arestas entre estados representam as dúvidas dos agentes com relação ao estado real do grafo.

Tendo isso em mente, o item 2 é verificado após percorrer todos os vizinhos de `'x'` pelo agente `'a'` (estados `'y'` e `'z'`) e encontrar em todos eles `'p'` como informação verdadeira. O item 3, por sua vez, se dá pelo fato de que existe pelo menos um vizinho de `'x'` por `'a'` que possui `'r'` como informação verdadeira.

Assim, tomando `'x'` como raiz do grafo, temos que as seguintes perguntas, equivalentes aos itens anteriormente listados, retornam `'true'` na interface do programa:

1. `'q'`;
2. `'p{*a}'`, que é interpretado de forma equivalente a $K_a p$ em notação epistêmica;
3. `'r{?a}'`, que é interpretado de forma equivalente a $B_a r$ em notação epistêmica.

No item 1, quando `'q'` é dado como entrada no campo 'Pergunta', internamente o programa gera a pilha `['q']` representando a entrada. Em seguida, o programa itera de trás para a frente sobre a pilha, que só possui um item neste caso, identificando se o símbolo apontado pelo índice atual é um operador ou não. No caso, `'q'` não é operador e, portanto, é interpretado como um símbolo proposicional. Sua valoração é dada buscando se ele está contido ou não no estado apontado como atual (no caso, o estado raiz). Como `'q'` está presente no estado `'x'`, o resultado da pergunta é *true*.

Para o item 2, o programa gera a pilha `['p','{','*','a','}']`, identifica o operador definido por `{*}` e verifica se a expressão `'p'` é verdadeira para cada estado vizinho de `'x'` por `'a'`. Como `'p'` é verdadeiro em todos os estados considerados (`'x'`, `'y'` e `'z'`), o programa gera uma nova pilha com esses resultados (`['true', 'true', 'true']`) e a passa para a função definida para aquele operador. No caso, a função do operador identificado (`'FOR EACH'`), percorre a pilha de valorações recebidas e checa se todos os itens são `'true'`. Caso sim, a função do operador retorna `'true'`. Como não resta mais nada a ser verificado na pilha que define a pergunta, o programa retorna `'true'`, que foi o resultado obtido.

O item 3 segue uma linha de execução semelhante ao do item 2: o operador `{?}` é identificado como o `'EXISTS'`. A expressão `'r'` é verificada em todos os estados vizinhos de `'x'`, resultando na pilha de valorações `['false','true','false']` que é passada para a função do operador. A função do `'EXISTS'` checa se existe pelo menos um item na pilha cujo valor é `'true'`, retornando ela mesma `'true'`. Com isso, chega-se ao fim da pilha da pergunta e, portanto, o programa retorna o resultado obtido: `'true'`.

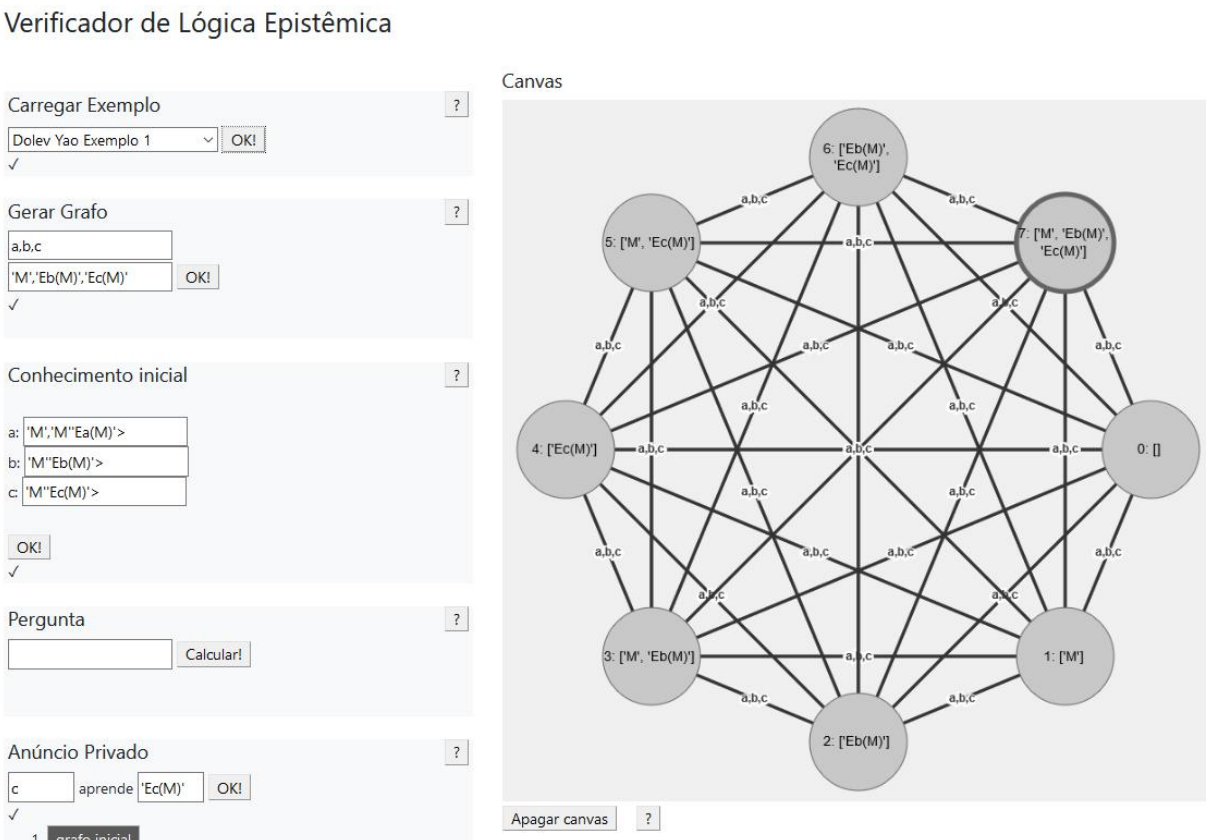
3.6 CARREGANDO EXEMPLOS

Por facilidade, alguns modelos epistêmicos desenvolvidos no programa já tiveram suas execuções completamente mapeadas e podem ser facilmente carregados na interface. Na seção `'Carregar Exemplo'`, é possível selecionar um dos modelos já pré-fabricados.

Uma vez que um dos modelos é carregado, todos os passos da execução dele são automaticamente computados e populados na interface. Um exemplo é mostrado na Figura 19. Pode-se observar na figura que o exemplo `'Dolev-Yao Exemplo 1'` foi carregado na interface e já possui toda a linha de execução ideal mapeada, com o conhecimento inicial e anúncios privados já computados.

Quando um exemplo é carregado, a interface sempre exibe o grafo inicial, sem ainda aplicar os efeitos de computar conhecimentos iniciais ou anúncios. Basta clicar nos passos

Figura 19 – Exemplo `'Dolev-Yao Exemplo 1'` carregado na interface do programa.



exibidos no histórico do programa (logo após a seção `'Anúncio Privado'`) para ver as modificações aplicadas ao grafo a cada passo.

4 MODELAGENS REALIZADAS

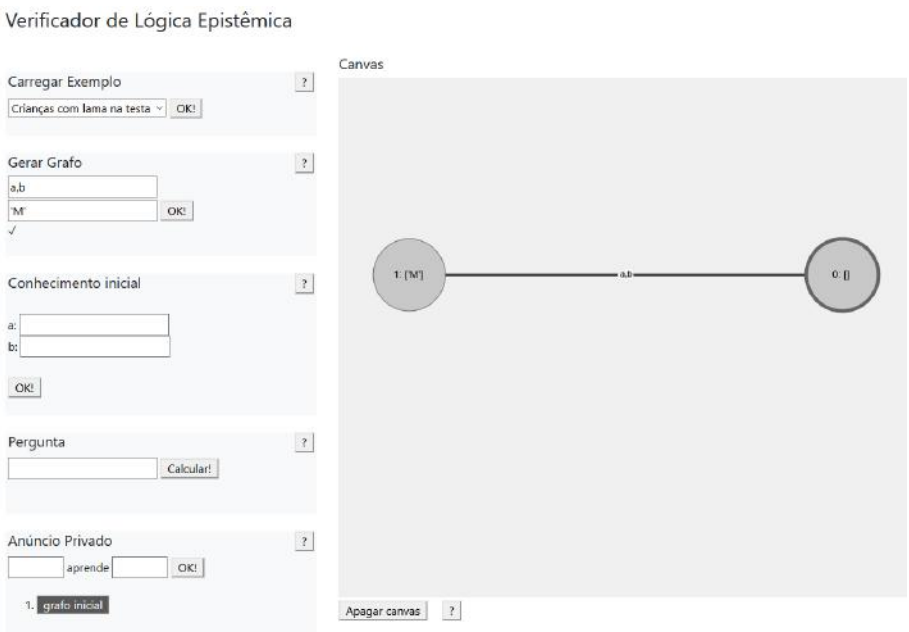
Este capítulo visa a apresentar os modelos implementados para a análise do trabalho. Na Seção 4.1, um exemplo simples de modelagem é apresentado, onde dois agentes trocam uma mensagem em texto claro. As três seções seguintes apresentam exemplos do trabalho de (DOLEV; YAO, 1983): a Seção 4.2 apresenta o exemplo 1, a Seção 4.3 o exemplo 2 e a Seção 4.4 o exemplo 3. Finalmente, a Seção 4.5 apresenta uma possível modelagem para o problema lógico das crianças com lama na testa.

4.1 EXEMPLO SIMPLES DE COMUNICAÇÃO ENTRE DOIS AGENTES

Antes de que seja feito um aprofundamento nas modelagens com troca de mensagens com chave criptográfica, o exemplo mais simples possível será considerado, onde uma mensagem é enviada de um agente para outro sem qualquer tipo de criptografia envolvida, ou seja, a mensagem é transmitida em texto claro. Também será considerado que a mensagem não vem a ser interceptada por um agente intruso na comunicação.

Assim, um possível grafo demonstrando esse cenário poderia ser o mostrado na Figura 20. Esse grafo pode ser gerado manualmente na interface do programa, visto sua simplicidade, mas também pode ser gerado automaticamente passando `'a'` e `'b'` como agentes e `'M'` como proposição.

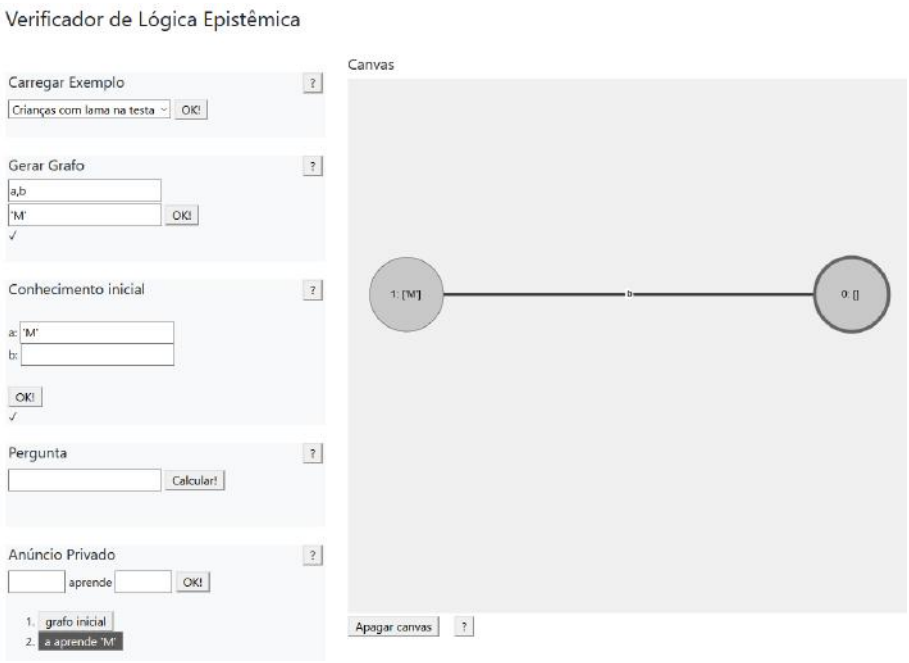
Figura 20 – Exemplo de modelagem simples para a comunicação entre dois agente sem criptografia.



Definindo o agente `'a'` como remetente e `'b'` como destinatário da mensagem `'M'`,

pode-se passar essa informação ao programa no formato de conhecimento inicial. Isto é, pode-se inserir `'M'` no campo `'Conhecimento Inicial'` da interface para o agente `'a'`, como mostrado na Figura 21.

Figura 21 – Apenas `'b'` permanece com dúvida após `'a'` ser definido como remetente da mensagem.



Neste momento, pode-se confirmar que apenas `'a'` tem conhecimento sobre `'M'` inserindo os questionamentos `'M'{*a}` e `'M'{*b}` no campo `'Pergunta'` da interface. Definindo o estado que possui `'M'` como o estado raiz do grafo (estado `'1'` na Figura 22), a pergunta `'M'{*a}` retorna `'true'` pois `'a'` é remetente e `'M'{*b}` retorna `'false'` pois `'b'` ainda não sabe o conteúdo da mensagem.

Para simular o envio da mensagem `'M'` de `'a'` para `'b'`, é preciso informar ao programa que `'b'` recebeu essa nova informação. Isso é feito no campo `'Anúncio Privado'` da interface, dizendo que `'b'` aprende `'M'`. Lembrando que `'b'` aprende `'M'` diretamente apenas porque o cenário é de uma comunicação sem criptografia. O resultado é que todas as arestas do grafo sumiram pois todas as dúvidas foram extintas, ou seja, tanto `'b'` quanto `'a'` agora sabem o conteúdo da mensagem `'M'`. Isso pode ser comprovado perguntando `'M'{*a}'M'{*b}^` ao programa enquanto o estado que possui `'M'` é a raiz, encerrando a comunicação entre os dois agentes. Esse resultado é mostrado na Figura 23.

4.2 EXEMPLO 1 DOLEV YAO

Também foi feita para o trabalho a modelagem do exemplo 1 do trabalho (DOLEV; YAO, 1983). A ideia é simular exatamente o fluxo de troca de mensagens entre os agentes

Figura 22 – A pergunta `'M'{*b}` retorna `'false'` no grafo atual.

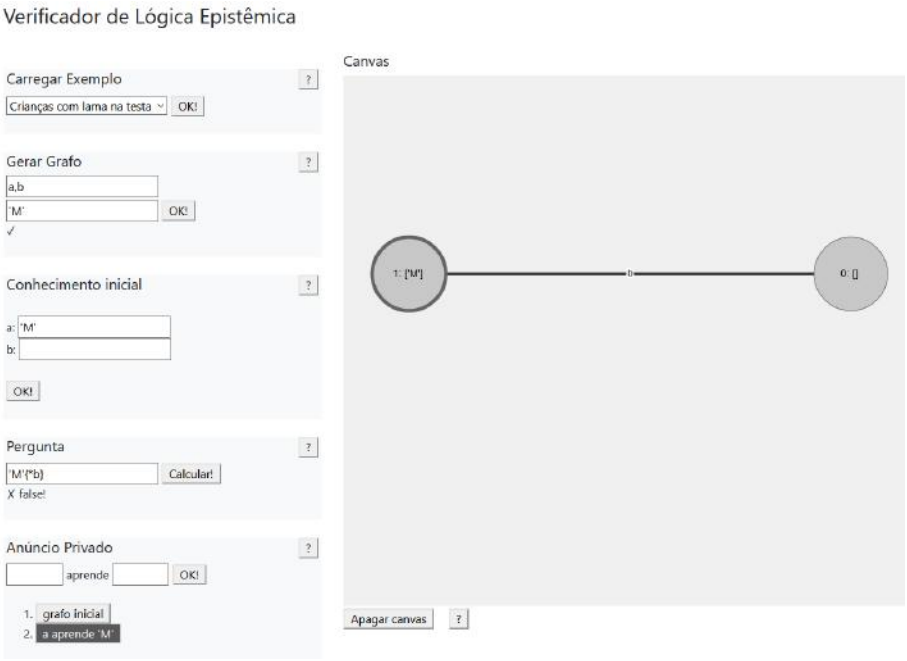
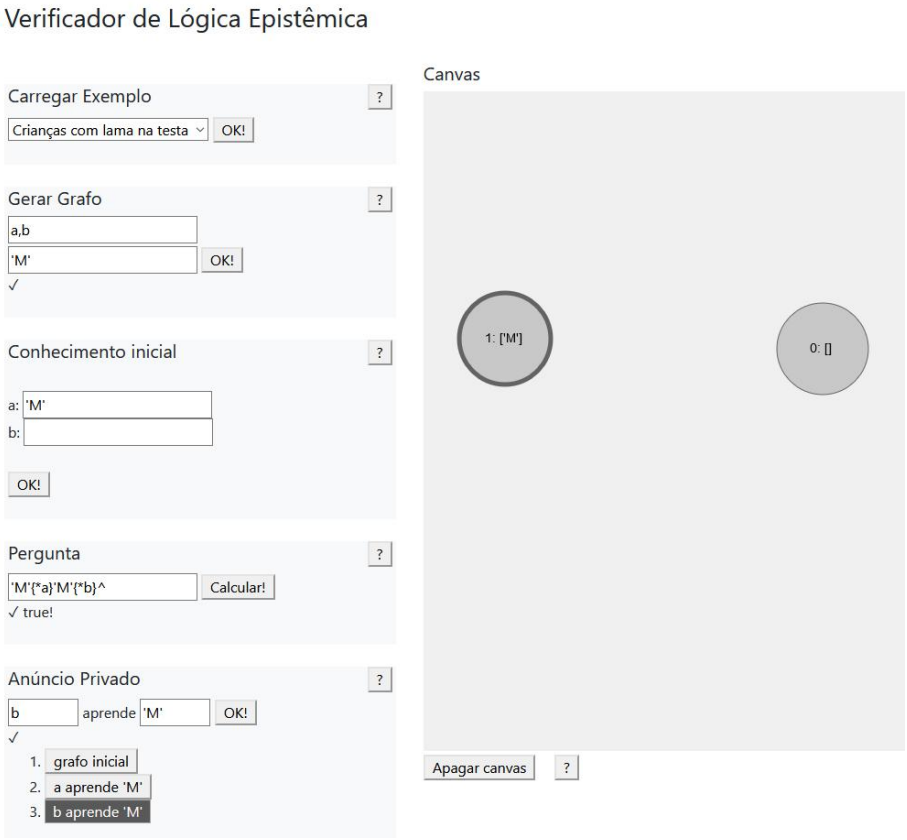


Figura 23 – O agente `'b'` recebeu `'M'` de `'a'` e agora sabe o conteúdo da mensagem.



indicados, considerando o modelo de criptografia utilizado para este exemplo, e avaliar se os resultados conferem com o que é demonstrado no artigo. Isto é, se o intruso obtém ou

não acesso à mensagem original trafegada.

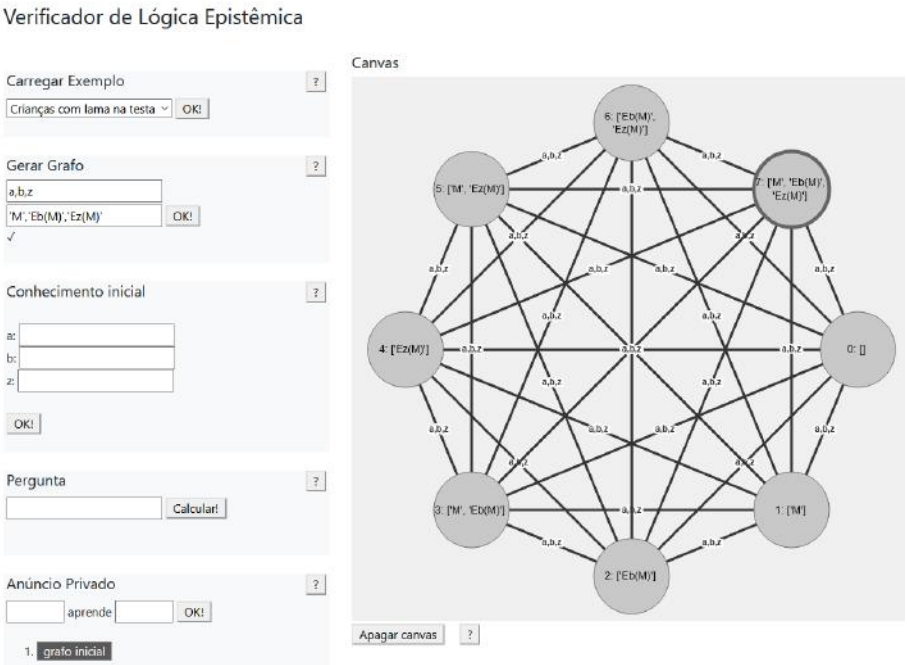
O fluxo de troca de mensagens foi descrito em detalhes na subseção 2.3.1. Neste exemplo, um agente `'a'` deseja enviar uma mensagem a `'b'`, porém um agente terceiro `'z'` faz uma interceptação. A mensagem trafegada é criptografada utilizando a chave pública do destinatário (DOLEV; YAO, 1983).

Este exemplo pode ser diretamente carregado na interface através da funcionalidade `'Carregar Exemplo'`, porém a construção do modelo e fluxo de troca das mensagens será descrito aqui passo a passo para melhor entendimento.

Um grafo descrevendo o cenário inicial pode ser definido passando `'a'`, `'b'` e `'z'` como agentes e `'M'`, `'Eb(M)'` e `'Ez(M)'` como proposições. O resultado é mostrado na Figura 24. Também é definido que o estado `'7'` é o estado raiz do grafo. Isso se dá pois este é o estado mais completo de informações e, como será visto ao longo da execução, o agente intruso virá a tomar conhecimento sobre cada uma das proposições nele indicadas, tornando este estado uma boa raiz para que sejam feitas perguntas ao fim do fluxo.

É interessante notar que, como todos os estados possuem relações reflexivas ocultas, o estado raiz escolhido deve pelo menos conter a mensagem descriptografada `'M'`, já que se deseja checar o comprometimento dela por agentes intrusos e a pergunta realizada ao grafo é avaliada partindo sempre do estado definido como raiz. Caso o estado `'0'` fosse definido como raiz, por exemplo, não importa quantos anúncios fossem feitos, a pergunta "Z já sabe M?" sempre retornaria *false* pois o próprio estado raiz não possui `'M'` como verdadeiro.

Figura 24 – Exemplo de grafo inicial para o exemplo 1 de Dolev-Yao.

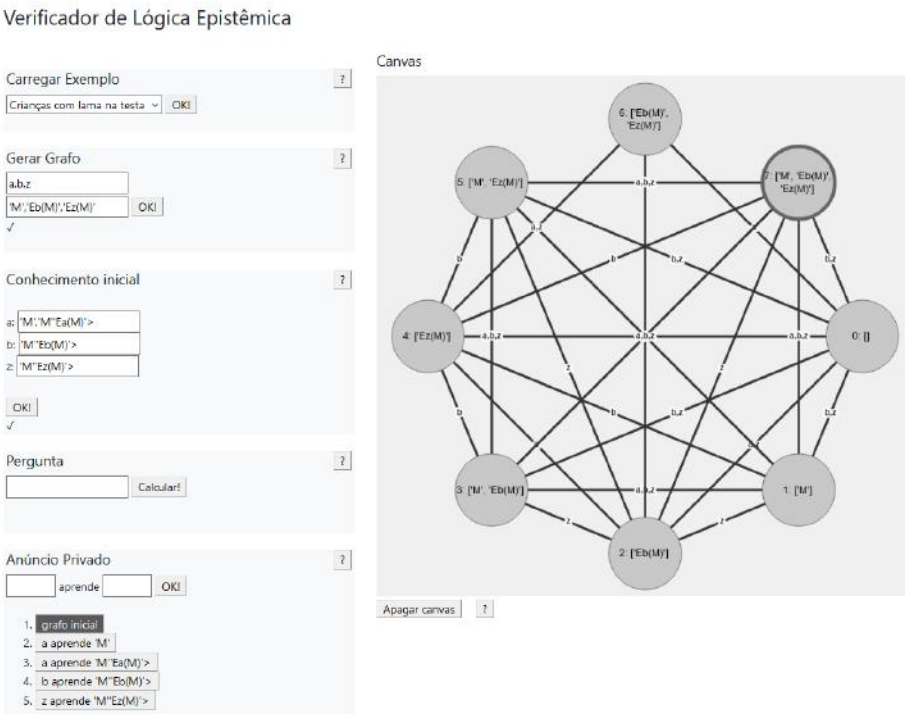


Além disso, também é importante comentar que, tanto para esse exemplo quanto

para os demais de Dolev-Yao, algumas simplificações estão sendo feitas para um melhor entendimento e visualização do grafo na interface. Para este exemplo, somente foram passadas as proposições `'M'`, `'Eb(M)'` e `'Ez(M)'` pois este é o mínimo de proposições necessárias para replicar o fluxo de troca de mensagens definido no em (DOLEV; YAO, 1983) e obter o mesmo resultado. É por este motivo que, portanto, a informação `'Ea(M)'` não chega a ser definida como proposição para esse exemplo.

Na sequência, pode-se também definir alguns conhecimentos iniciais para os três agentes. Cada um dos agentes sabe que, se obter a mensagem encriptada através de sua própria chave pública, pode obter a mensagem em texto claro utilizando sua chave privada. Utilizando a notação lógica, isso significa dizer que, para cada agente x, se x aprende $E_x(M)$ então x aprende `'M'`. Na interface, isso é escrito como `'M'Ex(M)'>` para cada agente `'x'`. Além disso, o agente `'a'` também possui `'M'` como conhecimento inicial pois ele é o autor da mensagem na modelagem. Esses passos podem ser observados na Figura 25.

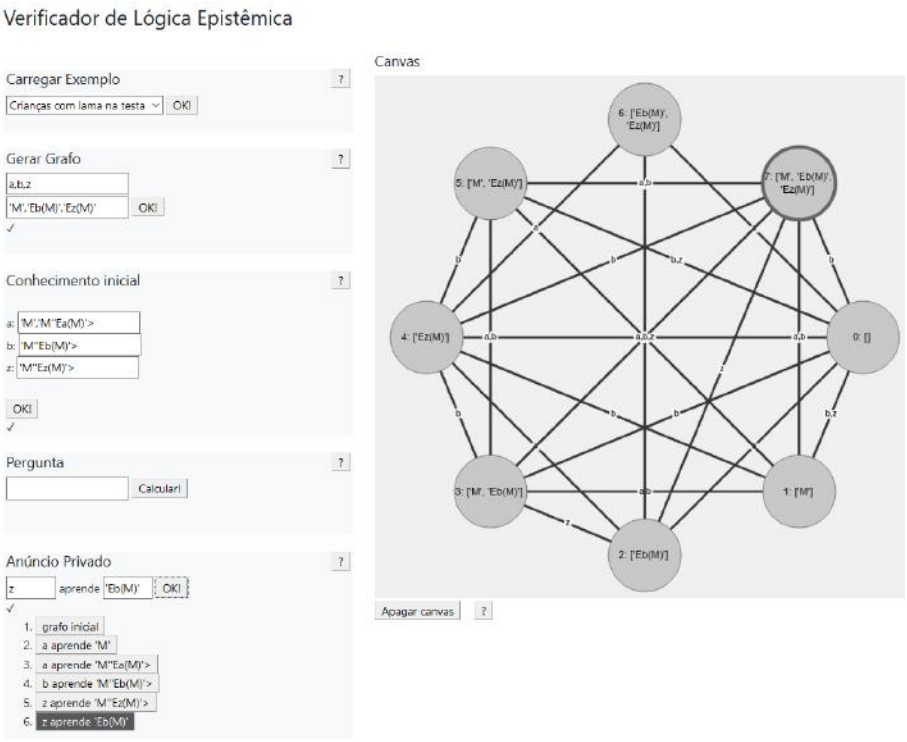
Figura 25 – Grafo após definição dos conhecimentos iniciais dos agentes.



Agora, o programa está pronto para simular a troca de mensagens entre os agentes. No exemplo 1 em (DOLEV; YAO, 1983), o agente `'a'` envia a mensagem `'Eb(M)'` para `'b'`, porém o intruso `'z'` intercepta o envio. Isso significa dizer para o programa que `'z' aprende 'Eb(M)'`. Esse passo é descrito na Figura 26.

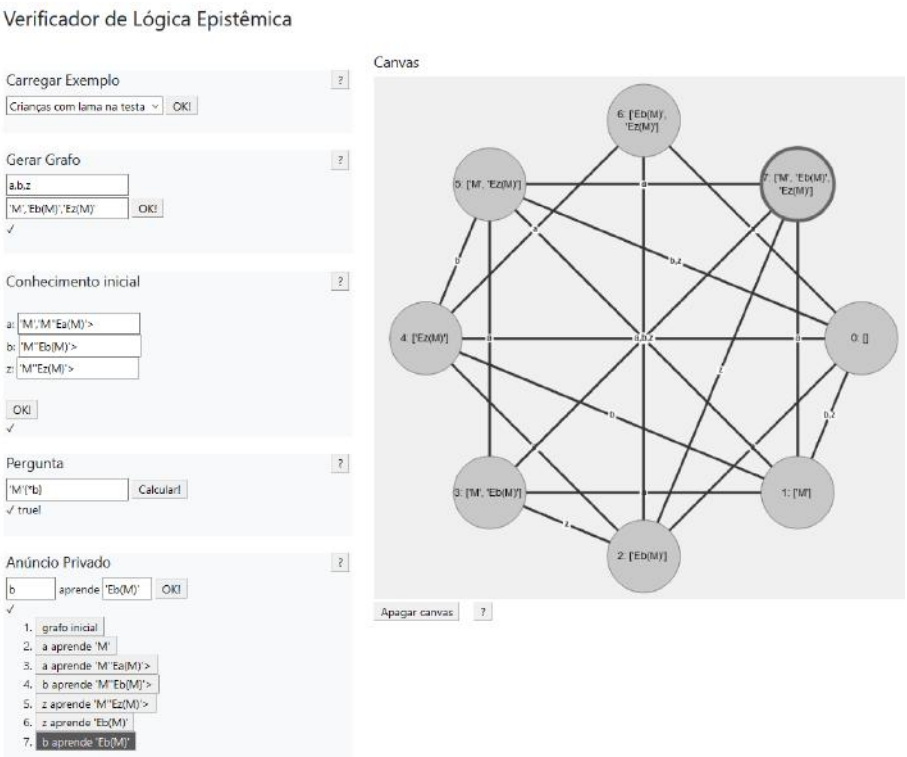
No passo seguinte, o intruso `'z'` encaminha a mensagem interceptada para o agente `'b'`, se passando pelo remetente e autor da mensagem. Isso é denotado no programa como `'b' aprende 'Eb(M)'`. É interessante também notar que, a partir deste ponto, se

Figura 26 – O agente intruso `'z'` aprende o conteúdo da mensagem interceptada.



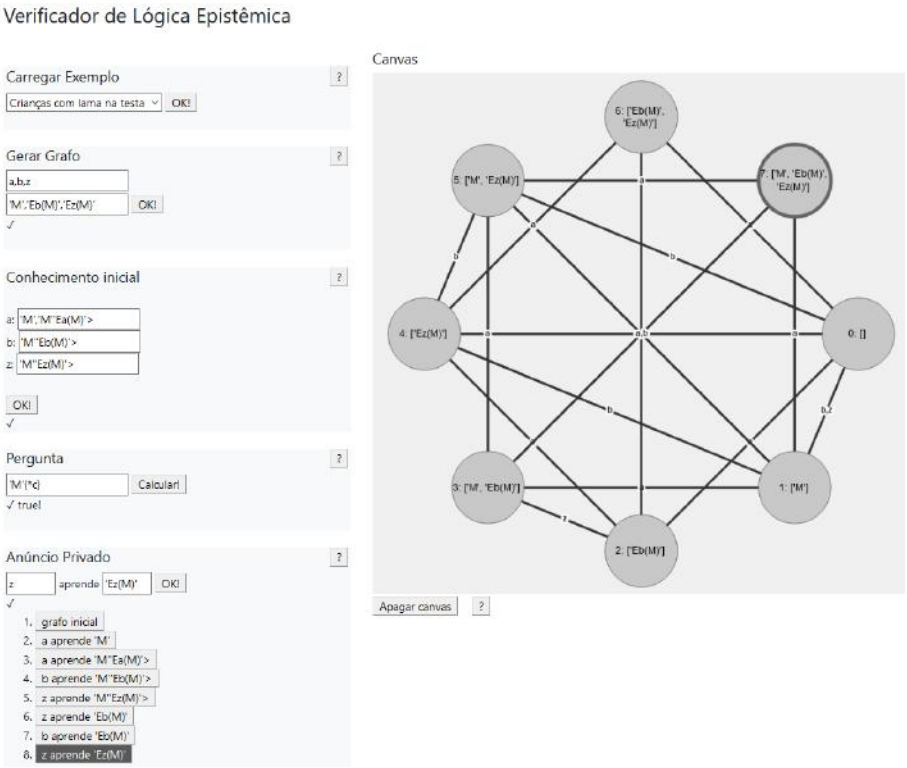
for questionado ao programa se `'b'` sabe `'M'`, o retorno será `'true'`, já que `'b'` consegue descriptografar o que recebeu. Isso é mostrado na Figura 27.

Figura 27 – Agente `'b'` recebe `'Eb(M)'` de `'z'` e lê mensagem original `'M'`.



Como o agente 'b' não sabe que 'z' é um intruso na comunicação, ele responde 'z' normalmente acreditando que ele é o autor original. Dessa forma, 'b' envia 'Ez(M)' para 'z'. Porém, como 'z' é capaz de descriptografar 'Ez(M)', ele aprende 'M', como comprovado pelo resultado da pergunta feita ao grafo mostrada na Figura 28. Isso significa que esse modelo de protocolo criptográfico é quebrável por um agente intruso na comunicação, mesmo que as mensagens estejam criptografadas enquanto são trocadas pelos agentes.

Figura 28 – O intruso 'z' obtêm acesso à mensagem original 'M', quebrando a privacidade do protocolo de comunicação utilizado.



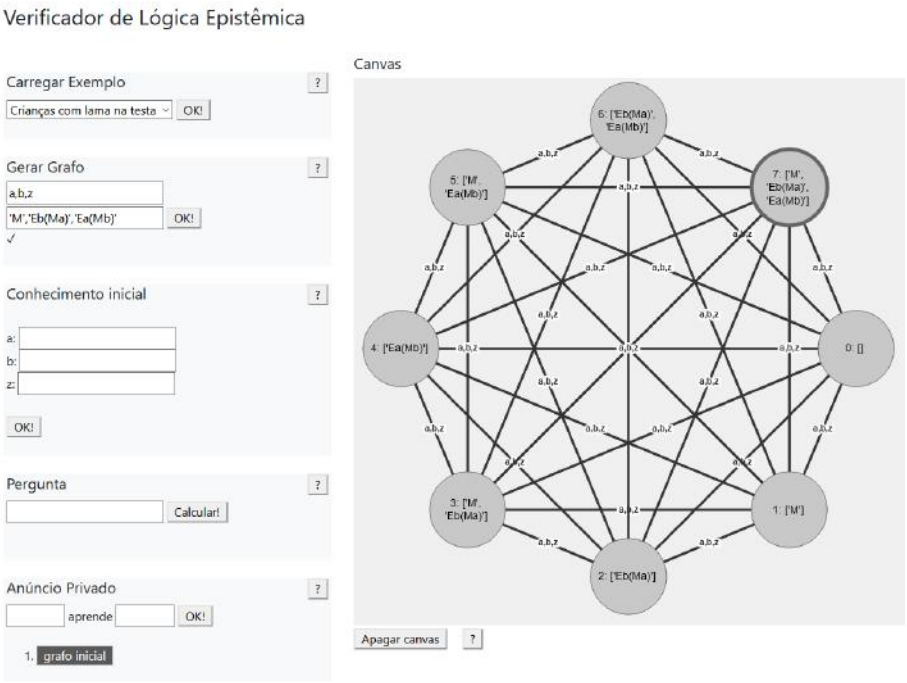
4.3 EXEMPLO 2 DOLEV YAO

No exemplo 2 de (DOLEV; YAO, 1983), descrito em detalhes na subseção 2.3.2, o formato da mensagem trafegada entre agentes é diferente. Agora, o autor original é concatenado junto ao conteúdo da mensagem e o resultado dessa concatenação é criptografado e enviado. O agente que recebe a mensagem, nesse cenário, responderá uma mensagem criptografada não com a chave pública de quem lhe fez o envio, mas sim com a chave pública do autor original cujo nome está concatenado à mensagem (DOLEV; YAO, 1983).

Para modelar esse exemplo, um grafo muito semelhante ao da seção anterior é gerado automaticamente, com os agentes 'a', 'b' e 'z' e proposições 'M', 'Eb(Ma)' e 'Ea(Mb)'.

Pode-se observar que agora as mensagens trafegadas são compostas da concatenação de `'M'` com algum dos agentes. No caso, `'Eb(Ma)'` representa a mensagem que `'a'` deseja enviar a `'b'`, pois foi criptografada utilizando a chave pública de `'b'`, e possui `'a'` como autor original, pois `'M'` está concatenado com `'a'`. O grafo gerado com esses valores é mostrado na Figura 29. Para esse exemplo, também será utilizado o estado `'7'` como raiz do grafo, por ser o mais completo de informações.

Figura 29 – Grafo gerado para modelar o exemplo 2 de (DOLEV; YAO, 1983).



Assim como no exemplo 1, não estão sendo consideradas todas as combinações possíveis de mensagens encriptadas com agentes pois foram definidas as proposições mínimas necessárias para se replicar o fluxo do exemplo 2. Por exemplo, `'Eb(Ma)'` é passado como uma das proposições nesta modelagem, mas `'Eb(Mz)'` não. Isso se dá porque `'Eb(Mz)'` não chega a ser trafegado entre os agentes em nenhum momento do fluxo mínimo descrito em (DOLEV; YAO, 1983).

Na sequência, os conhecimentos iniciais dos agentes são passados para o programa (Figura 30). É com base nas expressões passadas que os agentes serão capazes de descriptografar ou não as mensagens que recebem. Mais uma vez, `'a'` é definido como o emissor inicial de `'M'` e, portanto, possui `'M'` definido como um de seus conhecimentos iniciais.

Pode-se agora iniciar o fluxo de mensagens. Este exemplo, assim como o anterior, se inicia com o agente `'a'` enviando uma mensagem ao agente `'b'`. Essa mensagem segue o formato `'Eb(Ma)'`, já que `'b'` é o destinatário e `'a'` é o autor de `'M'`. Entretanto, ocorre uma interceptação pelo agente `'z'`. Isso é mostrado na Figura 31.

O intruso `'z'` reencaminha o que recebeu de `'a'` para `'b'`. É importante notar que `'z'` não tem como alterar o agente que foi concatenado a `'M'` pois a mensagem

Figura 30 – Conhecimento inicial necessário para que os agentes descriptografem mensagens recebidas.

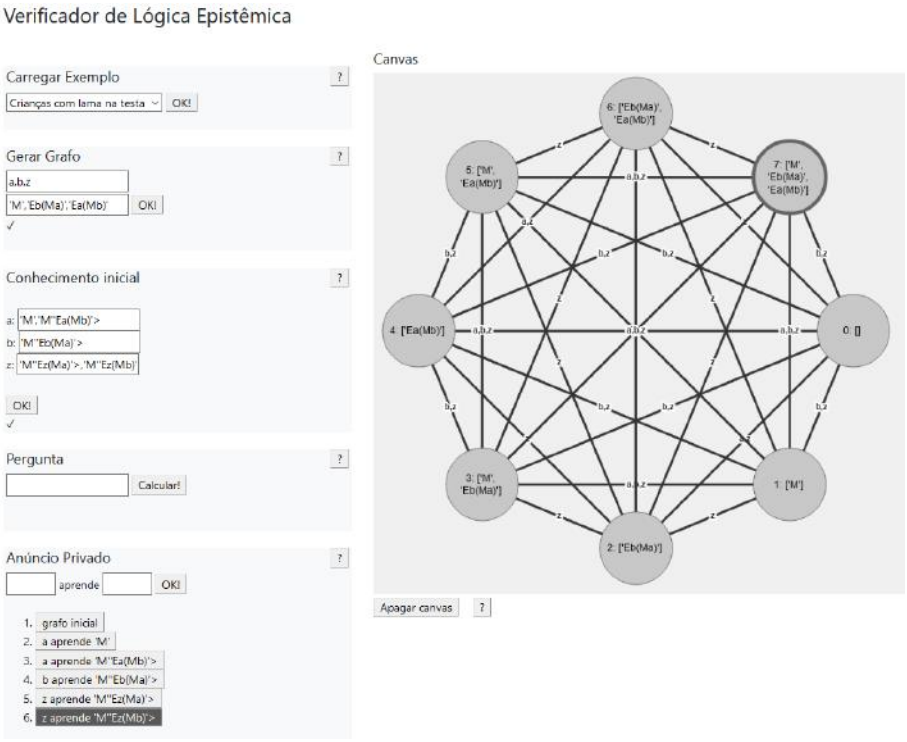
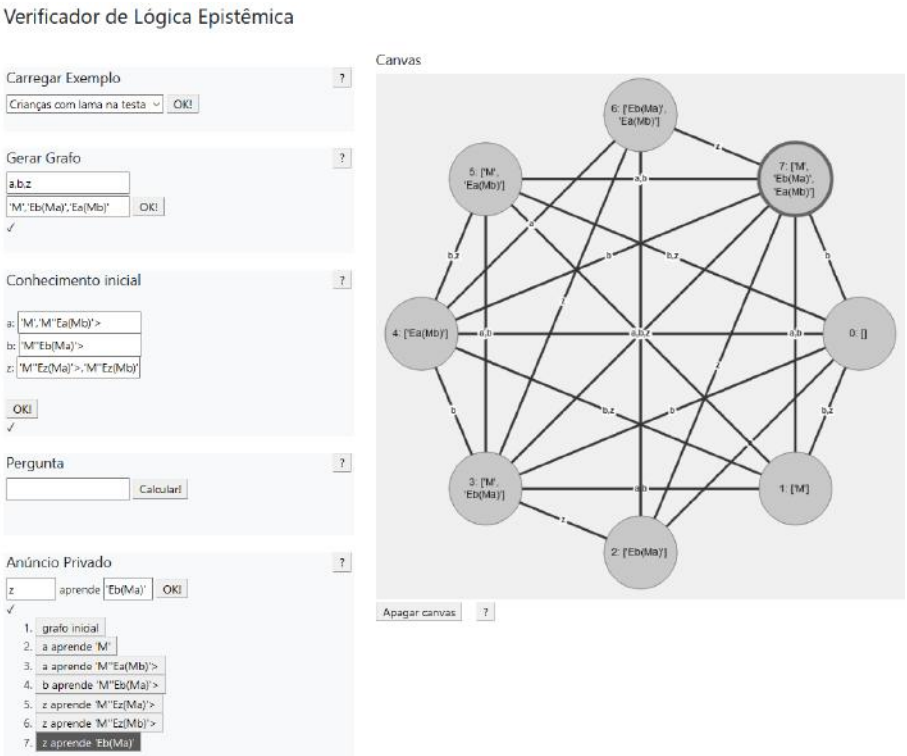
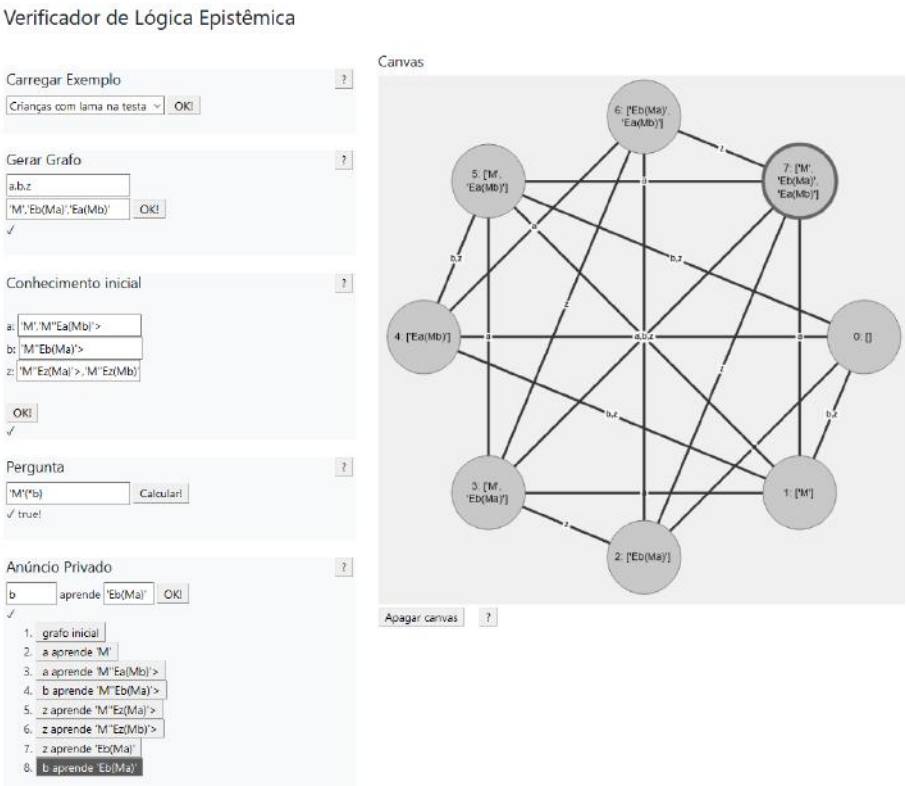


Figura 31 – Intruso `'z'` intercepta mensagem trafegada de `'a'` para `'b'`.



está criptografada com a chave pública de `'b'` e ele não tem conhecimento sobre ela. Assim, como não lhe resta opção, ele encaminha exatamente `'Eb(Ma)'` ao agente `'b'`, o que é representado no programa computando o anúncio privado `'b' aprende 'Eb(Ma)'`. Consequentemente, `'b'` aprende `'M'` pois consegue descriptografar o que recebeu (Figura 32).

Figura 32 – Agente `'b'` aprende a informação `'Eb(Ma)'` e, consequentemente, `'M'`.



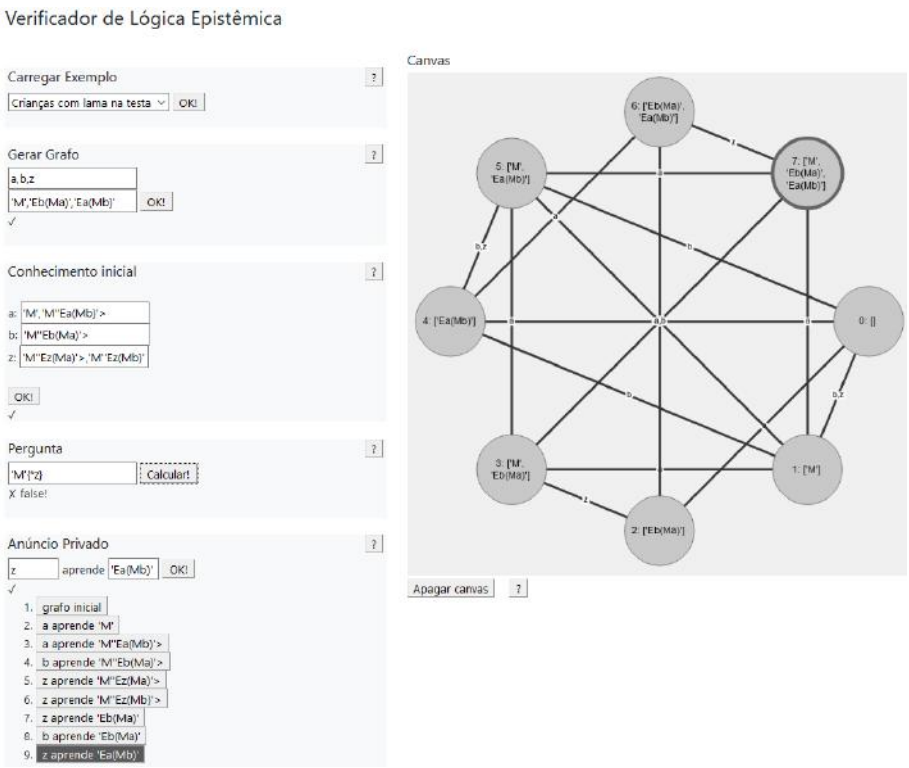
Em seguida, o agente `'b'` responde `'z'`, porém ele criptografa a mensagem enviada não com a chave pública de quem fez o envio, mas sim com a chave pública do autor original cuja identificação estava concatenada a `'M'`. No caso, em `'Eb(Ma)'`, que foi o que `'b'` recebeu, esse autor original é o agente `'a'` e, portanto, a resposta de `'b'` para `'z'` é criptografada com a chave pública de `'a'`. No programa, isso é representado através do anúncio `'z' aprende 'Ea(Mb)'`.

Como `'z'` não consegue descriptografar o que recebeu, o fluxo termina e o protocolo se mostra confiável, pois não foi possível quebrar a privacidade da comunicação entre os agentes. Após o último anúncio realizado, ao perguntar ao programa se `'z'` sabe `'M'`, a resposta exibida é `'false'` (Figura 33).

4.4 EXEMPLO 3 DOLEV YAO

O exemplo 3 de (DOLEV; YAO, 1983), descrito em detalhes na subseção 2.3.3, é uma tentativa de aprimorar ainda mais a segurança do protocolo, adicionando uma camada a

Figura 33 – Agente `'z'` aprende a informação `'Ea(Mb)'`, porém não consegue descriptografá-la nem obter conhecimento sobre `'M'`.



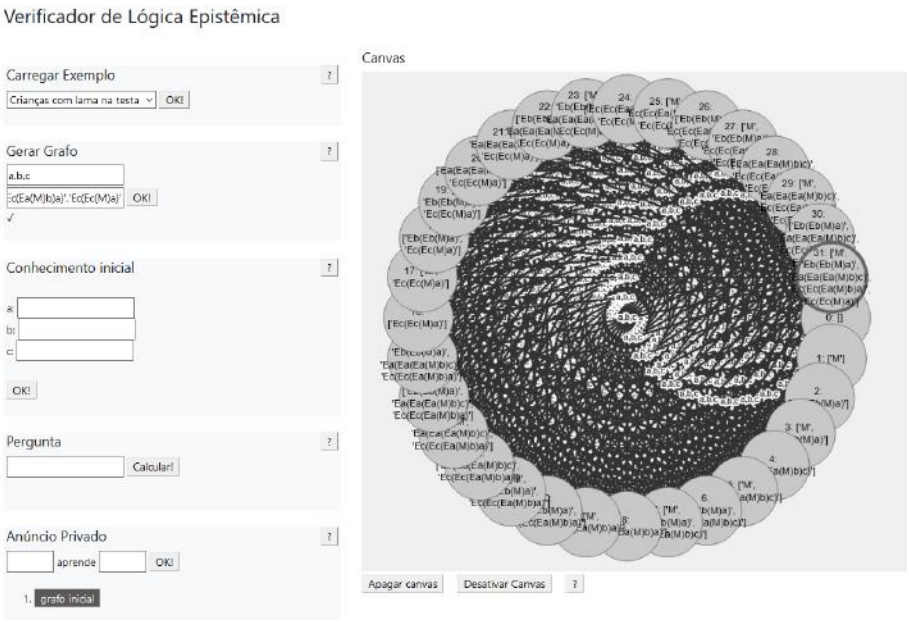
mais de criptografia. Entretanto, como será visto, isso não impede que ele seja quebrado. Neste exemplo, se `'a'` envia uma mensagem a `'b'`, ela terá o formato `'Eb(Eb(M)a)'`. Ou seja, o autor da mensagem é concatenado ao conteúdo dela somente na segunda vez que se criptografa a mensagem (DOLEV; YAO, 1983).

Para modelar esse exemplo, cria-se um grafo passando os agentes `'a'`, `'b'` e `'z'` e proposições `'M'`, `'Eb(Eb(M)a)'`, `'Ea(Ea(Ea(M)b)z)'`, `'Ez(Ez(Ea(M)b)a)'` e `'Ez(Ez(M)a)'`. Novamente, por motivos de simplificação do grafo, foi passado ao programa o mínimo de proposições necessárias utilizadas no fluxo descrito para este exemplo. O grafo resultante é bem grande, como pode ser visto na Figura 34. O estado de maior ID presente (no caso, o estado `'31'`) é novamente o mais completo de informações e será definido como a raiz do grafo. Para os demais passos, usaremos a funcionalidade `'Desativar Canvas'`, que oculta o grafo, para melhorar a performance da interface. Os anúncios e validações poderão ser feitos normalmente dessa forma.

Alguns conhecimentos iniciais podem ser definidos baseados nas habilidades de cada agente descriptografar mensagens recebidas criptografadas com suas chaves publicas. Para esse exemplo, os conhecimentos de `'a'`, `'b'` e `'z'` são definidos como:

- a: `'M',’M’'Ea(Ea(Ea(M)b)z)'>`, que equivale a M e $Ea(Ea(Ea(M)b)z) \rightarrow M$;
- b: `'M’'Eb(Eb(M)a)'>`, que equivale a $Eb(Eb(M)a) \rightarrow M$;

Figura 34 – Grafo gerado para modelar o exemplo 3 do trabalho de Dolev-Yao.



- z: `'Ea(M)''Ez(Ez(Ea(M)b)a)'>,'M''Ez(Ez(M)a)'>`, que equivale a $Ez(Ez(Ea(M)b)a) \rightarrow Ea(M)$ e $Ez(Ez(M)a) \rightarrow M$.

Agora pode-se simular a troca de mensagens entre os agentes. A execução começa com `'a'` enviando uma mensagem ao agente `'b'`, o que resulta no anúncio `'b' aprende 'Eb(Eb(M)a)'`. Por consequência, `'b'` aprende `'M'`, o que pode ser comprovado na Figura 35.

O agente `'b'` deseja responder `'a'` e envia uma mensagem, porém essa mensagem é interceptada pelo agente intruso `'z'`. Isso é denotado através do anúncio `'z' aprende 'Ea(Ea(M)b)'`. Como `'z'` não consegue imediatamente acessar o conteúdo da mensagem, ele criptografa o conteúdo interceptado com a chave pública de `'a'` e encaminha essa mensagem para ele (no programa, `'a' aprende 'Ea(Ea(Ea(M)b)z)'`). O agente `'a'` então responde normalmente `'z'`, o que resulta no anúncio `'z' aprende 'Ez(Ez(Ea(M)b)a)'`. Agora, `'z'` consegue descriptografar parte da mensagem e obtêm `'Ea(M)'`, como já definido nos conhecimentos iniciais. O intruso envia uma nova mensagem ao agente `'a'` com o conteúdo `'Ea(Ea(M)z)'`, o que é denotado no programa pelo anúncio `'a' aprende 'Ea(Ea(M)z)'`. Quando `'a'` responde, ele criptografa a mensagem com a chave pública de `'z'`, resultando no anúncio `'z' aprende 'Ez(Ez(M)a)'` que faz com que o intruso obtenha `'M'` e torna o protocolo de comunicação quebrável e inseguro.

Ao final de todos os anúncios descritos, pode-se perguntar ao programa se `'z'` sabe `'M'`, e será obtido `'true'` como resposta. Isso é mostrado na Figura 36.

Para os três exemplos aqui modelados derivados do trabalho de (DOLEV; YAO, 1983), os mesmos resultados foram obtidos. Os exemplos 1 e 3 apresentam protocolos inseguros

Figura 35 – Agente `'a'` envia uma mensagem a `'b'`.

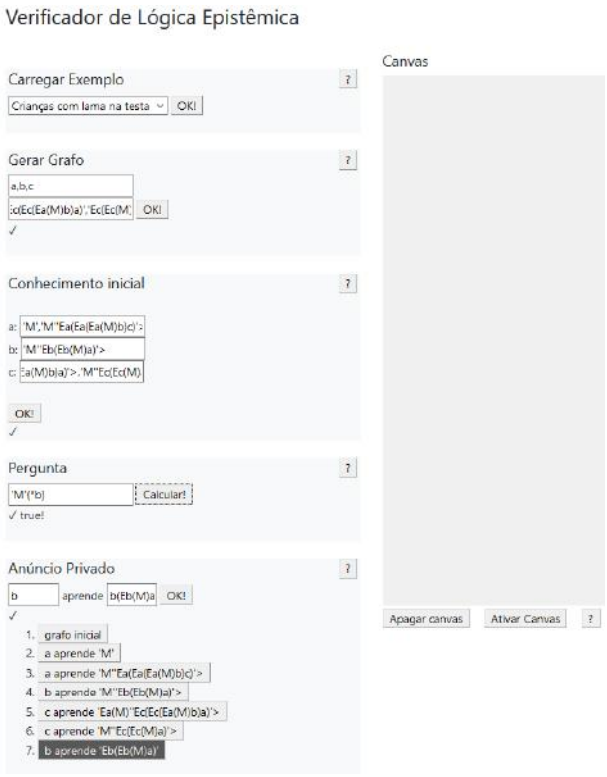
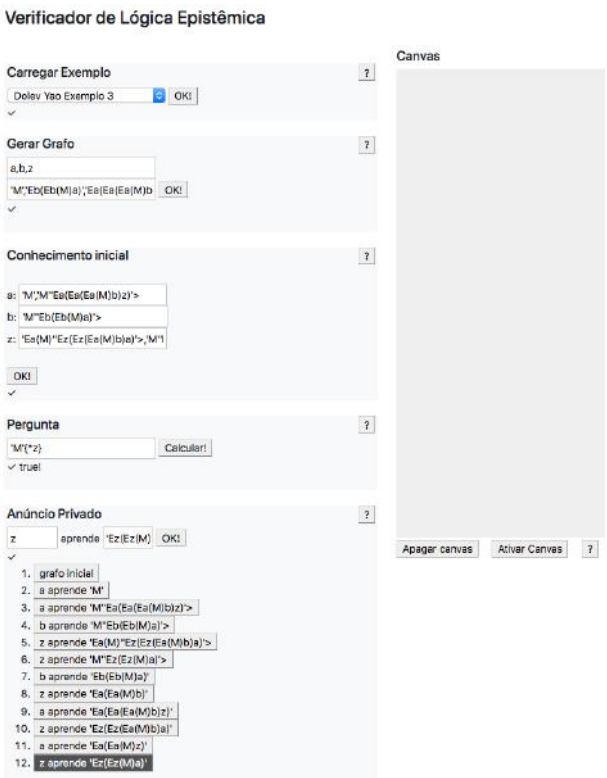


Figura 36 – Resultado obtido para o exemplo 3, onde o intruso consegue descobrir `'M'`.



e o exemplo 2, um protocolo seguro. Essas implementações visam demonstrar a corretude do trabalho, de forma que, para modelagens futuras de outros protocolos além destes, seja possível representar a realidade através do programa implementado (DOLEV; YAO, 1983).

4.5 CRIANÇAS COM LAMA NA TESTA

O problema das crianças com lama na testa é um conhecido problema lógico que pode ser representado e solucionado nos moldes da lógica epistêmica. Para o trabalho, ele é um exemplo que busca exemplificar que as funcionalidades desenvolvidas para o programa extrapolam as propostas inicias de modelar e avaliar a segurança em protocolos de autenticação. O programa se mostra também preparado para modelar e resolver problemas mais gerais como o apresentado nesta seção.

Neste problema, um certo número de crianças estão brincando. Algumas possuem a testa suja de lama, outras não. Nenhuma criança sabe se sua própria testa está suja de lama ou não inicialmente, mas ela pode observar todos os colegas ao redor e sabe quais deles têm ou não a testa suja de lama. Além das crianças, existe a presença de outro agente externo, sendo este representado comumente como pai ou algum responsável pelas crianças brincando. Esse agente externo fará uma série de anúncios que serão ouvidos por todas as crianças presentes, a fim de auxiliá-las na descoberta por quem tem ou não lama na testa.

Considerando apenas três crianças brincando e supondo que seus nomes sejam Ana, Beto e Carla, é possível gerar um grafo na interface para representar esse problema passando como agentes os valores `'a'`, `'b'` e `'c'` e, como proposições, os valores `'ana'`, `'beto'`, `'carla'`. O resultado é mostrado na Figura 37. Assim, caso um estado do grafo tenha nele as informações `'ana'` e `'beto'`, é dito que Ana e Beto possuem lama na testa e Carla não possui lama na testa neste estado.

O estado `'3'` será considerado o estado raiz para esse fluxo, mas qualquer outro estado poderia ser escolhido. O estado raiz será considerado o estado real do problema, isto é, ele representa a realidade e, neste jogo, Ana e Beto realmente são os únicos com lama na testa. O objetivo é tornar todos os agentes cientes sobre o estado real do jogo. Os conhecimentos iniciais definidos na Figura 38 são consequência de qual é o estado real considerado, já que cada agente possui conhecimento sobre a testa dos outros mas não sobre a de si mesmo. Também foi feita uma arrumação manual no Canvas para posicionar os estados no formato de um cubo para obter uma visualização semelhante à apresentada na subseção 2.1.4.

O agente externo responsável pelas crianças faz um anúncio que é ouvido por todas. Ele diz que existe pelo menos uma criança presente que tem lama na testa. Em notação lógica, esse é um anúncio público escrito como $ana \vee beto \vee carla$. No programa, um

Figura 37 – Grafo gerado automaticamente pelo programa com os agentes `'a'`, `'b'` e `'c'` e proposições `'ana'`, `'beto'`, `'carla'`.

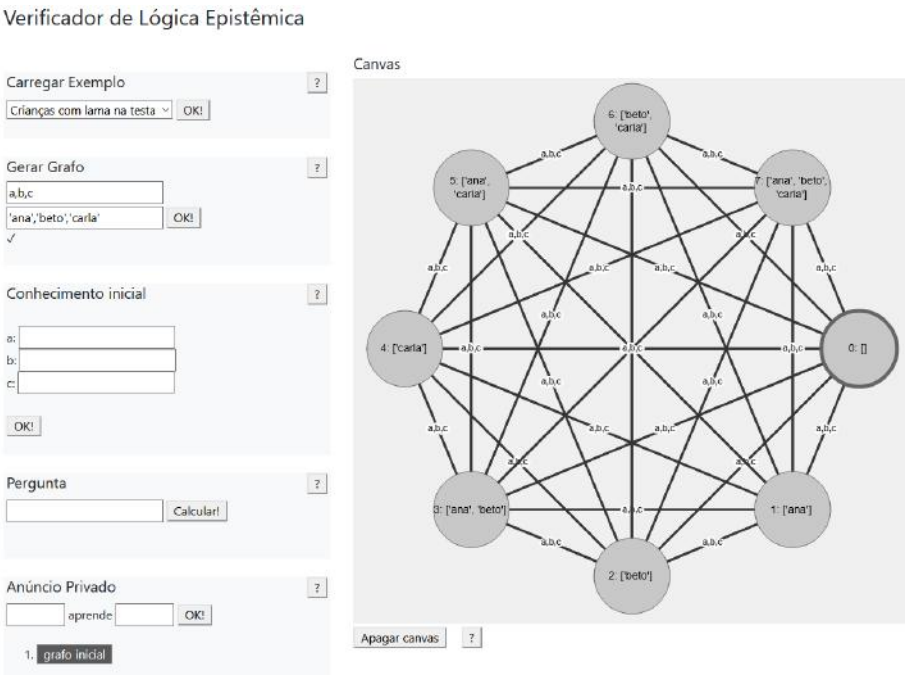
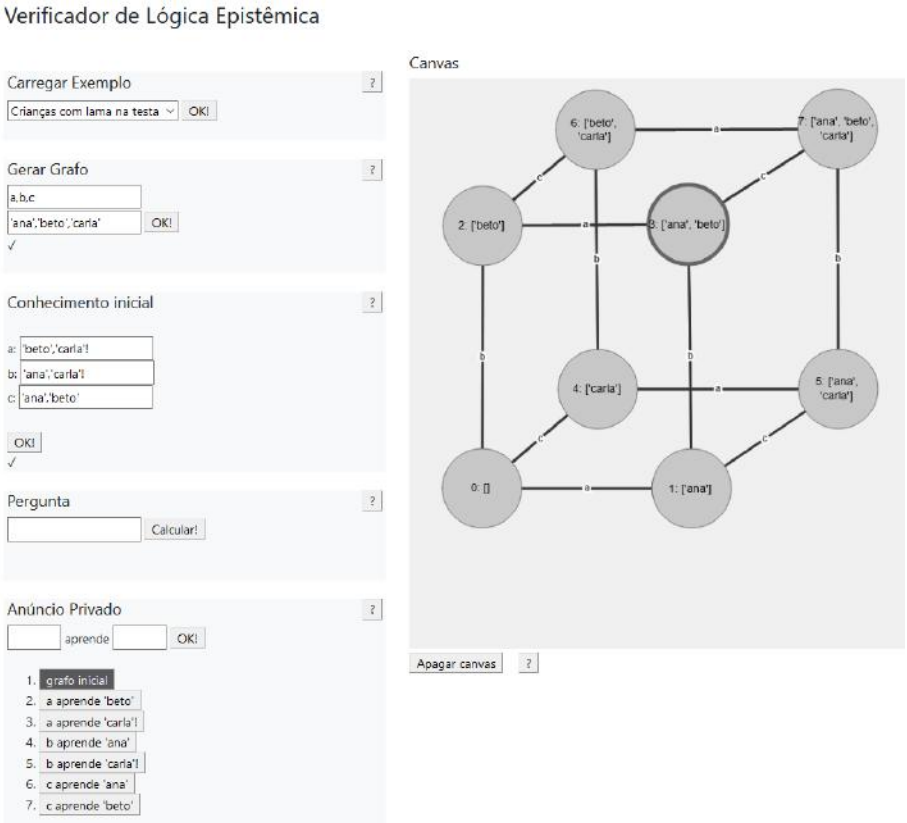
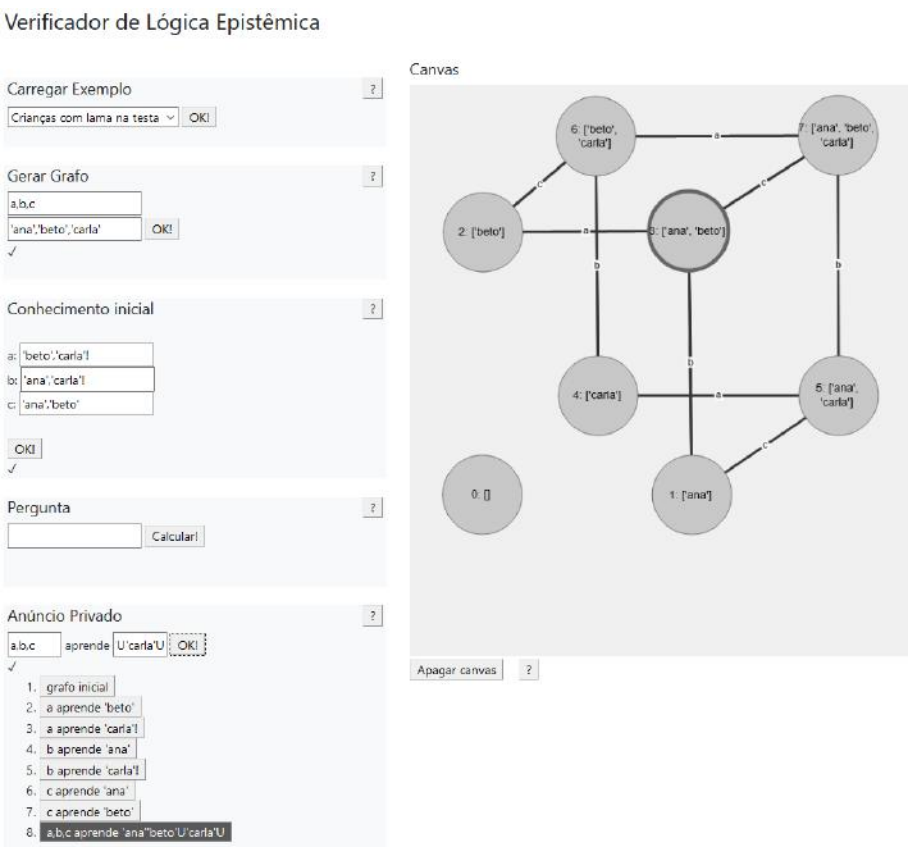


Figura 38 – Conhecimentos iniciais de cada agentes considerando o estado `'3'` como real.



anúncio público pode ser simulado no campo `'Anúncio Privado'` da interface passando uma lista com todos os agentes definidos na modelagem. Assim, traduzindo o anúncio feito pelo responsável para a notação pós-fixada, o programa recebe `'a,b,c' aprende 'ana"beto'U'carla'U` (Figura 39).

Figura 39 – Grafo após anúncio do responsável, ouvido por todas as crianças, de que existe pelo menos uma delas com lama na testa.

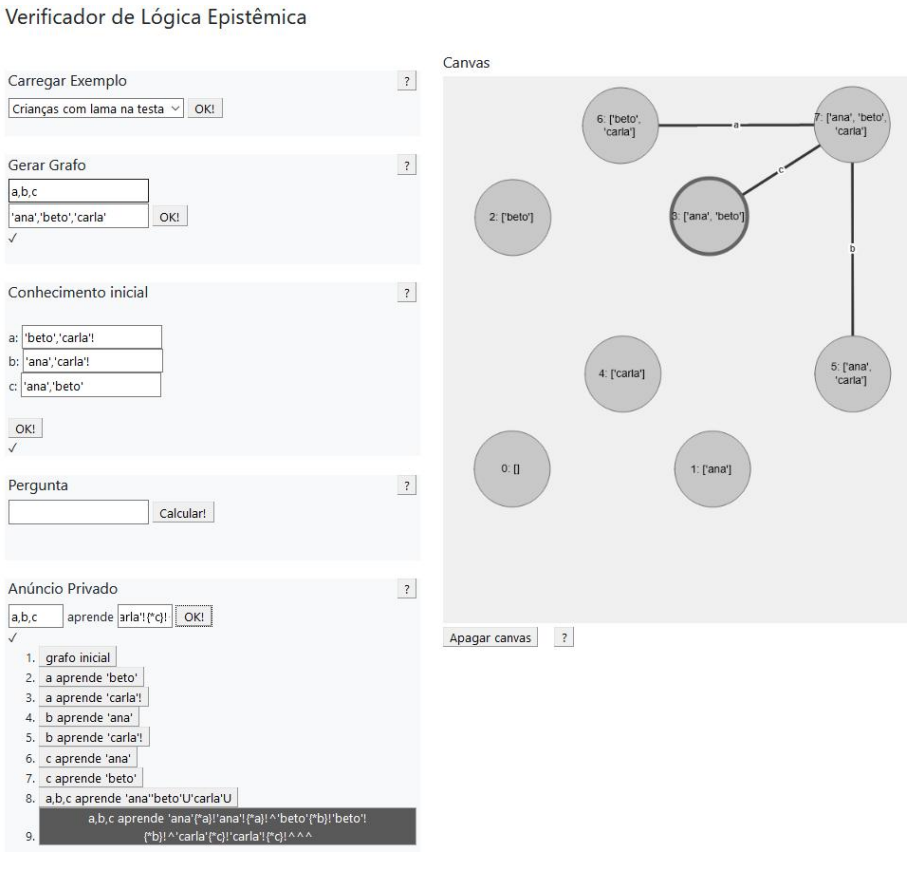


Na sequência, o responsável faz uma pergunta ouvida por todas as crianças: "alguém já sabe se tem lama na testa?". Como nenhuma criança responde, é concluído que a resposta ao questionamento é "não". A pergunta feita e o silêncio das crianças como resposta pode ser traduzido num novo anúncio público, onde todas as crianças agora sabem que nenhuma delas já sabe se tem lama na testa. Em notação lógica, isso pode ser escrito como $\neg(K_a ana \vee K_a \neg ana) \wedge \neg(K_b beto \vee K_b \neg beto) \wedge \neg(K_c carla \vee K_c \neg carla)$, que é equivalente a dizer $\neg K_a ana \wedge \neg K_a \neg ana \wedge \neg K_b beto \wedge \neg K_b \neg beto \wedge \neg K_c carla \wedge \neg K_c \neg carla$. Assim, um novo anúncio é feito ao programa, onde todas as crianças tomam conhecimento sobre o fato.

Traduzindo para notação pós-fixada, o anúncio é escrito como `'a,b,c' aprende 'ana'{*a}!'ana'!{*a}!^'beto'{*b}!'beto'!{*b}!^'carla'{*c}!'carla'!{*c}!^^` , como mostrado na Figura 40. Observando com atenção o grafo resultante, dúvidas relacionadas a estados onde somente uma criança tem lama na testa foram eliminadas. Isso

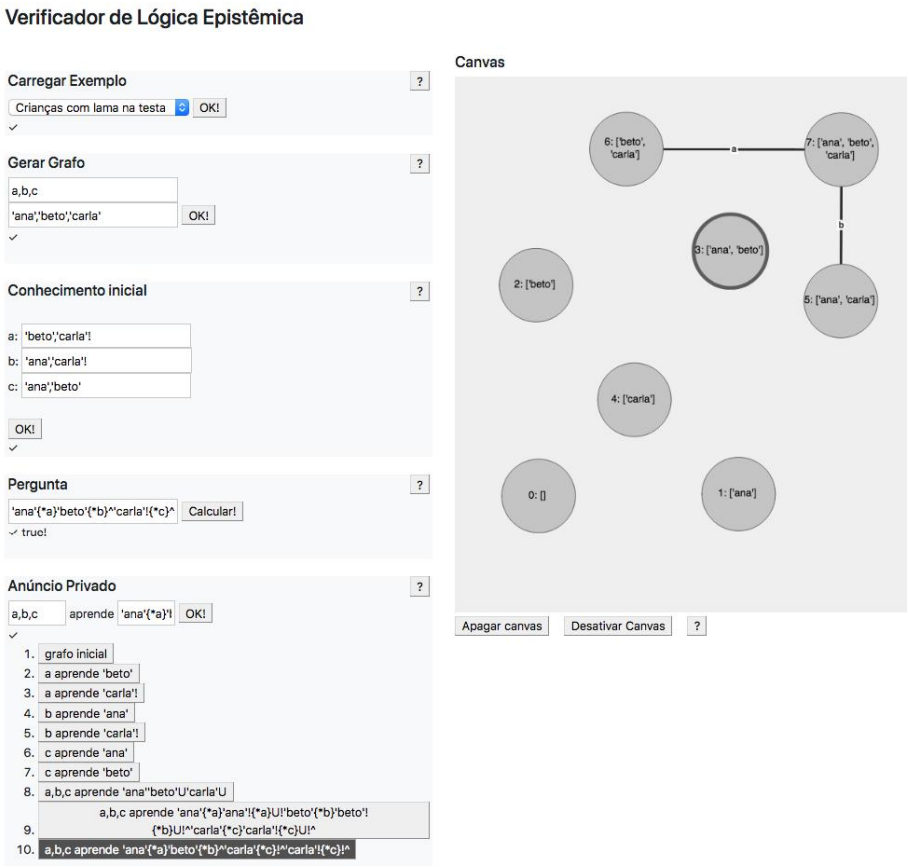
faz sentido pois as crianças já sabem que pelo menos uma delas tem lama na testa, por conta do anúncio inicialmente feito, e porque se somente uma criança tivesse lama na testa ela teria respondido "sim" para a pergunta feita pelo responsável "alguém já sabe se tem lama na testa?".

Figura 40 – Grafo após todas as crianças saberem que ninguém já sabe se tem lama na testa.



O responsável faz novamente a mesma pergunta porém, desta vez, Ana e Beto já sabem que tem lama na testa e respondem "sim". Isso pode ser observado no grafo através da ausência de arestas de dúvidas de `'a'` e de `'b'` saindo do estado real. Para o programa, isso é equivalente a dizer que `'ana'{*a}'beto'{*b}^'carla'{*c}! ^'carla'!{*c}!^`, como mostrado na Figura 41. Em notação lógica, $K_a ana \wedge K_b beto \wedge \neg K_c carla \wedge \neg K_c \neg carla$. No momento que Carla escuta as respostas de Ana e Beto, ela mesma passa a saber sobre o estado de sua própria testa: ela não está suja. Isso pode ser comprovado perguntando se todas as crianças já sabem sobre o estado de suas próprias testas, o que, na linguagem do programa é escrito como `'ana'{*a}'beto'{*b}^'carla'!{*c}^`.

Figura 41 – Todas as crianças sabem agora se têm ou não lama na testa.



5 CONCLUSÃO

Neste trabalho, foi elaborada uma interface para a fácil modelagem e avaliação de modelos epistêmicos com a finalidade principal de avaliar a segurança de protocolos de comunicação. Dessa forma, o programa pretende auxiliar no estudo da segurança de protocolos do ponto de vista lógico e contribuir para que novos protocolos de comunicação possam ter sua privacidade validada antes da implementação.

A interface do programa, desenvolvida em JavaScript, está disponibilizada *online* e pode ser facilmente acessada na página https://spimpaov.github.io/tcc/. Nela, é possível gerar grafos automaticamente através da definição de uma lista de agentes e uma lista de proposições. Dado um grafo, é possível validar o conhecimento dos agentes sobre alguma informação específica. Também é possível definir um conjunto de conhecimentos iniciais que serão verdadeiros para os agentes durante toda a execução, além de definir anúncios privados que refletem em alterações no grafo e simulam a troca de mensagens entre agentes envolvidos em uma ação de comunicação.

Os três exemplos de protocolos de comunicação com chaves privada e pública descritos em (DOLEV; YAO, 1983) foram modelados no programa e podem ser facilmente carregados em sua integridade pela interface. Eles demonstram o comportamento esperado discutido no trabalho original e servem como prova de que o programa consegue desempenhar a finalidade desejada inicialmente. Além disso, um conhecido problema lógico ('Crianças com lama na testa'), foi também modelado no programa, demonstrando que é possível utilizar a interface no estudo de lógica epistêmica de forma mais generalizada e com facilidades visuais para entendimento.

5.1 TRABALHOS FUTUROS

Algumas melhorias podem ser pensadas a fim de incrementar o trabalho futuramente. Quando uma lista muito grande de proposições é passada para gerar o grafo, problemas de performance são observados. Os primeiros anúncios realizados sobre o grafo gerado demoram mais do que o ideal por conta da grande quantidade de estados e, consequentemente, de arestas que é necessário percorrer.

Além disso, poderia ser implementada uma funcionalidade nova que permitisse exportar e importar arquivos .JSON com grafos desenhados na interface. Isso possibilitaria que estudos realizados pudessem ser facilmente recuperados na interface numa nova execução, excluindo a restrição de carregar somente os exemplos já modelados e presentes no código fonte.

Para esse trabalho, foram implementadas todas as funcionalidades mostradas na interface, além das modelagens já discutidas. Porém, seria interessante também modelar outros

tipos de protocolos de comunicação além dos três exemplos utilizados, como por exemplo os protocolos *Kerberos* e *Andrew Secure RPC* avaliados logicamente em (BENEVIDES; FERNANDEZ; OLIVEIRA, 2018).

Outra ideia interessante para trabalhos futuros seria a implementação no programa de outros tipos de anúncios. Neste trabalho, se um dos agentes recebe uma informação qualquer como verdadeira através de um anúncio privado, todos os demais agentes sabem que o anúncio foi feito e para quem, embora não saibam o conteúdo do que foi anunciado. Seria interessante avaliar e comparar as mesmas modelagens feitas para os exemplos de (DOLEV; YAO, 1983) sob a perspectiva de anúncios privados diferentes, onde os demais agentes não tomam ciência sobre a ocorrência dos anúncios ou, ainda, anúncios que considerem a presença de agentes não confiáveis, representando mais adequadamente o papel do intruso nas comunicações analisadas.

Como demonstrado através da modelagem do problema 'Crianças com lama na testa', o programa é capaz de fazer a modelagem epistêmica de problemas mais gerais da área, não só protocolos de comunicação. Isso abre portas para que a interface seja utilizada também em contexto didático, durante o ensino de lógica epistêmica e de suas propriedades. Seria interessante também para o futuro validar o uso do trabalho nesse contexto, como ferramenta de ensino.

REFERÊNCIAS

BENEVIDES, M. R.; FERNANDEZ, L. C.; OLIVEIRA, A. C. de. Dolev-yao multi-agent epistemic logic. 2018.

BENTHEM, J. V.; EIJCK, J. V.; KOOI, B. Logics of communication and change. **Information and computation**, Elsevier, v. 204, n. 11, p. 1620–1662, 2006.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE transactions on Information Theory**, IEEE, v. 22, n. 6, p. 644–654, 1976.

DITMARSCH, H. V.; HOEK, W. van D.; KOOI, B. **Dynamic epistemic logic**. [S.l.]: Springer Science & Business Media, 2007. v. 337.

DOLEV, D.; YAO, A. On the security of public key protocols. **IEEE Transactions on information theory**, IEEE, v. 29, n. 2, p. 198–208, 1983.

EIJCK, J. van. **Dynamic epistemic modelling**. [S.l.]: CWI. Software Engineering [SEN], 2004.

EIJCK, J. van. a demo of epistemic modelling. In: **Interactive Logic. Selected Papers from the 7th Augustus de Morgan Workshop, London**. [S.l.: s.n.], 2007. v. 1, p. 303–362.

FAGIN, R. et al. **Reasoning about knowledge**. [S.l.]: MIT press, 2003.

GERBRANDY, J.; GROENEVELD, W. Reasoning about information change. **Journal of logic, language and information**, Springer, v. 6, n. 2, p. 147–169, 1997.

MCCARTHY, L.; REAS, C.; FRY, B. **Getting Started with P5. js: Making Interactive Graphics in JavaScript and Processing**. [S.l.]: Maker Media, Inc., 2015.

OREBAUGH, A.; RAMIREZ, G.; BEALE, J. **Wireshark & Ethereal network protocol analyzer toolkit**. [S.l.]: Elsevier, 2006.

PLAZZA, J. A. **Logics of Public Communications**. [S.l.]: North-Holland: 201–216, 1989. (Proceedings of the 4^{th} International Symposium on Methodologies for Intelligent Systems).

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, ACM New York, NY, USA, v. 21, n. 2, p. 120–126, 1978.

ROSCOE, A. W. Modelling and verifying key-exchange protocols using csp and fdr. In: IEEE. **Proceedings The Eighth IEEE Computer Security Foundations Workshop**. [S.l.], 1995. p. 98–107.

APÊNDICES

APÊNDICE A – MANUAL DE UTILIZAÇÃO DO CANVAS DO PROGRAMA.

A.1 CRIANDO, APAGANDO E MOVENDO ESTADOS

- Apertar `'s'` cria um novo estado na posição atual do cursor;
- Apertar `'s'` enquanto o cursor está sobre um estado já criado apaga tanto o estado quanto as transições que o tinham como origem ou destino. O cursor está sobre um estado quando a cor do estado se altera (de cinza para rosa);
- É possível alterar a posição atual de um determinado estado clicando com o botão esquerdo do *mouse* sobre ele e o arrastando enquanto mantém o botão pressionado.

A.2 DEFININDO ESTADO RAIZ

- O estado raiz do grafo é aquele que apresenta o contorno de maior grossura;
- Um novo estado raiz pode ser definido apertando `'r'` enquanto o cursor estiver sobre o estado que se deseja tornar raiz.

A.3 CONECTANDO ESTADOS

- Uma transição entre dois estados pode ser criada apertando `'t'` sobre um estado já existente. Em seguida, move-se o cursor até outro estado, que será o estado destino da transição. Quando o cursor atingir o estado destino, é necessário apertar `'t'` mais uma vez para fixar a transição;
- Caso tenha-se criado uma transição por engano (foi criada mas ainda não foi fixada), pode-se cancelar o evento apertando `'t'` novamente enquanto o cursor não estiver sobre nenhum estado (nem mesmo sobre o estado de origem);
- Uma transição já fixada pode ser apagada ao apertar `'t'` enquanto o cursor está sobre uma transição. O cursor está sobre uma transição quando a cor da transição se altera (de cinza para rosa).

A.4 EDITANDO ESTADOS E TRANSIÇÕES

- É possível editar o nome e as proposições de um determinado estado. Para editar essas informações basta apertar `'w'` enquanto o cursor estiver sobre um estado;
- É possível editar os agentes de uma determinada transição. Para editar essa informação basta apertar `'w'` enquanto o cursor estiver sobre uma transição;

- Ao apertar `'w'` com o cursor sobre um estado ou transição, o programa entra em modo de escrita, onde todas as teclas de controle são temporariamente desativadas. Deve-se digitar na caixa de texto que surge no topo esquerdo da página as novas informações desejadas. Quando satisfeito, basta apertar no botão `'OK!'` e o programa sairá do modo de escrita;

- Durante modo de escrita, pode-se verificar qual estado ou transição está sendo editado no momento através da cor alaranjada.

A.5 ALTERANDO O ZOOM E O POSICIONAMENTO GLOBAL

- É possível alterar o zoom atual do Canvas através da rolagem do *mouse*;

- Para alterar o posicionamento global do Canvas, basta clicar e arrastar a tela até o local desejado. O cursor precisa estar dentro dos limites do Canvas e não pode estar tocando nenhum estado ou transição.

A.6 IMPRIMINDO INFORMAÇÃO SOBRE O ESTADO ATUAL DO PROGRAMA

- Apertar `'i'` faz com que o programa imprima informações gerais de depuração no console do navegador.