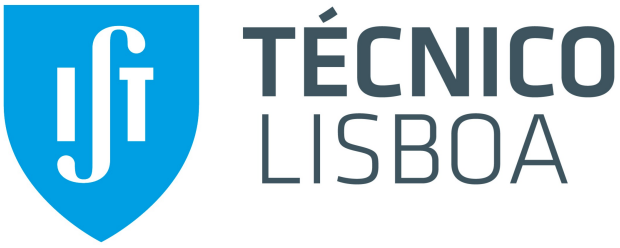
Instituto Superior Técnico
MEAer



Inteligência Artificial e Sistemas de Decisão

Projeto 2

Realizado por:
Henrique Ferreira, nº 78654;
Manuel Rosa, nº 78679.

Ano letivo 2017-18
1º Semestre

1 Descrição do problema

Um agente racional deve ter a capacidade de inferir novos factos a partir de uma base de conhecimento verdadeira. Para tal, é necessário que o agente represente o mundo numa base de conhecimento sintática, com fórmulas bem formuladas, da qual possa inferir novas fórmulas verdadeiras que posteriormente podem ser interpretadas para factos reais. Este trabalho assume que existe uma correta representação da base de conhecimento e centra-se na validação do novo conjunto (união da base de conhecimento com a fórmula a validar).

Verificar a validade lógica de um conjunto de fórmulas é um processo metódico com várias abordagens que podem ser implementadas. Neste projeto pretende-se implementar a validação lógica através do algoritmo de inferência por resolução. Este algoritmo assume que todas as fórmulas estão representadas sob a forma de "clauses" (disjunção de literais) e aplica a regra de resolução para inferir a validade do conjunto. A linguagem utilizada é a de lógica proposicional.

Assim sendo, o trabalho desenvolvido consiste em dois programas independentes. O Conversor CNF (Forma Normal Conjuntiva) lê um conjunto de fórmulas bem formuladas e aplica sucessivamente várias regras de transformação. O programa termina quando obtiver um conjunto de fórmulas, equivalente ao inicial, contendo apenas conjunções de clauses. O Algoritmo de Resolução lê um conjunto de fórmulas em CNF e aplica repetidamente a regra de resolução entre quaisquer duas de forma a produzir uma nova. Este programa termina quando de uma resolução resulta o conjunto vazio ou quando já não é possível criar novas clauses a partir das existentes e retorna o valor lógico associado à terminação.

2 Conversor CNF

Nesta secção descreve-se sucintamente o funcionamento do conversor CNF. O programa começa por ler do *stdin* uma lista de fórmulas bem formatada, seguindo a formatação para lógica proposicional representada na Figura 1a. Uma frase pode ser atómica, onde atom='C' e C corresponde ao nome da constante, ou composta. As frases compostas podem ser a negação de uma frase ou uma operação entre duas frases do tipo conjunção, disjunção, implicação ou equivalência, i.e. typ={'and', 'or', '=>', '<=>'}. De seguida, o programa opera separadamente sobre cada uma das frases da lista recebida, aplicando a cada uma a função de conversão para CNF e guardando o resultado numa nova lista. A nova lista é então unificada de forma a eliminar clauses iguais e literais repetidos na mesma clause. Finalmente, a saída do programa é escrita para o *stdout*, uma clause por linha, com a formatação apresentada na Figura 1b. Cada clause é constituída por um literal (frase atómica ou sua negação) ou por uma lista de literais.

A função de conversão para CNF começa por testar o tipo de frase recebida e atua em conformidade. Caso a frase seja uma equivalência, implicação ou conjunção a função é chamada de novo para converter uma versão simplificada da frase (1 equivalência → 2 implicações, 1 implicação → 1 disjunção, 1 conjunção → 2 novas frases). Caso a frase seja atómica ou a negação de um frase atómica, esta já se encontra na forma CNF e é retornada, para a negação de uma frase composta é aplicada a lei de Morgan correspondente e chamada a função de conversão para a nova frase. A situação mais complicada ocorre quando a frase é uma disjunção. No caso de uma disjunção pura, definida como a disjunção entre literais ou disjunções de literais, a frase já está em CNF e é retornada. Caso contrário, cada uma das frases é convertida para CNF e de seguida aplica-se a propriedade distributiva entre cada frase para todas as clauses resultantes.

A função de unificação percorre a lista de clauses recebida e ignora todas aquelas que são repetidas. Caso a clause seja nova, os seus literais são obtidos e testa-se se já existe alguma clause com exatamente os mesmos literais, caso não exista, estes são então adicionados a uma nova entrada na lista de lietarais a retornar. O resultado final deste programa é uma lista de conjuntos de literais que representa a conjunção de disjunções.

```
 atom
('not', sent )
( typ , sent , sent )
```

(a) Formatação da entrada

```
 atom
('not', atom )
[literal, ... , literal]
```

(b) Formatação da saída

Figura 1: Formatações para interação com o conversor CNF

3 Algoritmo de Resolução

Procede-se, nesta secção, à clarificação do funcionamento do programa *prover.py* que aplica o algoritmo de resolução a um determinado grupo de clauses e tem como output *True* ou *False*.

Em primeiro lugar, lê do stdin os vários conjuntos de literais provenientes do conversor CNF e transforma para clauses bem definidas no contexto deste programa. Por outras palavras, cada clause é uma lista de objetos da classe MyLiteral. Note que, uma instância MyLiteral é definida pelo seu Nome e Valor Lógico. Assim, após o processamento de todos os grupos de literais, chega-se a uma lista de clauses, as quais serão submetidas ao algoritmo de resolução.

De seguida, a lista de clauses sofre três processos de simplificação. O primeiro - *remove_tautologies* - elimina todas tautologias, ou seja, as clauses que incluem um certo literal e o seu complementar. De seguida, *remove_equal* retira da lista as clauses que aparecem mais do que uma vez. Por fim, executa-se *remove_no_complementary* que remove todas as clauses que possuem um determinado literal para o qual não existe o seu complementar em nenhuma outra clause. Salienta-se que cada vez que se modifica a lista de clauses, podem surgir novas que sejam elegíveis para serem removidas. Daí a necessidade de repetir este processo até que não hajam mais alterações.

À lista de clauses otimizada aplica-se concretamente o mecanismo de resolução. Em concordância com a filosofia *unit preference*, começa-se por ordenar a lista por tamanho de clause, e assim dá-se preferência às de menor dimensão para a realização do algoritmo (ver figura 2). Neste sentido, começa-se por selecionar a primeira clause (CL 1), um literal (Literal X) dessa clause e outra clause (CL 2). De seguida, verifica-se se é possível aplicar resolução, isto é, se existe o complementar do Literal X na clause 2. Em caso negativo, procede-se para a análise da clause 3, e assim sucessivamente até ao final da lista. Posteriormente, realiza-se todo o mesmo procedimento tendo em conta um diferente literal da clause 1 até não haver mais literais inutilizados nessa clause. Nesse momento, toma-se como ponto de partida a clause 2 e aplica-se a resolução para o Literal Y e outra clause referente a um índice superior na lista. Ao chegar ao final da lista e se a resolução nunca for aplicada, então o output do programa é *False*.

Por outro lado, caso seja possível realizar resolução com duas clauses, constrói-se uma nova aplicando regras de *factoring*. Além disso, a lista de clauses atualizada é submetida aos três processos de simplificação previamente mencionados. Seguidamente, faz-se uma comparação entre esta última lista e a inicial (antes da aplicação da resolução). No caso de serem diferentes todo o processo será repetido (começando pela ordenação da nova lista de clauses). Se forem iguais, então o processo de resolução não adicionou nenhuma informação relevante, continuando-se com a mesma estratégia até ao final da lista ou até se encontrar uma nova clause.

Neste sentido, o resultado deste programa é *True*, se por meio do mecanismo de resolução surgir uma clause vazia; *False*, se todas as tentativas de resolução possíveis não adicionarem uma única nova clause à lista.

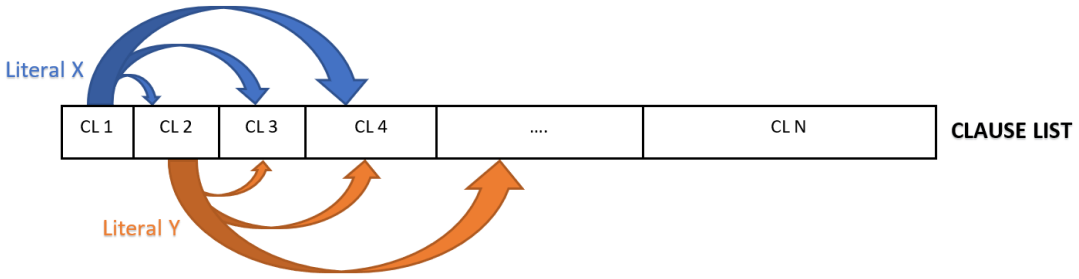


Figura 2: Implementação do mecanismo de resolução

Como considerações finais, releva-se que o programa poderia ser extendido para incluir uma *Blacklist* (construída ao longo do programa) de pares de clauses para as quais nunca seria possível aplicar resolução. Na mesma ótica de aumento de eficiência, poderia-se incluir uma quarta simplificação, baseada no facto de se existir uma clause que é um subconjunto de outra clause maior, então a última pode ser descartada.