Geração de fórmulas da Lógica Proposicional baseada na Síntese de Programas

Filipe da Silva Oliveira[1,2], **Elthon Oliveira**[1]

[1]Universidade Federal de Alagoas (UFAL) - Campus Arapiraca
Avenida Manoel Severino Barbosa, s/n – Bom Sucesso, 57309-005 – Arapiraca – AL
[2]Bolsista PIBIC Edital 2019-2020 UFAL/CNPq/FAPEAL.

{filipe.oliveira,elthon}@arapiraca.ufal.br

Resumo. *A elaboração de problemas com características específicas da disciplina de Lógica é tida como uma tarefa tediosa por parte do professor. Neste artigo é apresentado uma abordagem para geração automática de fórmulas da Lógica Proposicional. O trabalho adapta a técnica de Geração de Esboço da Síntese de Programas na construção das fórmulas. Foi desenvolvido um sintetizador capaz de gerar fórmulas válidas baseadas em parâmetros fornecidos pelo usuário. A abordagem apresentada teve bons resultados que impulsionam a evolução do sintetizador com o objetivo da síntese de argumentos da Lógica Proposicional.*

Abstract. *The elaboration of problems with specific characteristics of the Logic discipline is considered a tedious task on the part of the teacher. This article presents an approach for automatic generation of formulas from Propositional Logic. The work adapts the technique of Sketch Generation of the Program Synthesis in the construction of formulas. A synthesizer was developed capable of generating valid formulas based on parameters provided by the user. The presented approach had good results that propel the evolution of the synthesizer aiming the synthesis of arguments of Propositional Logic.*

1. Introdução

A *Síntese de Programas* [Gulwani et al. 2017] é a tarefa de encontrar automaticamente um programa numa determinada linguagem de programação. Esta busca deve satisfazer a intenção do usuário descrita na forma de alguma especificação. Este problema tem sido considerado o *Santo Graal* da Ciência da Computação. Alguns pesquisadores consideravam a *Síntese de Programas* como um dos problemas centrais da teoria da programação [Pnueli and Rosner 1989]. Neste trabalho, é utilizada e adaptada a abordagem de *Geração de Esboço* para sintetizar fórmulas da *Lógica Proposicional*.

Dentro do contexto de ensino formal de qualquer disciplina, criar novos problemas a serem solucionados pelos discentes é uma tarefa frequente do docente. Considerando que estes problemas precisam ter características específicas de solução, tal como um determinado nível de dificuldade ou que envolvam o uso de um determinado conjunto de conceitos, a tarefa pode ser extremamente tediosa. Uma solução para isto seria automatizar a geração de problemas.

A abordagem deste trabalho busca eliminar a tarefa da geração manual de fórmulas de lógica. Além disso, esta abordagem objetiva fornecer uma ferramenta para

alunos praticarem os conceitos da disciplina e para os professores, que possuirão uma fonte inesgotável de exercícios para seus alunos. A ferramenta desenvolvida com esta abordagem poderá evitar fraudes [Mozgovoy et al. 2010] em salas de aula ou MOOCs (*Massive Open Online Courses*), já que cada aluno pode receber um problema de lógica diferente, mas no mesmo nível de dificuldade.

O sintetizador apresentado neste trabalho foi desenvolvido com intuito da geração automática de fórmulas da *Linguagem Proposicional*. A partir da utilização de algumas métricas da computação, o sistema consegue gerar as fórmulas, tendo como base parâmetros fornecidos pelos usuários. O sistema é fundamentado na técnica da *Síntese de Programas* para *Geração de Esboço*.

Este artigo está organizado da seguinte forma. Inicialmente são citados alguns trabalhos relacionados na Seção 2. Alguns conceitos essenciais para o entendimento da abordagem deste artigo são apresentados na Seção 3. Na Seção 4 a abordagem em questão é exposta, assim como também a ferramenta desenvolvida neste trabalho. Alguns exemplos de fórmulas sintetizadas com a ferramenta e comparações realizadas com alguns trabalhos da Seção 2, são apresentadas na Seção 5. Conclusões sobre o artigo e trabalhos futuros estão na Seção 6.

2. Trabalhos Relacionados

Os trabalhos aqui apresentados dizem respeito a geração automática de problemas voltados à educação.

Em [Singh et al. 2013] os autores descrevem um método para fornecer feedback automaticamente para problemas iniciais de programação. Para utilizar o método, os usuários precisam fornecer uma implementação de referência da tarefa e de um erro modelo que consiste em possíveis correções a erros que os alunos podem fazer. A partir disso, é retornado para os usuários soluções mínimas para seus erros e um feedback desses erros. Os autores também utilizam uma extensão da abordagem de *Geração de Esboço*.

A resolução de problemas de geometria da régua e da bússola é descrita em [Gulwani et al. 2011]. A ferramenta proposta pelos autores pode sintetizar com sucesso construções para vários problemas de geometria, problemas estes levantados nos livros didáticos do ensino médio e em exames por um tempo razoável.

Uma possível solução para lidar com a educação virtual, no que diz respeito a geração de problemas é apresentada em [Sadigh et al. 2012]. Com o advento dos MOOCs, diversos desafios surgiram, dentre eles a automação da geração de problemas, criação de soluções e classificação desses problemas e soluções, tudo isso para lidar com o grande número de alunos. Com isso, os autores apresentam um passo para enfrentar tal desafio utilizando como exemplo um curso de sistemas embarcados.

Para estudo da álgebra podemos citar [Singh et al. 2012] que ao passar um problema de álgebra, a ferramenta gera outros problemas semelhantes a este. Segundo os autores, a abordagem é relevante, pois pode ser utilizada por professores na aplicação de exames com problemas similares e com alunos na prática repetitiva em uma determinada dificuldade. Outra proposta de álgebra é descrita por [Andersen et al. 2013]. A abordagem analisa as principais dificuldades na geração de problemas e propõe uma ferramenta para auxiliar nesse processo. Os autores sintetizam problemas de matemática para o en-

sino fundamental e médio.

Os trabalhos citados acima assemelham-se com a abordagem deste artigo em relação ao propósito, ou seja, na geração automática de exercícios. Em um contexto mais específico, a presente abordagem trata um problema diferente dos mencionados, que é a *Lógica Proposicional*.

3. Fundamentação Teórica

3.1. Síntese de programas

A *Síntese de Programas* é a tarefa de encontrar um programa automaticamente através de uma linguagem de programação subjacente, que satisfaça uma determinada especificação [Gulwani et al. 2017]. Desde o início da Inteligência Artificial (IA), em 1950, este problema tornou-se o *Santo Graal* da Ciência da Computação. Isso acontece devido a ideia central do tema fugir da realidade da programação de computadores comum que tem-se no dia a dia. Sendo que, ao contrário de compiladores típicos que traduzem um código totalmente especificado de alto nível para representação de máquina de baixo nível, a *Síntese de Programas* executa alguma forma de pesquisa sobre o espaço de programas para gerar um programa que é consistente com uma variedade de restrições [Gulwani et al. 2017].

A *Síntese de Programas* é notoriamente um problema desafiador. Seus dois principais desafios são:

- **Espaço de programas** - o número de programas em qualquer trivial linguagem de programação cresce exponencialmente com o tamanho do programa. Este grande número de possíveis candidatos de um programa por um longo tempo tem renderizado uma tarefa intratável.
- **Intenção do usuário** - muitos domínios de aplicações da vida real para *Síntese de Programas* são bem complexas para serem descritas completamente com uma formal ou informal especificação. Os métodos para expressar a intenção do usuário variam desde especificações lógicas para descrições informais em linguagem natural até exemplos de entrada e saída.

Na literatura de *Síntese de Programas* existem muitas técnicas para sintetização de programas. As principais abordagens são:

- Geração de Esboço - os programadores passam uma estrutura básica do programa, ou seja, um esboço, que possui lacunas em seu interior para serem preenchidos com a sintetização [Gulwani et al. 2017].
- Pesquisa Enumerativa - é uma abordagem de força bruta bastante óbvia, com um truque elegante, e apesar de sua ingenuidade aparente, tem sido usada com grande efeito [Bornholt 2015].
- Pesquisa Estocástica - a abordagem aprende uma distribuição sobre o espaço de programas no espaço de hipóteses que é condicionado à especificação e, em seguida, provam programas da distribuição para aprender um programa consistente [Gulwani et al. 2017].
- Programação por Exemplos - é um subcampo da *Síntese do Programas*, onde a especificação de um determinado programa é dada na forma de entrada e saída de exemplos [Gulwani 2016].

Quando trabalhamos com *Síntese de Programas*, é preciso ressaltar que os sintetizadores são desenvolvidos para gerar diferentes tipos de programas. Esses 'tipos' de programas, são definidos através das *gramáticas*, que geram objetos através dos sintetizadores, como por exemplo: um algoritmo simples, um número, uma equação matemática ou até mesmo uma fórmula da *Lógica Proposicional*. No contexto da *Teoria da Computação* uma *gramática* é um modelo utilizado para gerar uma linguagem.

3.2. Geração de Esboço

Neste trabalho utiliza-se uma abordagem baseada na *Geração de Esboço*. Esta técnica de síntese permite que os programadores expressem as atividades de alto nível de um problema escrevendo um esboço. O esboço é um programa parcial que codifica a estrutura de uma solução e deixa seus detalhes de baixo nível sem especificação [Solar-Lezama 2009]. Na *Geração de Esboço* o programador também precisa fornecer uma implementação de referência ou um conjunto de rotinas de teste que o código sintetizado deve passar.

O usuário ao informar um programa parcial para um determinado sintetizador, deve lembrar de fazê-lo deixando os *buracos* que devem ser preenchidos com a sintetização. Os *buracos* do programa parcial completam o programa sintetizado que deve ser testado com o conjunto de rotinas fornecidos pelo usuário [Solar-Lezama and Bodik 2008]. Nesse tipo de síntese o principal intuito é o preenchimento dos *buracos*, que são as partes que precisam ser sintetizadas em um programa parcial.

A abordagem deste artigo adapta para utilização outra característica importante da *Geração de Esboço*, que chamamos de *Oráculo*. O *Oráculo* foi introduzido com [Solar-Lezama and Bodik 2008] numa abordagem de *Síntese Indutiva* caracterizada como *counterexample-guided inductive synthesis (CEGIS)* - síntese indutiva guiada por contraexemplo. O CEGIS funciona da seguinte forma: dada uma especificação ϕ e um determinado programa candidato P, o *Oráculo* retorna "SIM" se P atender ϕ ou ("NÃO", x), se P não satisfaz ϕ. Onde, 'x', que é um contraexemplo de P, o qual viola ϕ.

3.3. Linguagem Proposicional

Quando vamos definir uma *Linguagem Formal* precisamos definir seus dois componentes básicos: um *alfabeto* e as *regras de produção*. O *alfabeto* da *Lógica Proposicional* é definido por:

- Um conjunto de *símbolos proposicionais*, que também são chamados de *átomos*, ou de *variáveis proposicionais*: $Q = \{p_0, p_1, ...\}$.
- O *conectivo unário* $\neg$ (negação, lê-se: NÃO).
- Os *Conectivos binários* $\wedge$ (conjunção, lê-se E), $\vee$ (disjunção, lê-se: OU), $\rightarrow$ (implicação, lê-se: SE ... ENTÃO ...), e $\leftrightarrow$ (bicondicional, lê-se: SE SOMENTE SE).
- Os elementos de pontuação, que são apenas os parênteses: '(' e ')'.

Os elementos L_{LP} da *Lógica Proposicional* são chamados de *fórmulas*, mais especificamente *fórmulas bem formadas - fbfs*. A seguir temos uma definição formal de *fbf* de acordo com [Silva et al. 2006].

O conjunto das fórmulas da *Lógica Proposicional* é definido por *indução* e possui três casos: um caso básico e dois casos indutivos. Sendo assim, o conjunto L_{LP} das

fórmulas proposicionais é definido indutivamente como sendo o menor conjunto satisfazendo às seguintes regras de formação:

1. **Caso básico:** Todos os símbolos proposicionais estão em L_{LP}, ou seja, $Q \subseteq L_{LP}$. Os símbolos proposicionais são chamados de fórmulas atômicas ou átomos. Dessa forma, os símbolos proposicionais também são *fbfs*.
2. **Caso indutivo 1:** Se $\alpha \in L_{LP}$, então $\neg\ \alpha \in L_{LP}$.
3. **Caso indutivo 2:** Se $\alpha, \beta \in L_{LP}$, então $(\alpha \wedge \beta) \in L_{LP}$, $(\alpha \vee \beta) \in L_{LP}$, $(\alpha \rightarrow \beta) \in L_{LP}$, $(\alpha \leftrightarrow \beta) \in L_{LP}$.

Na abordagem de geração automática de fórmulas da *Lógica Proposicional*, utiliza-se a definição formal apresentada nesta Seção. Além disso, uma *gramática livre de contexto* que descreve as palavras da *Linguagem Proposicional* foi utilizada no sintetizador desenvolvido.

4. Síntese de fórmulas

Nesta Seção é apresentada a implementação da abordagem apresentada. O sistema consiste em um sintetizador de fórmulas da *Lógica Proposicional*. A partir das técnicas citadas na Seção 3 o sistema consegue gerar as fórmulas de forma aleatória, considerando também os parâmetros fornecidos pelos usuários. Pela técnica de *Geração de Esboço* o usuário informa qual será o conectivo lógico principal que as fórmulas devem conter. Por exemplo, se o usuário escolher a conjunção, as fórmulas a serem sintetizadas terão a seguinte estrutura:

$$(\alpha) \wedge (\beta)$$

Após a escolha do conectivo lógico principal, o usuário deve definir alguns critérios, tais como: a quantidade de fórmulas a serem geradas, quantidade de átomos diferentes no processo de síntese e a quantidade de operações lógicas a serem utilizadas além do conectivo principal.

O sintetizador foi desenvolvido na linguagem de programação Python. É constituído por dois arquivos contendo classes e operações essenciais para o seu funcionamento. O arquivo *FBF.py* contém a classe *FBF* que representa o modelo de uma *fbf* da *Linguagem Proposicional*. Tendo como base a definição de *fbf* apresentada na Seção 3.3, a classe possui três atributos: um conectivo lógico, uma parte esquerda e uma parte direita. Pela definição as partes (esquerda e direita) são *fbfs*. Para representar a negação definimos uma métrica onde a parte esquerda é nula e o operador é o $\neg$, onde a operação possui a estrutura a seguir:

$$\neg\ (\alpha)$$

A classe *FBF* possui desde métodos simples como os métodos de acesso e modificação dos atributos até métodos mais sofisticados como os listados a seguir:

- **toString()** - retorna a representação textual de uma fórmula. Ao acioná-lo a *fbf* não será apresentada na estrutura de árvore, ou seja, como o objetivo da classe é propriamente dito, mas sim como um texto.
- **insertFBF()** - insere uma *fbf* no esboço atual. A posição onde a *fbf* é inserida é obtida de forma aleatória a partir da quantidade de *buracos* presentes no esboço.

- **count()** - gera um *ranking* para a fórmula. Este método caracteriza uma função de *ranking* que conta quantas vezes o fenômeno de lados (esquerdo e direito) iguais de uma *fbf* acontece. O método considera toda parte recursiva do objeto *fbf* e incrementa 1 em um contador quando o fenômeno acontece. No final da ativação da função de *ranking* é retornado um valor inteiro que representa a contagem. Atualmente a função de *ranking* é utilizada no sintetizador para gerar e atribuir um *score* para cada uma das fórmulas. Considerando a inserção aleatória de operações no esboço e átomos nos *buracos*, podemos ativar a função e a partir de sua métrica de utilização encontrar um valor inteiro associado a cada fórmula. Sendo assim, quando a sintetização acontece, as fórmulas são apresentadas para o usuário em ordem crescente pelos seus *scores*, ou seja, um *ranking*.

O arquivo *sintetizador.py* contém operações essenciais para o funcionamento do sistema. Abaixo são apresentadas as operações e suas respectivas especificações:

- **countAtoms()** - conta a quantidade de átomos presentes em uma fórmula;
- **generateSketch()** - operação que gerencia a criação do esboço para as fórmulas e insere *fbfs* em um total de até duas além da operação principal;
- **insertComplexFBF()** - insere uma *fbf* em uma fórmula quando a quantidade de operações é maior que três, a partir de uma delegação de funcionalidade feita por 'generateSketch()';
- **getAtoms()** - retorna uma lista de átomos dependendo da quantidade requerida pelo usuário, sendo que os átomos dessa lista inicia a partir das primeiras letras do alfabeto;
- **fillWithAtoms()** - depois que o esboço da fórmula é gerado, os átomos são inseridos aleatoriamente nos *buracos* do esboço;
- **synthesize()** - recebe os parâmetros do usuário e delega para que as outras operações consigam gerar as fórmulas;
- **getFormulas()** - extrai as fórmulas da lista de tuplas (fórmula, *score*) que são gerados na sintetização das fórmulas;
- **removeParentheses()** - remove parênteses desnecessários da fórmula;
- **sortFormulasByScore()** - ordena as fórmulas pelos *scores*.

A arquitetura e funcionamento do sintetizador são apresentados na Figura 1. A ferramenta opera com entradas do usuário e o processamento entre operações e objetos.

Na Figura 1, em (1) o usuário inicia a ferramenta e informa a quantidade de fórmulas que deseja na sintetização. Em (2) escolhe a operação principal que vai definir como será o esboço das fórmulas geradas. Em (3) e (4), respectivamente, informam a quantidade átomos presentes na sintetização e quantidade de operações nas fórmulas além da principal.

Os principais componentes da arquitetura da ferramenta consistem em: *Sintetizar* e *GerarEsboço*. Em (5) o sistema transfere as informações de entrada fornecidas pelo usuário para a operação *Sintetizar*, que delega algumas das informações para a operação *GerarEsboço* (6). A operação *GerarEsboço* gera um esboço de uma fórmula, inserindo as operações além da operação principal como estruturas *fbfs*. As inserções de operações no esboço da fórmula são de acordo com a quantidade que o usuário informou. O esboço é retornado para o *Sintetizar* em (7). O esboço da fórmula possui alguns *buracos* que são

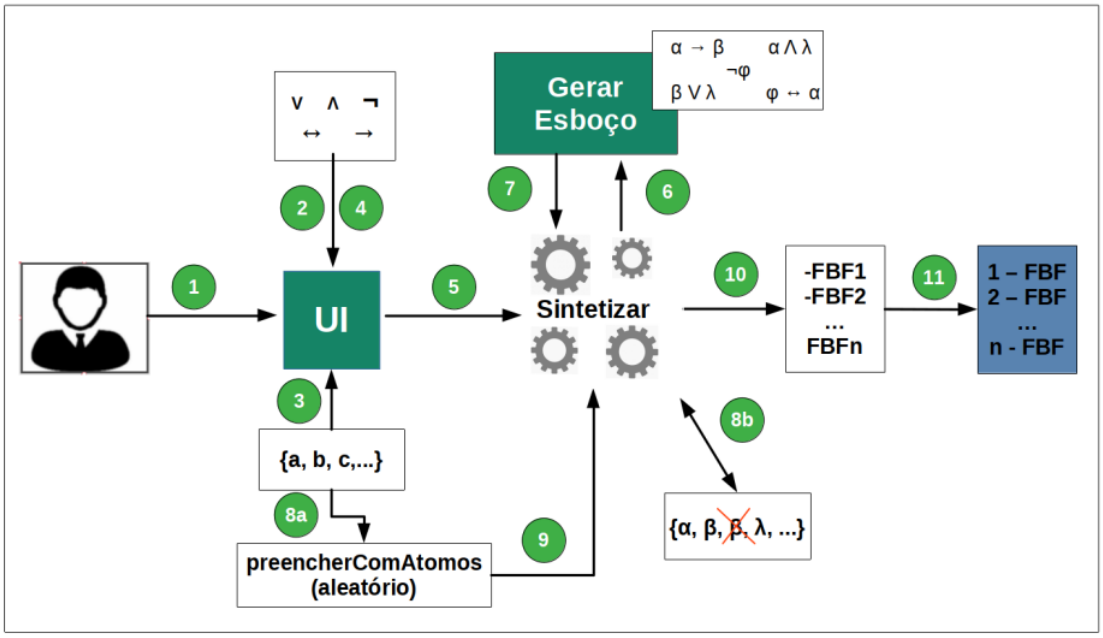

Figura 1. Componentes principais da arquitetura que permitem o funcionamento do sintetizador.

Fonte: Autoria própria. ;

preenchidos aleatoriamente com átomos em (8a). Depois dos *buracos* serem preenchidos, uma fórmula lógica é gerada e retornada para *Sintetizar* (9). O sistema verifica se há duplicidade da fórmula gerada em (8b).

A verificação de duplicidade funciona como uma extensão do *oráculo* descrito na Seção 3.2 e sua representação pode ser observada na Figura 2. A fórmula candidata é gerada pelo *GerarEsboço* e é verificada sua duplicidade pelo *Verificador*. Se a fórmula já tiver sido sintetizada ela é descartada e o *Verificador* informa ao *GerarEsboço* que inicie uma nova síntese para a fórmula. Se não foi sintetizada, ela é acrescentada à lista de fórmulas sintetizadas, assim como também o *score* gerado com a função 'count()'. A síntese continua até o número de fórmulas requerido pelo usuário ser atendido em (10).

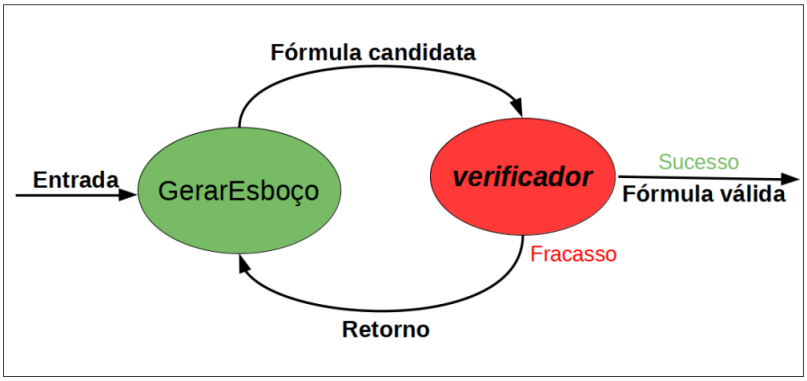

Figura 2. Verificador de duplicidade.
Fonte: Baseado em [Solar-Lezama and Bodik 2008],

Ainda sobre o funcionamento do sintetizador apresentado na Figura 1, quando

número de fórmulas requerido pelo usuário é atingido, a lista de fórmulas passa por um processo de ordenação pelo *score* de cada fórmula. Após a ordenação, as fórmulas são apresentadas para o usuário (11) por meio de um ranqueamento crescente.

5. Resultados

Para mostrar o funcionamento da abordagem de síntese proposta neste artigo, foi desenvolvido um sintetizador de fórmulas da *Lógica Proposicional* apresentado na Seção 4. A seguir são apresentados alguns resultados da implementação realizada:

Exemplo 1: ***Síntese de fórmulas com a conjunção como operação principal.***

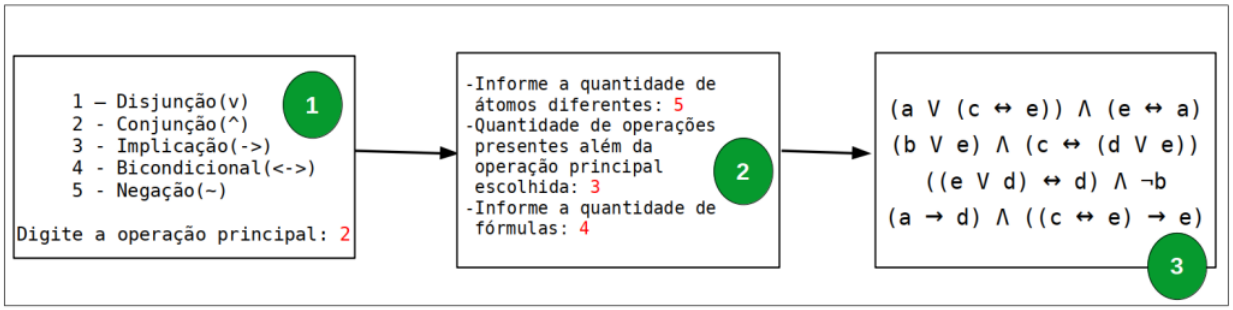

Figura 3. Sequência de telas para conjunção como operação principal.
Fonte: Autoria própria.

A sequência de passos realizada na execução da ferramenta são apresentados na Figura 3. Como a ferramenta possui uma interface de texto, são apresentadas os textos mostrados ao usuário. No **passo 1** o usuário informa que a conjunção é o conectivo principal das fórmulas. O sistema pede as métricas para a síntese (quantidade de átomos diferentes, quantidade operações além da principal e quantidade de fórmulas) no **passo 2**. E por fim, no **passo 3** as fórmulas sintetizadas são apresentadas para o usuário.

Exemplo 2: ***Síntese de fórmulas com a disjunção como operação principal.***

Na Figura 4 o processo é similar ao da Figura 3. Neste exemplo a operação principal é a disjunção e os parâmetros fornecidos para a síntese são diferentes do primeiro exemplo, consequentemente a síntese gera um conjunto diferente de fórmulas apresentadas no **passo 3**.

Os exemplos um e dois mostram a síntese de fórmulas da *Lógica Proposicional* realizadas com o sintetizador. Os dois exemplos tomam como base o funcionamento apresentado nas Figuras 1 e 2.

Como já dito, a abordagem deste artigo assemelha-se com os trabalhos citados na Seção 2 quando leva-se em consideração a geração automática de questões, mas diferencia-se quando observa-se a síntese de fórmulas da *Lógica Proposicional*. Além disso, assemelha-se em duas características particulares de dois trabalhos citados. A primeira característica, presente em [Sadigh et al. 2012], é a de lidar com os desafios adquiridos com o advento dos MOOCs. O sintetizador desenvolvido, possui a capacidade de automação da geração de problemas e a classificação desses problemas, levando em consideração que a ferramenta será evoluída, tornando-se disponível para a comunidade.

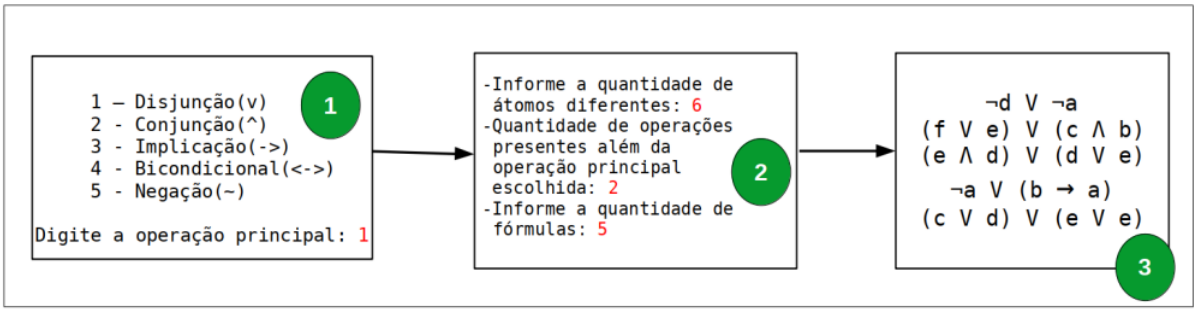

Figura 4. Sequência de telas para disjunção como operação principal.
Fonte: Autoria própria.

Com o sintetizador, um professor, por exemplo, também pode definir o conteúdo dos assuntos que as fórmulas geradas devem apresentar, característica presente na abordagem de [Singh et al. 2012].

6. Conclusão e Trabalhos Futuros

Neste artigo foi apresentado uma abordagem de *Síntese de Programas* para geração automática de fórmulas da *Lógica Proposicional*. O trabalho ainda está em fase de desenvolvimento, sendo que nesse primeiro momento a geração das fórmulas propriamente teve um maior foco. A abordagem tende a ser bastante promissora, pois busca suprir uma necessidade real dos estudantes de Lógica e professores da disciplina. A partir disso, foi desenvolvido um sintetizador capaz de gerar os elementos da *Linguagem Proposicional*. Com este sintetizador, os estudantes terão acesso a um arsenal de questões para praticar os conteúdos curriculares. Logo, os professores não precisarão investir muito tempo com a elaboração de exercícios para os estudantes.

Atualmente está sendo desenhada uma extensão para a abordagem apresentada que possibilitará a geração de argumentos da *Lógica Proposicional*. Desta maneira, alunos e professores poderão trabalhar com dedução natural. Os argumentos serão sintetizados a partir da geração de fórmulas. Porém, a síntese de argumentos traz outros desafios, como a improbabilidade de gerar um argumento válido aleatoriamente. Além disso, é pretendido projetar uma API (*Interface de Programação de Aplicações*) e um sistema completo para utilização aos usuários.

Referências

Andersen, E., Gulwani, S., and Popovic, Z. (2013). A trace-based framework for analyzing and synthesizing educational progressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 773–782.

Bornholt, J. (2015). Program synthesis explained. `https://www.cs.utexas.edu/~bornholt/post/synthesis-explained.html`. Último acesso em 16/10/2019.

Gulwani, S. (2016). Programming by examples (and its applications in data wrangling). In *Verification and Synthesis of Correct and Secure Systems*. IOS Press.

Gulwani, S., Korthikanti, V. A., and Tiwari, A. (2011). Synthesizing geometry constructions. *ACM SIGPLAN Notices*, 46(6):50–61.

Gulwani, S., Polozov, A., and Singh, R. (2017). *Program Synthesis*, volume 4. NOW.

Mozgovoy, M., Kakkonen, T., and Cosma, G. (2010). Automatic student plagiarism detection: future perspectives. *Journal of Educational Computing Research*, 43(4):511–531.

Pnueli, A. and Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190.

Sadigh, D., Seshia, S. A., and Gupta, M. (2012). Automating exercise generation: A step towards meeting the mooc challenge for embedded systems. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, pages 1–8.

Silva, F. S. C. d., Finger, M., and Melo, A. C. V. d. (2006). *Lógica para computação*. Cengage Learning.

Singh, R., Gulwani, S., and Rajamani, S. (2012). Automatically generating algebra problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Singh, R., Gulwani, S., and Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 15–26.

Solar-Lezama, A. (2009). The sketching approach to program synthesis. In Hu, Z., editor, *Programming Languages and Systems*, pages 4–13, Berlin, Heidelberg. Springer Berlin Heidelberg.

Solar-Lezama, A. and Bodik, R. (2008). *Program synthesis by sketching*. Citeseer.