Modelando Enredos de Narrativas como Planos

Angelo E. M. Ciarlini
e-mail: angelo@inf.puc-rio

Antonio L. Furtado
e-mail: furtado@inf.puc-rio.br

Sérgio Lifschitz
e-mail: lifschit@inf.puc-rio.br

PUC-RioInf.MCC 7/97. March, 1997

Abstract: The present work describes a method for modelling plots of narratives as plans, which permits to handle automatically the classification, complementation, adaptation and generation of plots. In particular, are presented the results of applying these ideas to modelling plots belonging to the literary gender of fairy tales, through a prototype system integrating plan recognition and generation. The method reflects the influence of literary theories, mainly from the Structuralist school.

Keywords: Plots of Narratives, Artificial Intelligence, Structuralism, Plan Recognition, Planning

Resumo: Neste trabalho, descreve-se um método para a modelagem de enredos de narrativas como planos, o que possibilita um tratamento automático para a classificação, complementação, adaptação e geração de enredos. São apresentados, em particular, os resultados da aplicação dessas idéias na modelagem de enredos de um certo gênero literário (contos de fadas) através de um sistema protótipo que integra reconhecimento e geração de planos. O método reflete a influência de teorias literárias, sobretudo da escola Estruturalista.

Palavras-Chave: Enredos de Narrativas; Inteligência Artificial; Estruturalismo; Reconhecimento de Planos; Geração de Planos

1. Introdução

O pesquisador russo Vladimir Propp [Pro68], um dos precursores da teoria literária conhecida como Estruturalismo, sugeriu que os textos literários poderiam ser estudados pela identificação de trechos, aos quais seriam associadas *funções* dentro da narrativa. Dessa forma, um enredo nada mais seria que uma seqüência de funções. Para se compreender melhor o conceito de função, adotado por Propp, podemos citar o gênero dos contos de fadas, com o qual trabalhou. Nesse gênero, seriam funções, por exemplo, *a vilania*, *a luta do herói com o vilão* e *a recompensa do herói*. Esse tratamento facilita sobremaneira a análise morfológica dos textos de um gênero, possibilitando a sua comparação e classificação. A análise morfológica pode servir como instrumento para pesquisas históricas, sobre a origem de um determinado texto, bem como para pesquisas antropológicas [Lev58], sobre a relação entre as literaturas de culturas diferentes e a ligação, dentro de uma certa cultura, entre a literatura e o contexto social, político e econômico de uma época.

Utilizando as idéias de Propp, procura-se, neste trabalho, uma forma de automatizar o tratamento de enredos de narrativas. Para isso, a idéia central é definir enredos como planos, da forma como esses últimos são vistos dentro da Inteligência Artificial. Um *plano* é uma seqüência de *operadores* que têm como objetivo gerar um efeito global específico. Cada operador, por sua vez, possui *pré-condições*, ou seja, um conjunto de fatos que devem ser válidos para que o operador possa ser aplicado, e *efeitos*, ou seja, modificações feitas por ele no conjunto de fatos válidos. O trabalho de modelagem implicou na necessidade de definir, de forma precisa, as funções proppianas para que elas correspondessem a operadores, possuindo, portanto, pré-condições e efeitos. Sendo assim, o operador correspondente a uma função de *vilania* pode ter, como pré-condição, que *a vítima esteja frágil* e, como efeito, que *ela passe a estar sofrendo*. Tornou-se necessário, também, identificar o efeito global (ou objetivo) da aplicação de um plano correspondente a um enredo típico. Um exemplo de objetivo associado a um conto de fadas típico pode ser *tornar o herói rico*, *fazer com que o herói se case com a vítima*, *punir o vilão*, etc.

A vantagem da modelagem de enredos como planos é a possibilidade de utilização de ferramentas de reconhecimento e geração de planos. O *reconhecimento de planos* tem como finalidade descobrir, uma vez fornecidas observações que indicam um conjunto de operadores aplicados, quais os planos, contidos em uma biblioteca de planos típicos, em que esse conjunto pode se encaixar. Uma vez reconhecido o plano, pode-se saber qual o objetivo implícito na execução do conjunto de operadores. Por outro lado, a *geração de planos (ou planejamento)* permite que, conhecendo-se um objetivo a ser alcançado, sejam geradas seqüências de operadores (planos) para atingi-lo. Integrando-se reconhecimento e geração de planos pode-se obter, como se verá mais adiante, um conjunto bem maior de planos alternativos. Será mostrada a utilização de um sistema que integra reconhecimento e geração de planos dentro do contexto dos enredos de contos de fadas.

Como resultado deste trabalho, é demonstrada a viabilidade da abordagem proposta para a classificação de enredos, para a previsão de próximos acontecimentos, quando se conhece apenas uma parte do enredo, para a adaptação de um enredo conhecido a condições específicas e para o apoio à criação de novos enredos. Deve-se notar que, em princípio, esse resultado pode ser utilizado em contextos não-literários. Pode-se imaginar a utilização da abordagem proposta, por exemplo, no apoio à elucidação de crimes, com base em algumas pistas e em uma biblioteca de crimes elucidados previamente; ou então no apoio a decisões estratégicas de um executivo dentro de uma empresa; ou ainda na elaboração de jogos de computador, que envolvem a utilização de estratégias por parte dos usuários.

Na seção 2, os enredos de narrativas são apresentados segundo a visão do Estruturalismo e é feita a ligação entre os conceitos de enredos de narrativas e de planos. Na seção 3, é descrito o sistema geral para reconhecimento e geração de planos a ser aplicado, segundo a metodologia desenvolvida, para se obter um tratamento automático dos enredos. Na seção 4, essa metodologia é apresentada, através da modelagem referente ao gênero dos contos de fadas, e são mostrados, também, os resultados obtidos com a aplicação do sistema descrito. Além disso, algumas extensões específicas ao sistema descrito na seção 3 são introduzidas para possibilitar o tratamento de enredos. Por fim, na seção 5, são apresentadas as conclusões e objetivos para trabalhos futuros.

2. Análise Estrutural e o Conceito de Enredos de Narrativas

Propp [Pro68] sugeriu que o gênero contos de fadas poderia ser caracterizado através de 31 funções, as quais são descritas em seu livro de modo semi-formal. Tomou como exemplos um certo número de contos de fadas russos, coletados por Afanas'ev [Afa45]. A cada um dos contos, associou uma sequência de aplicações dessas funções para denotar o esquema ou *estrutura* do conto. As estruturas proppianas correspondem, segundo essa visão, aos *enredos* das narrativas desenvolvidas nos textos dos contos.

Escolas posteriores ao Estruturalismo criticaram a caracterização dos enredos apenas por essas estruturas, entre outros motivos por adotar abordagem demasiadamente sintática, não levando em conta o conteúdo semântico que leva ao encadeamento entre as funções. Ainda assim, diversos pesquisadores reconhecem o valor das contribuições do Estruturalismo. Por outro lado, deve-se ressaltar sua especial adequação ao processamento automático; os conceitos de Propp, por exemplo, serviram de base a Rumelhart para introduzir o formalismo das gramáticas de histórias [Rum75].

Propp enumerou 7 *personagens*, entre os quais se distribui a execução das funções; note-se que algumas funções, que serão apresentadas a seguir, envolvem mais de um personagem, como o combate ("struggle"), por exemplo, que se dá entre o herói e o vilão. Os personagens são:

1. vilão (**villain**)
2. doador (**donor**)
3. auxiliar (**helper**)
4. vítima (**victim**)
5. mandante (**dispatcher**)
6. herói (**hero**)
7. falso herói (**false hero**)

Dando prosseguimento aos estudos de Propp, Greimas [Gre66] mostrou a conveniência de reduzir o número das funções, através do grupamento de algumas delas. Com a finalidade de tornar mais inteligível a estrutura dos contos de fadas, foi criada, para este trabalho, uma hierarquia de composição ("part_of") em diversos níveis. Essa hierarquia é relacionada a seguir, juntamente com a explicação de cada função. A raiz corresponde à totalidade de um conto e as folhas correspondem às 31 funções proppianas (em negrito e seguidas de número de ordem entre parênteses):

- tale
  - preliminaries
    - **absence** (1):
      - Um dos membros da família da vítima se ausenta de casa.
    - injunction
      - **interdiction** (2):
        - Uma proibição é feita à vítima.
      - **violation** (3):
        - A proibição é violada.
    - survey
      - **reconnaissance** (4):
        - O vilão busca informações sobre a vítima.
      - **delivery of information** (5):
        - A informação sobre a vítima é obtida.
    - trick
      - **fraud** (6):
        - O vilão tenta enganar a vítima para tomar posse dela ou de seus bens.
      - **complicity** (7):
        - A vítima, sem querer, colabora com o vilão.

- quest
    - **villainy or lack** (8):
        - O vilão causa dano a um dos membros da família.
    - voyage
        - mission
            - **mediation** (9):
                - O herói toma conhecimento do infortúnio.
            - **counteraction** (10):
                - O herói concorda em ir em busca da vítima.
            - **departure** (11):
                - O herói sai de casa.
        - preparation
            - **proof** (12):
                - O herói é testado por um doador para saber se merece receber o objeto mágico ou um auxiliar.
            - **reaction** (13):
                - O herói reage às ações do futuro doador.
            - **receipt of magical object** (14):
                - O herói adquire um objeto mágico ou um auxiliar.
        - **translocation** (15):
            - O herói é transportado para as vizinhanças de onde está a vítima.
    - confrontation
        - **struggle** (16):
            - O herói combate com o vilão.
        - **marking** (17):
            - O herói recebe uma marca.
        - **victory** (18):
            - O herói vence o vilão.
    - **liquidation** (19):
        - A vítima é libertada ou a causa de infelicidade é eliminada.
- homecoming
    - **return** (20):
        - O herói retorna.
    - capture
        - **pursuit** (21):
            - O herói é perseguido.
        - **rescue** (22):
            - O herói é resgatado da perseguição.
    - entrance
        - **arrival** (23):
            - O herói chega em casa sem ser reconhecido.
        - **pretentions** (24):
            - Um falso herói se apresenta reclamando a recompensa.
    - testing
        - **task** (25):
            - Uma tarefa difícil é proposta ao herói.
        - **solution** (26):
            - O herói resolve a tarefa.
- reckoning
    - revelation
        - **recognition** (27):
            - O herói é reconhecido.
        - **exposure** (28):
            - O vilão ou falso herói é exposto.
    - **transfiguration** (29):
        - O herói ganha uma nova aparência.
    - retribution

punishment (30):

O vilão é punido.

wedding (31):

O herói é recompensado, usualmente casando-se.

Propp indicou, para cada uma das 31 funções, diferentes formas que a função pode assumir, as quais denomina-se aqui como *casos*. Por exemplo, a *vilania* pode ser *seqüestro*, ou *mutilação*, ou *assassinato*, etc. Isso é representado através de uma hierarquia de especialização ("is_a"), que deixamos de reproduzir aqui por razões de espaço e porque coincidem com as formas indicadas por Propp (ver [Pro68, pgs. 149-155]). Na seção 4, trabalha-se com alguns operadores que são especializações dos operadores correspondentes às funções.

Propp afirmou que algumas funções podem não fazer parte do enredo de um determinado conto, mas insistiu que, quando presentes, a ordem de ocorrência é sempre a mesma - o que não é confirmado pelos exemplos fornecidos por ele. De fato, verifica-se que há alguma liberdade, e que portanto o ordenamento da ocorrência das funções é apenas uma *ordem parcial*. As possíveis utilizações de funções, e seus casos, com variadas ordens parciais, podem ser agrupadas e encaradas como enredos típicos do gênero literário contos de fadas.

O tratamento formal proporcionado por Propp é incompleto em pelo menos um aspecto: as funções não são definidas, mas apenas "explicadas" de modo intuitivo. Para defini-las, cabe introduzir alguns *predicados* (ver seção 4.1) que exprimem propriedades com as quais é descrito o *estado* do "mundo de aplicação", tais como o local em que se encontra um personagem, sua atitude em relação a outro personagem, etc. Graças à noção de estado, representado pelo conjunto de predicados que valem em certo momento, pode-se caracterizar uma função como um *operador* que transforma um estado s_i em outro estado s_j, através da *negação* de alguns dos predicados que valem em s_i e da *afirmação* de outros predicados válidos em s_j.

Dentro do formalismo STRIPS [FN71], usualmente adotado em sistemas de planejamento e que será utilizado aqui para modelar as funções como operadores, cada operador é definido através de suas *pré-condições* (referentes a s_i) e *efeitos* (referentes a s_j), ambos assumindo a forma de expressões lógicas envolvendo predicados. As pré-condições têm ainda a finalidade de impor *restrições* à aplicabilidade do operador: um operador só pode ser aplicado se em s_i valerem todas as pré-condições especificadas. Por exemplo, para que a *vilania* possa ser empreendida pelo *vilão*, uma pré-condição é *que a vítima esteja vulnerável*; um efeito, se a *vilania* toma a forma de *seqüestro*, é que a *vítima estará na mesma terra habitada pelo vilão*. A própria relação de ordem entre alguns operadores não precisa mais ser imposta arbitrariamente: passa a ser conseqüência do fato de um operador incluir entre seus efeitos alguma pré-condição de outro operador. Por exemplo, se *os pais da vítima se ausentam*, desse operador resulta como efeito que *a vítima se torna vulnerável* - uma das pré-condições para a *vilania*. Vê-se ainda que essas dependências entre operadores são uma das justificativas para que eles se encadeiem na formação de enredos. Em virtude de tudo isso, pode-se dizer que a especificação das funções proppianas através de operadores com pré-condições e efeitos introduz algum conteúdo *semântico* na análise estrutural.

Voltando agora a considerar enredos em sua totalidade, verifica-se que o efeito global de um enredo é transformar um *estado inicial*, s_0, em um *estado final*, s_f, passando através de estados intermediários alcançados através dos diversos operadores. Em geral, nos contos de fadas, o estado s_0 apresenta *a vítima* em situação relativamente satisfatória, a qual é rompida pela *vilania* e finalmente recomposta pela *ação do herói*, atingindo-se em s_f uma situação ainda melhor para a *vítima* assim como para o *herói*. Tais mudanças de estado podem ser encaradas como as motivações que levam os personagens a agir. Assim, o estado s_f pode ser visto como um *objetivo* a ser atingido. Tal entendimento nos conduz a caracterizar um enredo como um *plano* para, a partir da situação corrente s_0, atingir um objetivo s_f desejado. Faz então sentido apurar se um dado enredo (completo ou apenas parcial) se enquadra ou não em algum dos padrões próprios de um dado gênero literário (como contos de fadas), já que esse problema é análogo a reconhecer se um conjunto de observações sobre operadores executados por alguém indicam ou não que essa pessoa está tentando completar um plano típico bem conhecido (problema de *reconhecimento de* *planos*). Faz igualmente sentido modificar ou compor enredos dentro de um certo gênero literário, por ser um problema análogo a modificar ou gerar um plano para atingir um objetivo (problema de *geração de planos*).

A visão de enredos como planos voltados para objetivos permite que um terceiro aspecto se acrescente aos aspectos sintáticos e semânticos já introduzidos: o aspecto *pragmático*. Não se pretende, entretanto, que o uso estendido da análise estrutural aqui apresentado esgote as possibilidades de manipulação automática de enredos de narrativas. Estão sendo estudados resultados de outras teorias literárias [Web96, Eag94, LM87, Lim83] para tentar, em fase posterior deste projeto, utilizá-los na revisão e expansão do método aqui proposto. Uma das objeções levantadas em [Web96] é que a simples seqüência de acontecimentos não basta para criar um enredo de um texto literário, pois seqüências válidas dentro de uma certa estrutura podem ser tão óbvias que não despertam interesse algum no leitor. Por outro lado, informações podem ser inferidas pelo leitor a partir de um evento, e, para que esse evento possa ser devidamente tratado por um sistema automático, essas inferências devem ser levadas em consideração.

Concluindo esta seção, deve-se observar que o método não está limitado ao gênero contos de fadas. Aplica-se, em princípio, a qualquer gênero de narrativa literária [Jol72], devendo-se para isso introduzir elencos diferentes de operadores, predicados e personagens apropriados ao gênero. É além disso possível tratar enredos de narrativas não-literárias, tais como carreiras de empregados de uma empresa, acompanhamento de obras de engenharia, ou movimentos de contas bancárias.

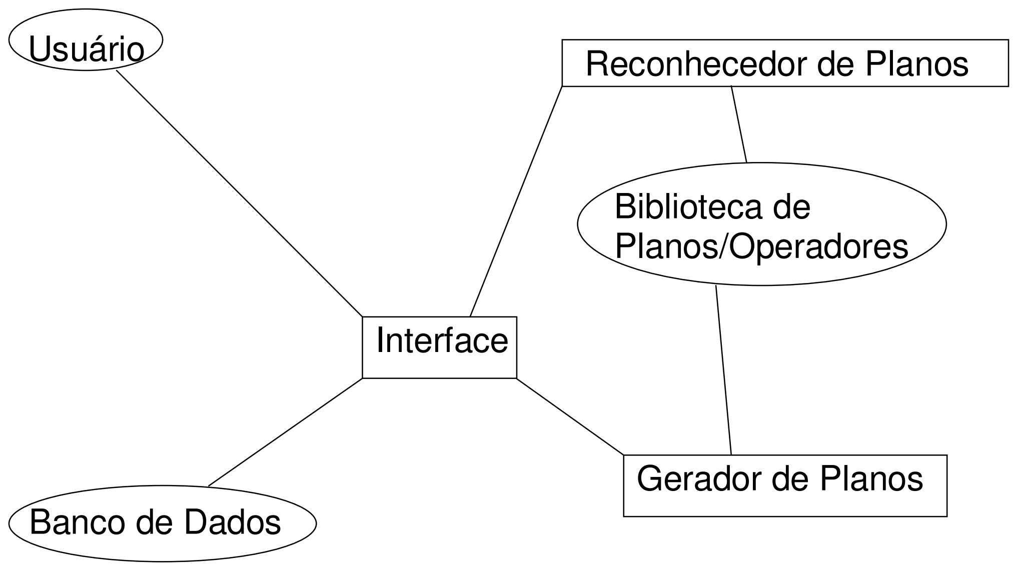
3. Reconhecimento e Geração de Planos

O Sistema para Reconhecimento e Geração de Planos (SRGP), adotado neste trabalho, procura integrar esses dois componentes para o apoio a tarefas que podem utilizá-los de forma explícita, como as realizadas por Sistemas de Apoio a Decisão, ou de forma implícita, como nas Interfaces Cooperativas de Bancos de Dados [Alc95]. Nesta seção, serão dados uma breve descrição do funcionamento do Sistema e os conceitos básicos necessários para se entender a sua aplicação no tratamento de enredos de narrativas. Maiores detalhes em relação ao funcionamento do SRGP, incluindo os algoritmos que utiliza e a descrição de sua aplicação no contexto de Interfaces Cooperativas de Bancos de Dados, podem ser obtidos em [CF96].

Um *sistema de reconhecimento de planos* tem como objetivo descobrir planos em que um conjunto de observações sobre operadores a serem executados se encaixa. A busca desses planos é feita em uma biblioteca de planos, definida estaticamente, que, em princípio, contém todos os planos que podem ser reconhecidos. Um exemplo típico de reconhecimento de planos pode ser a inferência, dentro do contexto de uma agência de turismo, que *um certo cliente deseja fazer uma viagem de avião para uma certa cidade*, porque ele *foi ao aeroporto* e *fez uma reserva em um hotel nessa cidade*. Um *sistema de geração de planos ou planejamento* procura definir um conjunto de operações a serem executadas, de forma que um certo objetivo seja alcançado. Um exemplo típico de planejamento pode ser extraído também dentro do contexto de uma agência de turismo. Um planejador pode gerar para o objetivo de *o cliente ter visitado uma determinada cidade*, a seqüência de operações a seguir: *comprar passagem de ônibus, reservar hotel, ir à rodoviária, embarcar no ônibus, visitar a cidade e retornar*.

O SRGP faz a integração entre reconhecimento e geração de planos quando um plano reconhecido não pode ser aplicado. Quando um plano é reconhecido, testa-se a sua aplicabilidade, verificando-se se os fatos requisitados como pré-condições para a sua aplicação são todos válidos. Se alguma pré-condição não é satisfeita, o plano reconhecido não é aplicável, mas pode-se aproveitar o fato de ter-se descoberto o objetivo implícito nas observações fornecidas. Consegue-se assim acionar o gerador de planos para adaptar o plano reconhecido ou para criar um plano novo que gere os mesmos efeitos. Sendo assim, o objetivo de *o cliente ter visitado uma determinada cidade*, citado anteriormente, pode ser inferido a partir do reconhecimento de um plano *viajar de avião* que não pode ser executado porque *o cliente não possui dinheiro para pagar a passagem de avião*, o que exige a elaboração de uma outra alternativa.

O SRGP é constituído de três partes: um reconhecedor, um planejador e uma interface que faz a ligação entre os dois subsistemas e se comunica com o usuário. A figura a seguir demonstra a estrutura e funcionamento do SRGP.



Tanto o gerador quanto o reconhecedor de planos compartilham a mesma biblioteca de planos/operadores. Para o SRGP, não existe diferença conceitual entre plano e operador, como ocorre em outros planejadores. Os planos possuem pré-condições e efeitos, mas contêm, além disso, uma complexidade correspondente aos seus suboperadores, parcialmente ordenados, que produzem, em conjunto, os efeitos globais do plano. A existência de suboperadores dá origem a uma *hierarquia de composição dos operadores*. Sendo assim, são denominados *operadores complexos* os operadores que possuem suboperadores e *operadores simples* os que não se decompõem. A sintaxe de definição dos planos/operadores é apresentada no Apêndice I.

O usuário fornece, através da interface, uma lista de *observações* (instâncias de execução de operadores) correspondentes a ações para se atingir um determinado objetivo, que consiste nos efeitos do plano a ser detectado. Com base nessas observações, é aplicado um algoritmo de reconhecimento de planos que consulta uma biblioteca criada previamente. Uma vez reconhecido um plano, a interface verifica se as respectivas pré-condições são fatos válidos na base de dados e, portanto, o plano reconhecido pode ser executado. Se puder, o plano é informado ao usuário, caso contrário são indicadas, além do plano reconhecido, as pré-condições não satisfeitas. Nesse momento, o usuário tem a opção de solicitar que o gerador de planos seja acionado, seja para adaptar o plano reconhecido ou para gerar um novo plano com os mesmos efeitos.

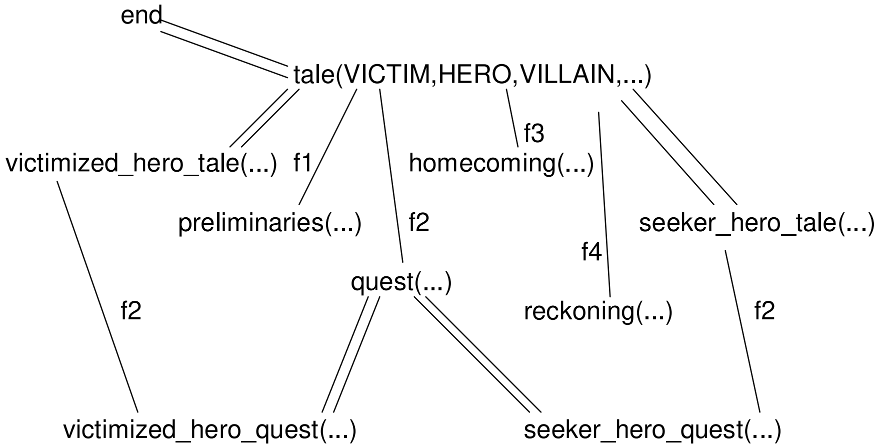
A hierarquia de composição, já citada, permite decompor um operador como *viajar de avião para a cidade tal* nos suboperadores *comprar passagem de avião, reservar hotel, ir ao aeroporto, embarcar, visitar e retornar*. Os operadores são organizados também segundo uma *hierarquia de abstração*, ao longo da qual os operadores herdam a composição de operadores mais abstratos, podendo ainda acrescentar operadores com novos atributos ou substituir o operador correspondente a um certo atributo. Um operador *viajar de avião* pode herdar, em sua composição, o suboperador *visitar* de um operador mais abstrato *viajar*, substituir o suboperador *comprar passagem* pelo suboperador *comprar passagem de avião* e acrescentar o suboperador *ir ao aeroporto*. A relação inversa da abstração é chamada de especialização. A hierarquia de abstrações é sempre uma floresta, ou seja, não é admitida herança múltipla. A hierarquia de composição, contudo, não é obrigatoriamente uma floresta pois um determinado operador pode ser componente de vários operadores distintos.

Além das hierarquias citadas, que são utilizadas no reconhecimento de planos, há uma hierarquia entre os predicados utilizados nas pré-condições. Essa hierarquia, utilizada pelo gerador de planos, permite que o planejamento seja feito segundo os níveis de importância atribuídos aos predicados. O planejador procura fazer com que todas as pré-condições de um certo nível, contidas em operadores do plano, estejam satisfeitas antes de levar em conta pré-condições do nível inferior. Como exemplo, pode-se citar, no caso da *agência de turismo*, a possibilidade de se estabelecer uma hierarquia de predicados onde o predicado *estar_em(CLIENTE,LUGAR)* tem prioridade sobre o predicado *ter_reserva_em_hotel(PESSOA)*. Nesse caso, é possível que o operador *reservar hotel* seja inserido em um plano somente depois que foram inseridos todos os operadores que implicam no *deslocamento do cliente*. Consegue-se assim fazer um planejamento com níveis sucessivos de refinamento.

O SRGP foi integralmente implementado utilizando-se o ARITY PROLOG [Ari88]. Os algoritmos escolhidos correspondem ao estado da arte quanto ao reconhecimento e geração de planos, conforme levantamento efetuado em [Alc95]. O reconhecedor de planos adota o algoritmo especificado por Kautz em [Kau91]. O gerador de planos é uma adaptação em Prolog do ABTWEAK [YTW92], originalmente implementado em Lisp por Woods [Woo91]. O ABTWEAK estende as características de comprometimento mínimo de planejadores não-lineares, tais como o TWEAK [Cha87], com as de planejadores hierárquicos, como o ABSTRIPS [Sac74]. Como foi dito antes, os algoritmos de reconhecimento e geração utilizam uma única biblioteca de planos/operadores, o que exigiu compatibilizar o conceito de plano, utilizado no reconhecimento, que não se preocupa com pré-condições e efeitos, com o conceito de operador, utilizado na geração de planos, que não utiliza hierarquias de composição nem de abstração de operadores. Além disso, o gerador passou a admitir que o planejamento fosse iniciado a partir de um plano da biblioteca a ser adaptado. O algoritmo de reconhecimento de Kautz foi estendido para admitir que os planos/operadores possuíssem, além de uma identificação, um número qualquer de variáveis, o que permite parametrizá-los.

Do mesmo modo que para o caso da *agência de turismo*, é possível construir, como será demonstrado na seção 4, uma biblioteca que contém os operadores correspondentes às funções proppianas e aos enredos típicos de contos de fadas. A fim de permitir exemplificar em maiores detalhes o reconhecimento de planos, a interpretação dos resultados desse reconhecimento e a geração de planos, será introduzido ainda nesta seção um contexto simplificado correspondente aos contos de fadas. Nesse contexto, um conto (*tale*) possui quatro suboperadores básicos: as ações preliminares (*preliminaries*; a aventura principal do herói, iniciada pela vilania (*quest*); as dificuldades no retorno do herói para casa (*homecoming*); e as ações que levam o herói a ser reconhecido e recompensado (*reckoning*). Foram identificados dois tipos de conto: *seeker_hero_tale* e *victimized_hero_tale* que são especializações de *tale*. A primeira especialização corresponde aos contos onde o herói é um personagem distinto da vítima e a segunda aos contos onde o herói é a própria vítima. Como a busca (*quest*) feita pelo herói é muito diferente nos dois casos, o suboperador *quest* foi redefinido, tanto para *seeker_hero_tale* como para *victimized_hero_tale,* com os valores respectivos *seeker_hero_quest* e *victimized_hero_quest.*

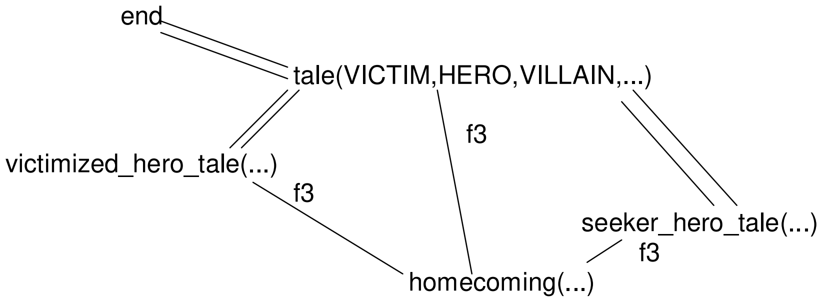
A figura a seguir mostra a hierarquia de composição (arcos simples, com o nome do atributo correspondente ao suboperador) e de abstração (arcos duplos) dos planos/operadores para este exemplo.



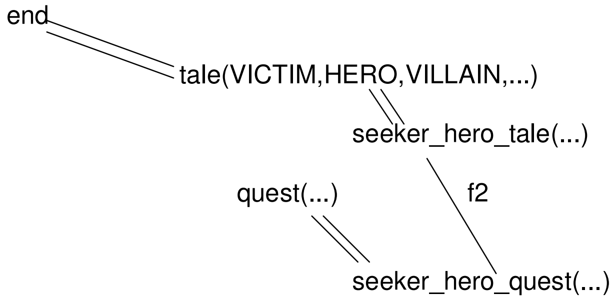
3.1. Reconhecimento de Planos

O reconhecimento de planos, conforme especificado no algoritmo de Kautz, consiste em gerar grafos de explicação ("explanation graphs") das observações fornecidas. Esses grafos contêm, como nós, os planos/operadores e, como arcos, arcos de composição e de abstração. Existe um plano denominado *end* que é a abstração de todos os planos cujos efeitos podem ser reconhecidos como objetivo final. O grafo de explicação de uma observação descreve de que forma o operador correspondente a essa observação pode ser enquadrado como parte de um ou mais planos que sejam especializações) do plano *end*.

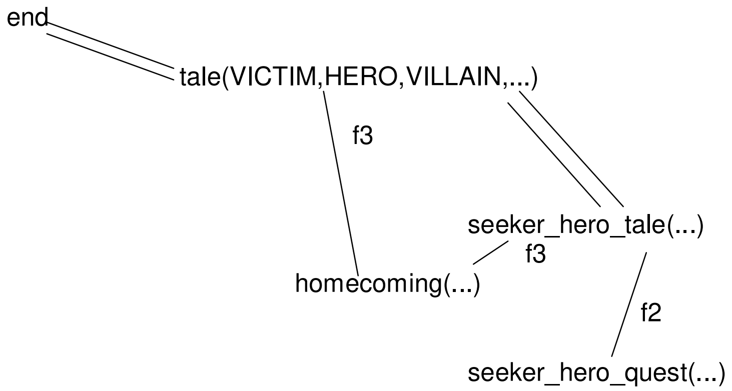
No exemplo, se for fornecido, como observação, um operador *homecoming*, o sistema verificará que esse operador é componente do operador *tale* que é uma especialização de *end*. Como *tale* possui as especializações *victimized_hero_tale* e *seeker_hero_tale*, esses planos/operadores farão parte do grafo e serão os planos mais especializados, os quais podem ser reconhecidos como possíveis geradores do objetivo. A seguir é apresentado o grafo de explicação para *homecoming*.



Para a observação *seeker_hero_quest* é montado o grafo a seguir, onde o único plano que pode ser reconhecido como possível gerador do objetivo é *seeker_hero_tale*.



Após se montar um grafo para cada observação, os grafos são unificados, para que se obtenha um grafo final que contém todas as especializações de *end* onde todas as observações fornecidas se encaixam, ou seja, todas elas são operadores a serem executados. A seguir é apresentado o grafo resultante da unificação dos dois grafos anteriores.



3.2. Interpretação dos Resultados do Reconhecimento

Após a obtenção de um grafo que explica um conjunto de observações fornecido pelo usuário, tem-se como tarefa identificar nesse grafo quais são os planos que correspondem a intenções finais. O SRGP admite, para isso, que podem ser executados *planos concretos*, ou seja, especializações do operador *end* que, por sua vez, não possuem especializações. No grafo resultante apresentado na subseção 3.1, é plano concreto apenas o operador *seeker_hero_tale*.

Após a identificação de um plano concreto, testam-se as suas pré-condições. Para que uma pré-condição seja considerada válida, é necessário que o seu valor seja um fato, básico ou dedutível, do banco de dados. Se todas as pré-condições são válidas, o plano pode ser executado e o que resta fazer é comunicar o resultado ao usuário. Se alguma pré-condição falha, então o gerador de planos pode ser útil para se chegar a uma solução satisfatória.

A primeira opção de utilização do gerador é tentar adaptar o plano reconhecido, de forma que, com a inclusão de alguns operadores e/ou restrições, os efeitos primários do plano reconhecido possam ser obtidos. A segunda opção é gerar um plano absolutamente novo que tem como objetivo gerar os efeitos primários do plano reconhecido. A escolha entre geração e adaptação é deixada sempre ao usuário.

3.3. Geração de Planos

Como já foi mencionado, a geração de planos é feita através de uma versão modificada do planejador hierárquico não-linear ABTWEAK. A modificação consistiu em fazer com que o planejamento pudesse se iniciar tendo como ponto de partida um plano da biblioteca a ser adaptado. Foram feitas também modificações na forma de consulta ao conteúdo dos operadores da biblioteca, pois a estrutura desses operadores passou a conter tanto definições necessárias para o reconhecimento quanto para a geração de planos.

Um plano ABTWEAK é definido por operadores que não precisam estar completamente ordenados em uma seqüência. Durante o planejamento, são definidas relações de ordem apenas quando se torna necessário. Sendo assim, é possível que, para a satisfação de uma meta, o ABTWEAK forneça uma ordem meramente parcial. Além dos operadores e das relações de ordem, fazem parte do plano restrições que obrigam que duas variáveis referenciem um mesmo objeto (correferência) ou que não referenciem um mesmo objeto (impossibilidade de correferência).

Para exemplificar a definição de ordem parcial e de restrições de referência das variáveis, pode-se retornar ao exemplo da subseção 3.1. Os operadores *seeker_hero_quest*, *homecoming* e *reckoning* são componentes do plano *seeker_hero_tale.* Pode não existir nenhuma restrição de ordem relacionando *homecoming* com *reckoning*, permitindo-se que o herói seja reconhecido antes ou depois do retorno para casa, mas deve-se especificar que os dois operadores ocorram após o operador *seeker_hero_quest,* pois o herói não pode retornar para casa nem ser reconhecido antes de efetuar a busca para salvar a vítima. Como exemplo de restrição de impossibilidade de correferência de variáveis a um objeto, pode-se citar a restrição de que, no plano *seeker_hero_tale(VICTIM, HERO, VILLAIN),* as variáveis *VICTIM* e *HERO* tenham valores diferentes.

O planejamento consiste em verificar quais pré-condições de operadores contidos no plano não estão satisfeitas e adicionar novos operadores e/ou restrições, de forma que isto aconteça. A seleção de pré-condições a serem satisfeitas é feita, como foi dito antes, segundo uma hierarquia. A hierarquia de pré-condições é implementada através da atribuição de níveis de criticalidade a cada predicado. O planejamento procura fazer com que todas as pré-condições de nível mais alto sejam satisfeitas, antes de serem consideradas pré-condições de um nível inferior. O processo termina quando são satisfeitas as pré-condições do nível zero. Para que esse processo possa ser efetuado, cada plano possui uma indicação do nível de criticalidade já resolvido por ele. Como exemplo de hierarquia de pré-condições, pode-se estabelecer que as pré-condições que descrevem o *estado do herói* ao longo do enredo tenham prioridade sobre as demais. Sendo assim, é feito um planejamento completo relativo à satisfação das pré-condições desse tipo, antes que as pré-condições dos demais tipos sejam consideradas.

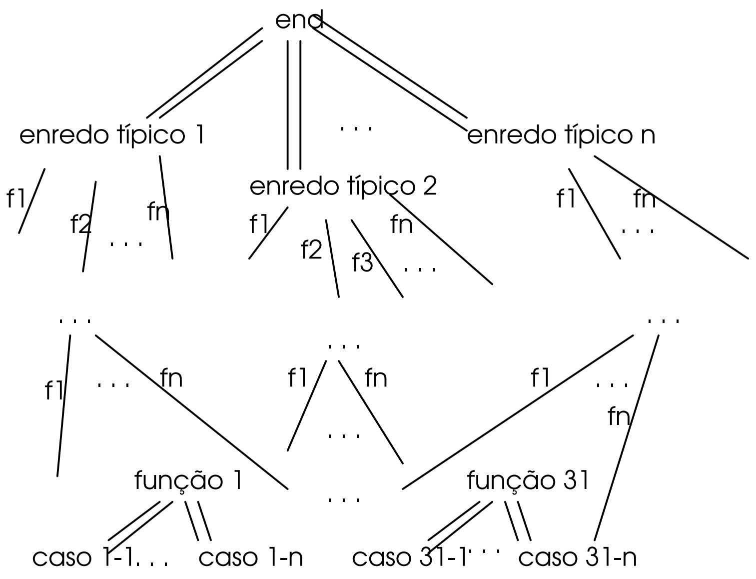
A cada operador é associado um custo que deve refletir os recursos a serem utilizados para a sua execução bem como a prioridade a ser dada na sua utilização, uma vez que o planejamento procura gerar planos com o menor custo possível. Durante o processo, são geradas todas as alternativas para que uma dada pré-condição seja satisfeita. Essas alternativas dão origem a vários planos sucessores do plano original, que são incluídos em uma fila de prioridades, ordenada segundo o custo total estimado para se atingir a solução. Para se fazer essa estimativa, é necessária uma função heurística, que atribui um custo menor para a conclusão de planos com nível de criticalidade mais baixo, ou seja, que já tiveram as pré-condições com criticalidade mais alta resolvidas. Deve-se ressaltar que, uma vez mostrado um plano como resultado do planejamento, o usuário pode solicitar que ele seja descartado e o planejamento continue, levando em conta os demais planos candidatos contidos na fila.

4. Modelagem dos Enredos

4.1. Definição de Enredos Típicos

Para a modelagem exemplo, referente aos contos de fadas, foram utilizadas as funções definidas por Propp. A cada uma das 31 funções foi associado um operador com pré-condições e efeitos especificados. Nessa modelagem foram utilizadas tanto a hierarquia de composição de operadores como a de abstração. Os operadores correspondentes às funções foram agrupados, através da hierarquia de composição, em operadores mais complexos. No nível mais alto dessa hierarquia de composição, foram definidos operadores/planos correspondentes aos *enredos típicos* que, por sua vez, são especializações do plano abstrato *end*. Essas especializações do plano *end* constituem portanto o conjunto de enredos que podem ser reconhecidos pelo reconhecedor de planos.

Cada uma das 31 funções possui, em geral, diversos casos. Os casos foram implementados como especializações do operador correspondente a cada função. Quando um enredo típico tem como componente um operador correspondente a uma função, ele está aceitando qualquer uma das especializações desse operador (casos). Cabe ressaltar que, na maior parte dos enredos, são utilizados os casos específicos das funções e não as funções diretamente. Essa organização pode ser observada na figura a seguir.



As funções determinadas por Propp são realmente funções no sentido matemático, porque mapeiam um estado que descreve a situação dos personagens em outro. Propp, porém, não se preocupou em criar predicados para descrever formalmente esses estados. Para que novos enredos pudessem ser gerados ou adaptados pelo gerador de planos, tornou-se necessário definir um conjunto de predicados correspondentes às pré-condições e efeitos dos operadores. Tais predicados deveriam permitir que o enredo de qualquer conto catalogado por Propp pudesse ser gerado automaticamente pelo gerador. Por outro lado, deveriam ser suficientemente gerais ou "primitivos" de forma a permitir o melhor reaproveitamento dos operadores em outros enredos a serem inseridos na biblioteca ou gerados automaticamente. Em virtude de não terem sido modelados ainda todos os enredos típicos de contos de fadas catalogados, não se pode garantir que esses predicados sejam os únicos necessários, mas, além de terem sido suficientes para a geração dos enredos típicos modelados, sua generalidade sugere que, se for necessário aumentar o conjunto, esse aumento não deverá ser significativo. O estado inicial para o planejamento é definido por uma base de dados que contém os fatos válidos inicialmente, expressos através de cláusulas instanciando esses predicados. Foram definidos ao todo 7 predicados, listados a seguir:

- **character(PERS,ROLE)**, onde ROLE é o papel do personagem PERS na história. Ex: hero, villain.
- **place(PERS,LOC)**, onde LOC é o local em que PERS se encontra. Ex: home, intermediate_land, opposite_land.
- **victim_state(PERS,STATE)**, onde STATE representa um estado típico da vítima PERS. Ex: young, interdicted, vulnerability_possible, vulnerable, suffering, dead.
- **hero_state(PERS,STATE)**, onde STATE representa um estado típico do herói PERS. Ex: powerful, victorious, rich, strong, ready_to_depart, motivated_to_seek, fighting, trying_to_solve_task.
- **villain_state(PERS,STATE)**, onde STATE representa um estado típico do vilão PERS. Ex: humiliated, exposed, punished, deserving_punishment, punishment_possible.
- **assistant_state(PERS,STATE)**, onde STATE representa um estado típico de um personagem auxiliar PERS. Ex: powerful, single.
- **feels(PERS1,FEELING,PERS2)**, onde FEELING representa o sentimento de PERS1 em relação a PERS2. Ex: love, fear, trust.

Ao se estabelecerem os possíveis valores dos argumentos dos predicados listados acima, procurou-se evitar o aparecimento de sinônimos. A razão disto é a mesma da necessidade de se restringir a escolha de predicados, ou seja, fazer com que mais enredos possam ser inseridos na biblioteca ou gerados automaticamente, mantendo-se um mesmo conjunto de operadores básicos. Se, para cada enredo típico a ser inserido, fossem utilizados valores diferentes para esses argumentos, não haveria, em absoluto, o reaproveitamento de operadores já criados para outros enredos típicos. Evitou-se também a utilização de antônimos, uma vez que eles podem ser expressos pela negação de um valor já existente.

Para que o planejamento possa ser feito eficientemente, torna-se fundamental estabelecer uma hierarquia de pré-condições adequada, ou seja, definir uma ordem de importância das pré-condições no decorrer dos enredos. Intuitivamente, pelo fato de os enredos terem a presença do herói e da vítima em grande número[1] de operadores e pelo fato de o papel dos personagens ser consultado com muita freqüência, pode-se admitir que os predicados *character*, *victim_state* e *hero_state* têm importância maior que os demais. Sendo assim, experimentou-se alternar a importância dada a cada um dos três predicados através da atribuição de níveis de criticalidade que variavam de 3 a 1. Os demais predicados receberam em todos os casos o nível de criticalidade 0. As hierarquias foram aplicadas na geração de cada um dos três enredos típicos utilizados para validar o protótipo, obtendo-se como resultado [1] a tabela a seguir.

Hierararquia ***Enredo***	***character hero_state victim_state***	***character victim_state hero_state***	***hero_state character victim_state***	***hero_state victim_state character***	***victim_state character hero_state***	***victim_state hero_state character***
Rapto	99 s	197 s	101 s	104 s	200 s	102 s
Expulsão pela Água	231 s	96 s	57 s	57 s	97 s	57 s
Assassinato Simples	9 s	4 s	8 s	4 s	5 s	4 s

Os resultados mostram o tempo necessário para gerar cada um dos enredos em cada hierarquia. Pode-se observar que a hierarquia "*hero_state* (criticalidade 3), *victim_state* (criticalidade 2), *character* (criticalidade 1)" e a hierarquia "*victim_state* (criticalidade 3), *hero_state* (criticalidade 2), *character* (criticalidade 1)" obtiveram bons resultados na geração de todos os enredos.

Outra questão de modelagem também importante refere-se à atribuição de custos aos operadores. Uma vez que os enredos típicos são também operadores, embora complexos, eles tanto podem como devem ser levados em conta durante o planejamento. Cada operador correspondente a uma função (ou a um caso seu) recebeu um custo genérico de 10. Se for atribuído aos operadores dos enredos o mesmo valor 10, eles ficam privilegiados porque conseguem gerar o efeito combinado de vários operadores com o custo de um único operador, não se conseguindo portanto gerar enredos constituídos essencialmente por operadores básicos. Se, por outro lado, os operadores dos enredos tiverem como custo um valor muito alto, tal como a soma dos custos de seus suboperadores, eles é que perdem a competitividade e não conseguem ser levados em conta durante o planejamento. Através da utilização de vários custos diferentes para os operadores dos enredos típicos, chegou-se à conclusão que a faixa em que eles se tornam competitivos, mas não eliminam a competitividade das alternativas baseadas essencialmente em operadores básicos, oscila entre 1/4 e 1/2 do somatório dos custos de seus componentes. Esses valores significam que um plano que mistura um operador complexo, com *N* suboperadores, e 3/4 de N ou 1/2 de N operadores simples tem prioridade sobre um plano com *N* operadores simples.

Também para a eficiência do planejamento, torna-se importante criar uma boa função heurística que forneça uma estimativa do custo necessário para um plano satisfatório ser gerado, tendo como base um determinado plano resolvido até um certo nível de criticalidade. O resultado dessa função, para cada plano que está sendo gerado simultaneamente, somado ao seu custo atual respectivo, é utilizado para ordenar a lista de planos a serem levados em conta no planejamento. Se a função heurística fornecesse sempre um custo nulo, estar-se-ia fazendo uma busca apenas em largura, na qual a hierarquia de pré-condições não teria efeito. Se, por outro lado, a função heurística atribuísse uma vantagem muito grande para cada nível de criticalidade resolvido, estar-se-ia

[1] O valores transcritos foram obtidos em um computador Pentium 166 MHz com Windows 95 e executando o Arity Prolog 6.0 em modo interpretado.

fazendo essencialmente uma busca em profundidade. A função heurística utilizada foi estabelecida admitindo-se, com base nos enredos típicos utilizados no protótipo, que um plano que está sendo resolvido nos níveis de criticalidade 3, 2, 1 ou 0 deverá agregar provavelmente operadores com um custo total de 120, 80, 40 ou 0, respectivamente, para a geração de um plano sucessor que seja satisfatório. Essa estimativa foi feita admitindo-se um número médio de doze operadores em cada plano gerado e ainda a inclusão de quatro operadores nos níveis de criticalidade 3, 2 e 1 e nenhum no nível 0.

4.2. Exemplos de Enredos Típicos

Foram inseridos três enredos típicos na biblioteca, baseados em três histórias distintas, descritas brevemente a seguir:

1. Rapto (*kidnapping(VICTIM,HERO,VILLAIN,DISPATCHER)*): Um tsar tem três filhas. As filhas saem para passear (op. *absence_of_younger_people(VICTIM)*) e demoram-se no jardim (op. i*nterdiction_violated(VICTIM)*). Um dragão as rapta (op. *kidnapping_of_a_person(VICTIM,VILLAIN)*). É feito um pedido de socorro (op. *call_for_help(HERO, DISPATCHER)*). Três heróis saem à procura das moças (ops. *counteraction(HERO)* e *departure_of_seeker_hero(HERO)*). Ocorrem batalhas com o dragão, vencidas pelos heróis (ops. *fight_in_an_open_field(HERO,VILLAIN)* e *victory_in_open_battle(HERO, VILLAIN)*). As moças são resgatadas (op. *liquidation_as_direct_result(VICTIM,HERO)*). Ocorre o retorno (op. *return(HERO)*) e os heróis são recompensados (op. *reward(HERO)*). No caso desse conto específico, *VICTIM=daughters*, *HERO=heroes*, *VILLAIN=dragon* e *DISPATCHER=tsar*.
2. Expulsão pela Água (*expulsion_by_water(HERO,VILLAIN)*): Um mercador tem mulher e filho. Um rouxinol prevê a humilhação dos pais pelo filho. Os pais colocam o filho adormecido em um barco e o atiram ao mar (ops. *casting_into_the_sea(HERO,VILLAIN)* e *departure_of_victimized_hero(HERO)*). Um barqueiro salva o menino e o leva com ele (op. *hero_is_carried(HERO)*). Eles chegam a *Xvalynsk*. O tsar propõe ao herói (o filho) uma tarefa (op. *task(HERO)*). O herói soluciona a tarefa (op. *solution(HERO)*). O herói se casa com a filha do tsar (op. *wedding(HERO,IMPORTANT_PERSON)*). O herói retorna (op. *return(HERO)*) e é reconhecido pelos pais em uma hospedaria (op. *recognition(HERO,VILLAIN)*). No caso desse conto específico, *HERO=son*, *VILLAIN=parents* e *IMPORTANT_PERSON=tsar_s_daughter*.
3. Assassinato Simples (*simple_murder(VICTIM,VILLAIN)*). Um sacerdote e sua mulher têm uma filha, *Alenuska*, e um filho, *Ivanuska*. *Alenuska* vai ao bosque para buscar grãos. Sua mãe ordena que leve seu irmão junto (ops. *order_or_command(VICTIM)*, *absence_of_younger_people(VICTIM)* e *order_carried_out(VICTIM)*). *Ivanuska* colhe mais grãos do que *Alenuska* (motivação para a vilania). *Alenuska* se oferece para catar piolhos em *Ivanuska* (op. *persuasions_of_a_villain(VICTIM,VILLAIN)*). *Ivanuska* adormece (op. *reacts_to_persuasion(VICTIM,VILLAIN)*). *Alenuska* mata seu irmão (op. *murder(VICTIM,VILLAIN)*). Um fino caniço nasce sobre a sepultura e é utilizado por um pastor para fazer uma gaita (op. *out_of_the_earth_receipt*). Quando a gaita é tocada, o assassinato é revelado (op. *exposure(VILLAIN)*). Os pais, ao tomarem conhecimento do fato, expulsam a filha de casa (op. *punishment(VILLAIN)*). No caso desse conto específico, *VICTIM=ivanuska* e *VILLAIN=alenuska*.

Os enredos típicos criados, embora tenham-se baseado em contos específicos, podem-se adequar a outros contos, porque os operadores possuem argumentos que podem ser instanciados diversamente durante os processos de reconhecimento e geração de planos. O enredo das histórias acima é reconhecido ou pode ser gerado exatamente conforme especificado quando o estado inicial dos personagens é descrito de forma compatível no banco de dados. Para cada um dos contos, foram criados bancos de dados que contêm os seguintes fatos iniciais:

A) Para o conto do Rapto: *victim_state(daughters, young)*, *character(dragon, villain)*, *feels(tsar, love, daughters)*, *character(heroes, hero)*, *hero_state(heroes, strong)*, *place(heroes, home)* e *victim_state(daughters, interdicted)*.

B) Para o conto da Expulsão pela Água: *assistant_state(tsar_s_daughter, powerful)*, *assistant_state(tsar_s_daughter, single)*, *place(son, home)*, *victim_state(son, vulnerable)*, *character(parents, villain)*, *hero_state(son, strong)* e *feels(parents, fear, son)*.

C) Para o conto do Assassinato Simples: *victim_state(ivanuska, young)* e *character(alenuska, villain)*.

4.3. Reconhecimento de Enredos

A primeira aplicação do SRGP ao contexto dos contos procurou verificar simplesmente o reconhecimento de planos aplicáveis, ou seja, colocou-se, como banco de dados corrente, o estado inicial necessário para a execução dos planos correspondentes a cada um dos contos usados para criar os enredos típicos. Nesse caso, quando são fornecidos como observações operadores contidos apenas em um dos enredos, esse enredo é reconhecido pelo SRGP e informa-se que o plano correspondente é aplicável. Esse tipo de aplicação pode ser verificado nos seguintes exemplos:

B. D.	***Observações***	***Plano Reconhecido***
A	absence_of_younger_people(daughters); return(heroes)	kidnapping(daughters,heroes, dragon,tsar)
B	return(son); recognition(son,parents)	expulsion_by_water(son,parents)
C	absence_of_younger_people(ivanuska); murder(ivanuska,alenuska)	simple_murder(ivanuska,alenuska)

Deve-se notar que alguns argumentos dos planos reconhecidos não aparecem explicitamente nas observações, tais como os personagens *dragon* e *tsar* no plano *kidnapping*. Esses argumentos são instanciados na consulta ao banco de dados que é feita para se verificar a satisfação das pré-condições do plano *character(VILLAIN,villain)* e *feels(DISPATCHER,love,daughters)*.

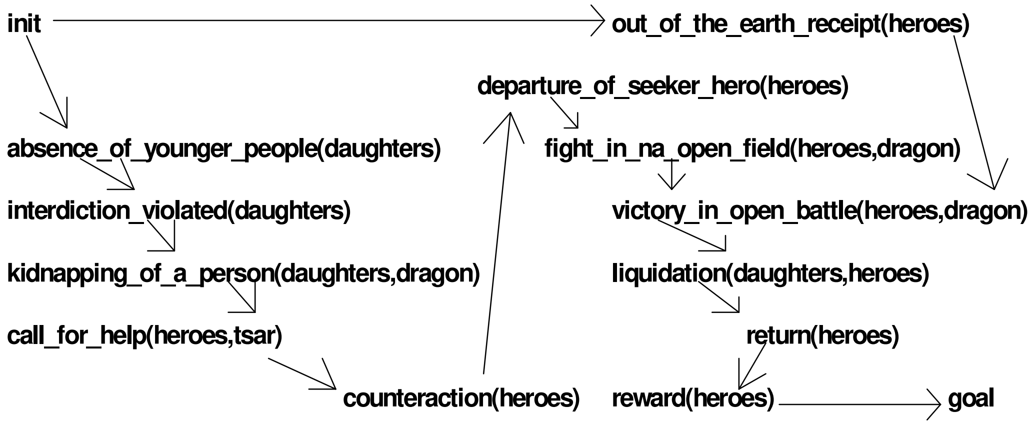
Procurou-se verificar quais planos eram reconhecidos quando as observações tratavam de operações comuns a mais de um conto. Pode-se observar que o operador *absence_of_younger_people* é comum tanto ao conto *kidnapping* quanto ao conto *simple_murder* enquanto que o operador *return* é comum tanto ao conto *kidnapping* quanto ao conto *expulsion_by_water*. Como os resultados dependem do banco de dados corrente, a tabela a seguir os apresenta nas variadas situações. Quando o banco de dados não é exatamente o relativo ao conto em cima do qual o enredo típico reconhecido foi criado, podem aparecer pré-condições não satisfeitas inicialmente, o que indica que o plano reconhecido não é automaticamente aplicável. As observações foram fornecidas com variáveis nos seus argumentos para que houvesse maior flexibilidade no reconhecimento dos planos.

B. D.	Observação	Plano Reconhecido	Inst. de Variáveis	Pré-cond. não Satisfeitas
A	absence_of_younger_people(X)	kidnapping(daughters,heroes, dragon,tsar)	X=daughters	Nenhuma
A	absence_of_younger_people(X)	simple_murder(daughters, dragon)	X=daughters	Nenhuma
C	absence_of_younger_people(X)	kidnapping(ivanuska,Y, alenuska,Z)	X=ivanuska; Y=?; Z=?	feels(Z,love,ivanuska); character(Y,hero); hero_state(Y,strong); place(Y,home); victim_state(ivanuska, interdicted)
C	absence_of_younger_people(X)	simple_murder(ivanuska, alenuska)	X=ivanuska	Nenhuma
A	return(X)	kidnapping(daughters,heroes, dragon,tsar)	X=heroes	Nenhuma
A	return(X)	expulsion_by_water(heroes,dragon)	X=heroes	assistant_state(Y,powerful); assistant_state(Y,single); victim_state(heroes, vulnerable); feels(dragon,fear,heroes)
B	return(X)	kidnapping(Y, son, parents, Z)	X=son; Y=?; Z=?	victim_state(Y,young); feels(Z,love,Y); character(son,hero); victim_state(Y,interdicted)
B	return(X)	expulsion_by_water(son,parents)	X=heroes	Nenhuma

Deve-se notar que algumas variáveis constantes tanto na identificação dos planos como em suas pré-condições podem continuar sem serem instanciadas, em conseqüência de o banco de dados não possuir informação suficiente para instanciá-las.

4.4. Adaptação de Enredos

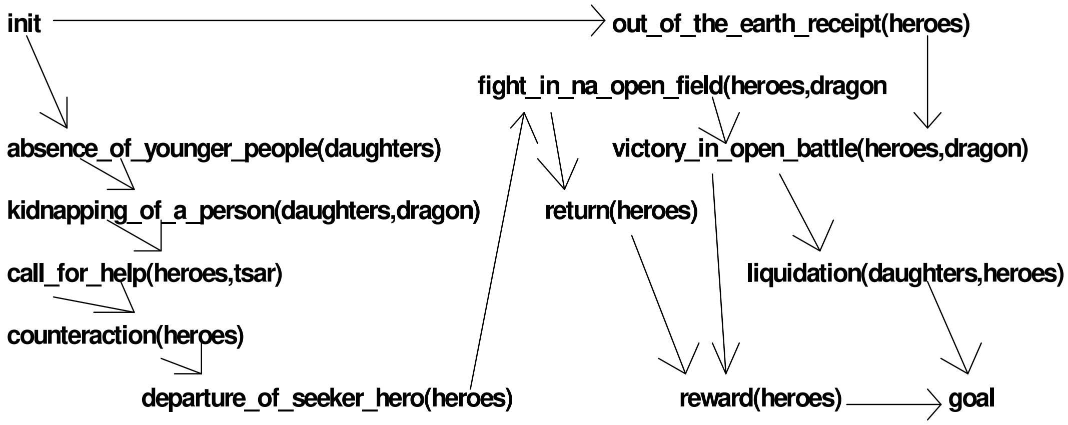
Quando um plano reconhecido não pode ser aplicado, mas a modificação da sua estrutura para atender a alguma pré-condição não satisfeita parece plausível para o usuário, pode ser solicitada a sua adaptação, de forma que, partindo-se do estado inicial, consiga-se, adicionando-se novos operadores e restrições (temporais e/ou de correferência de variáveis), atingir o objetivo especificado como efeito primário do plano reconhecido. Para se testar a adaptação dos planos, foi feita a experiência de se retirar do banco de dados **A** o fato *hero_state(heroes, strong)*. Dessa forma, quando são fornecidos, como observações, os operadores *absence_of_younger_people(daughters)* e *return(heroes)*, então o plano reconhecido *kidnapping(daughters,heroes,dragon,tsar)* não pode ser aplicado de imediato. Isto acontece porque, conforme especificado como pré-condição no operador *victory_in_open_battle*, sem que os heróis sejam fortes (*hero_state(heroes, strong)*), eles não têm capacidade para vencer o dragão. O fato é informado ao usuário, juntamente com a indicação de que os efeitos primários do plano reconhecido são tornar o herói rico (*hero_state(heroes,rich)*) e fazer com que a vítima não esteja sofrendo (*not(victim_state(daughters,suffering))*). Quando é feita a solicitação de adaptação do plano reconhecido, está sendo solicitado, na verdade, que seja gerado um plano que preserve, no mínimo, os operadores de *kidnapping* e gere os seus efeitos primários. O grafo a seguir mostra o plano resultante para a adaptação solicitada. Os arcos mostram a ordem parcial entre os operadores.



No grafo acima, pode-se verificar que foi incluído apenas o operador *out_of_the_earth_receipt(heroes)*, o qual estabelece a pré-condição *hero_state(heroes,strong)*. Deve-se notar que, como o planejador não é linear, o novo operador pode ser executado em qualquer momento antes do operador *victory_in_open_battle(heroes,dragon)*. Essa restrição temporal significa que o herói deve se tornar forte antes da vitória. De forma semelhante, em outros casos de adaptações, podem ser inseridos vários operadores tanto no meio da seqüência do plano original quanto criando paralelismos.

4.5. Geração de Enredos

Quando um plano reconhecido não é aplicável, oferece-se, além da possibilidade de adaptá-lo, a alternativa de se gerar um novo plano que produza os mesmos efeitos primários. Para o caso do enredo *kidnapping*, com o banco de dados **A**, sem o fato *hero_state(heroes,strong)*, e a meta do planejamento (correspondente aos efeitos primários) com os fatos *hero_state(heroes,rich)* e *not(victim_state(daughters,suffering))* temos a seguinte opção de plano gerada:



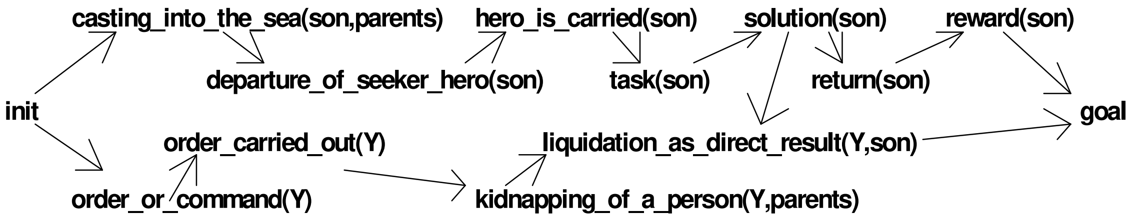
O plano gerado difere ligeiramente do plano típico. Primeiramente, pode-se observar que o operador *interdiction_violated(daughters)* não está presente. Isso ocorre porque seu principal efeito é tornar a vítima vulnerável, inserindo o fato *victim_state(vulnerable,daughters)*, papel esse já desempenhado pelo operador *absence_of_younger_people(daughters)*. A presença dos dois operadores é redundante. Nas outras alternativas geradas pelo planejador, o operador *absence_of_younger_people(daughters)* é substituído ora pelo operador *interdiction_violated(daughters)*, ora pela dupla de operadores em seqüência *order_or_command(daughters)* e *order_carried_out(daughters)*, que também têm o efeito de tornar a vítima vulnerável. A segunda diferença refere-se à flexibilidade na ordem dos operadores *return(heroes)* e *liquidation(daughters,heroes)*. No plano gerado, não é obrigatório que o herói obtenha a vitória antes de retornar à sua terra, nem é obrigatório que a vítima já esteja livre para que o herói seja recompensado, uma vez que ele já tenha sido vitorioso. A flexibilidade na ordenação dos operadores surge porque a seqüência de satisfação das pré-condições não é restritiva o suficiente para gerar exatamente a mesma ordem do plano típico. Isso pode indicar tanto possibilidades de enredos novos mas aceitáveis, que são úteis ao usuário, quanto deficiências na definição das pré-condições e efeitos dos operadores, o que implica na necessidade de revisão da definição dos operadores.

Quando se tenta gerar exatamente o plano típico correspondente ao conto *expulsion_by_water*, obtém-se um resultado satisfatório. Em outra alternativa gerada para esse conto, ocorre a substituição do par de operadores *task(son)* e *solution(son)*, que tornam o filho vitorioso, pelo par de operadores *fight_in_an_open_field(son,parents)* e *victory_in_open_battle(son,parents)* que causam o mesmo efeito. A aplicação do gerador na tentativa de produzir exatamente o enredo correspondente ao conto *simple_murder* dá origem a variantes análogas às geradas para o conto *kidnapping*.

Quando um conto tem como pré-condição o objetivo de outro conto, pode ocorrer um encadeamento de contos. É possível imaginar um conto que tenha como pré-condição o fato de o herói ser rico (*hero_state(HERO,rich)*). Nesse caso, quando for reconhecido o conto em questão e se verificar que ele não é imediatamente aplicável no banco de dados **A**, o acionamento do planejador gerará, como resultado, um plano composto pelo operador correspondente ao plano *kidnapping*, que torna o herói rico, seguido do operador correspondente ao conto cujo enredo foi reconhecido. O esquema correspondente a esse plano alternativo é mostrado a seguir.

init → **kidnapping(daughters,heroes,...)** → **rich_hero_tale(...)** → **goal**

Outro tipo interessante de mistura de enredos típicos é o que dá origem a um enredo com seqüências que se desenvolvem em paralelo e se juntam em algum ponto. Essas alternativas surgem basicamente quando o banco de dados não corresponde ao do conto reconhecido e aciona-se o gerador de planos para criar uma nova alternativa com os mesmos objetivos desse conto. Isso ocorre, por exemplo, quando a observação de entrada é um operador *return(X)*, o banco de dados é o banco de dados **B** (adequado para o conto *expulsion_by_water*) e o plano reconhecido é *kidnapping(Y,son,parents,Z)*. Ao se solicitar a geração de um novo plano que gere os efeitos primários de *kidnapping*, ou seja, *hero_state(son,rich)* e *not(victim_state(Y,suffering))*, é fornecido o seguinte resultado.

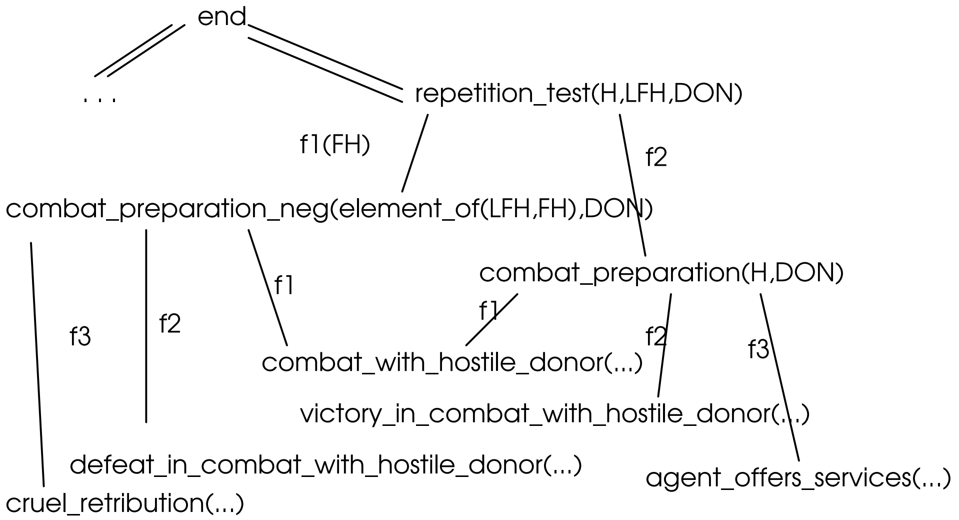


Pode-se verificar claramente no grafo acima que houve uma mistura dos enredos correspondentes aos contos *kidnapping* e *expulsion_by_water*, mas essa mistura corresponde a um enredo coerente, onde os pais vilões (*parents*) agem contra o filho (*son*) e contra uma vítima não identificada (variável “Y”). O filho, uma vez salvo, consegue resolver uma tarefa que o torna rico e liberta a vítima “Y”.

4.6. Enredos com Repetição de Operadores

Diversos contos catalogados por Propp contêm seqüências repetidas de funções aplicadas a personagens diferentes. O SRGP foi, então, adaptado para reconhecer planos onde houvesse um número indefinido de repetições de um certo operador, como é muito comum encontrar em contos de fadas. A possível repetição de um operador dentro de um plano passou a ser especificada da forma descrita no Apêndice II.

Para fins de teste do SRGP, já com a alteração relativa à repetição de operadores, foi criado um quarto enredo típico (*repetition_test(H,LFH,DON)*), criado a partir de uma parte de um enredo catalogado por Propp: "Três irmãos tentam obter um objeto mágico de um doador hostil. Os irmãos tentam a façanha por ordem de idade. Os dois mais velhos (*first_son* e *second_son*) são derrotados pelo doador, sendo caracterizados como falsos heróis, mas o mais novo (*third_son*), que é o verdadeiro herói, vence o desafio e se torna forte". O plano correspondente contém dois atributos com os seguintes suboperadores: *combat_preparation_neg* e *combat_preparation*. O primeiro suboperador é um operador de repetição que executa as tentativas fracassadas dos dois irmãos mais velhos enquanto que o segundo operador executa a tentativa bem sucedida do irmão mais novo. A variável *H* é ligada, nesse enredo, ao herói (irmão mais novo), *LFH* agrupa os falsos heróis (irmãos mais velhos) e *DON* é ligada ao doador. Na figura a seguir, é mostrada a parte da biblioteca de planos/operadores que contém os operadores relacionados a esse enredo típico. Pode-se observar que os dois suboperadores do plano também possuem, por sua vez, suboperadores.



O banco de dados que permite a aplicação imediata do plano *repetition_test*, quando reconhecido, contém os seguintes fatos: *character(hostile_donor,donor)*, *character(third_son,hero)*, *precedence(first_son,1)*, *precedence(second_son,2)* e *false_hero_list([first_son,second_son])*. Sendo que o predicado *precedence* estabelece uma precedência entre os dois primeiros filhos e o predicado *false_hero_list* tem o objetivo de definir explicitamente a lista de falsos heróis que devem tentar vencer o doador antes da tentativa do verdadeiro herói. As pré-condições do plano são as seguintes: *character(H,hero)*, *character(DON,donor)* e *false_hero_list(LFH)*. A lista de efeitos do plano contém unicamente o predicado *hero_state(H,strong)*.

Se, dentro do contexto descrito, o usuário informa como observações *cruel_retribution(first_son,hostile_donor)* e *combat_with_hostile_donor(second_son, hostile_donor)*, o reconhecimento fornece, como primeira hipótese de plano reconhecido, *repetition_test(second_son,[first_son],hostile_donor)*. Este plano, porém, possui como pré-condições não satisfeitas *character(second_son,hero)* e *false_hero_list([first_son])*. Uma vez que seja solicitada outra alternativa, é fornecido o plano *repetition_test(third_son,[first_son,second_son],hostile_donor)* que é automaticamente aplicável.

Se a lista de observações fornecidas contiver apenas o operador *cruel_retribution(first_son,hostile_donor)*, o plano reconhecido *repetition_test(third_son, [first_son], hostile_donor)* não é automaticamente aplicável porque indica a existência de apenas um falso herói (*first_son*). Ao ser solicitada a geração de um plano alternativo, é obtido o plano a seguir, o qual inclui a tentativa fracassada do segundo filho antes da tentativa do terceiro filho.

init

↓

combat_preparation_neg(first_son, donor)

↓

combat_preparation_neg(second_son,donor)

↓

combat_preparation(third_son,donor)

↓

goal

Deve-se ressaltar que a inclusão da tentativa do segundo filho só é possível porque está estabelecido nas pré-condições de *combat_preparation* que a lista de falsos heróis (*false_hero_list*) deve estar vazia antes de o operador ser aplicado. A lista de falsos heróis é reduzida a cada aplicação do operador *combat_preparation_neg*, porque, em suas pré-condições e efeitos, também está estabelecido que este operador tem que ser aplicado ao primeiro elemento da lista, antes dos demais, e que este primeiro elemento é eliminado após a aplicação do operador.

5. Conclusões e Trabalhos Futuros

O trabalho apresentado comprova a viabilidade de se tratar enredos como planos, obtendo-se, com isso, a possibilidade de, automaticamente, classificar enredos, predizer acontecimentos dentro de um certo contexto e gerar alternativas.

Duas frentes para a continuação deste trabalho parecem naturais. A verificação da aplicabilidade do SRGP e da metodologia proposta a outros gêneros literários é a primeira delas. A segunda é a criação de ferramentas que transformem textos, sob a forma de linguagem natural, em enredos, sob a forma de planos, e vice-versa. Torna-se necessário, também, estudar mais a fundo a possibilidade de introduzir conceitos tomados de outras teorias literárias. Além disso, os enredos não se limitam a gêneros literários. A metodologia aqui proposta pode ser adaptada a outros contextos tais como jogos, apoio a decisões gerenciais, acompanhamento de obras de engenharia, etc.

Conseguiu-se fazer uma aplicação real do protótipo implementado para o SRGP, utilizando largamente os recursos de reconhecimento e geração de planos e estudando as formas de interligação entre esses dois componentes. Tornou-se evidente o interesse em se aperfeiçoar o SRGP, para sua utilização tanto no contexto literário quanto em outros contextos. Dentre as características que seria desejável incorporar, pode ser destacada a necessidade de expandir o conceito de observações, para que possa incluir: disjunções de operadores e não apenas conjunções; restrições temporais; indicação explícita de objetivos a serem alcançados; e imposições de condições sobre estados intermediários. Torna-se também desejável fazer com que as pré-condições incluam, além de predicados simples, que exprimem fatos, outros predicados definidos por meio de regras.

Referências Bibliográficas

[Afa45] Afanas'ev, Aleksandr. "Russian Fairy Tales". Pantheon Books, New York, EUA. 1945.

[Alc95] Alcazar, J.J.P. “Respostas cooperativas baseadas na geração e reconhecimento de planos”. Tese de doutorado, Departamento de Informática, PUC/RJ, Brasil. 1995.

[Ari88] Arity Corporation. "Arity/Prolog Language Reference Manual". Concord, MA, EUA. 1988.

[CF96] Ciarlini,A.E.M.; Furtado, A.L. "Uso integrado de reconhecimento e geração de planos no contexto de bancos de dados". In Anais - XI Simpósio Brasileiro de Banco de Dados: 276-291, São Carlos, Brasil. 1996.

[Cha87] Chapman, D. "Planning for conjuctive goals". In Artificial Inteligence, 32:333-377. 1987.

[Eag94] Eagleton, Terry. "Teoria da Literatura: Uma Introdução", traduzido por W. Dutra. Martins Fontes, São Paulo, Brasil. 1994.

[FN71] Fikes, R. E.; Nilsson, N. J. "STRIPS: A new approach to the application of theorem proving to problem solving". In Artificial Intelligence 2(3-4): 189-208. 1971.

[Gre66] Greimas, A. J. "Sémantique Structurale". Librairie Larousse, Paris, França. 1966.

[Jol72] Jolles, André. "Formes Simples". Éditions du Seuil, Paris, França. 1972.

[Kau91] Kautz, H.A. “Formal theory of plan recognition”. In Reasoning about Plans - Allen, J.F.; Kautz, H.A.; Telavin, R.N.; Tenemberg, J.D. (eds.), Morgan Kaufmann, Los Altos, CA, EUA, Cap. 2. 1991.

[Lev58] Levi-Strauss, C. "Anthropologie Structurale". Plon, Paris, França. 1958.

[Lim83] Lima, Luiz Costa (editor). "Teoria da Literatura em suas Fontes". Francisco Alves, Rio de Janeiro, Brasil. 1983.

[LM87] Lambropoulos, Vassilis and Miller, David N. (eds.). "Twentieth-Century Literary Theory". State University of New York Press, Albany, EUA. 1987.

[Pro68] Propp, Vladimir. "Morphology of the Folktale", traduzido por Laurence Scott. University of Texas Press, Austin, EUA. 1968.

[Rum75] Rumelhart, David E. "Notes on a Schema for Stories". in Representation and Undertanding - Studies in Cognitive Science - D. G. Bobrow and A. Collins (eds.), Academic Press, New York, EUA. 1975.

[Sac74] Sacerdoti, E. "Planning in a hierarchy of abstraction spaces". In Artificial Intelligence, 5:115-135. 1974.

[YTW92] Yang, Q; Tenenberg, J.; Woods, S. "Abstraction in nonlinear planning". Technical Report n° CS-92-32, University of Waterloo, Canadá.1992.

[Web96] Webster, R. "Studying Literary Theory". Arnold, Londres, Inglaterra. 1996.

[Woo91] Woods, S. G. "An Implementation and evaluation of a hierarchical non-linear planner”. Master's thesis, Computer Science Department, University of Waterloo, Canadá. 1991.

Apêndice I. Estruturas de Dados dos Planos/Operadores

Para que o SRGP pudesse satisfazer tanto as necessidades do reconhecimento quanto da geração de planos, criou-se uma estrutura de dados para a definição dos planos/operadores que engloba todas as informações utilizadas nessas duas etapas.

Os operadores são definidos por uma lista Prolog com os seguintes elementos, na ordem:

- Identificador do operador.
- O termo de identificação do operador que é um functor PROLOG contendo o nome e os parâmetros do operador na forma *nome(PAR1,PAR2,PAR3,...,PARn).*
- Uma lista de pré-condições do operador. As pré-condições afirmativas são definidas sob a forma de functores, contendo o nome e os parâmetros (variáveis ou constantes). Caso a pré-condição seja uma negação, ela é envolvida pelo functor "not", da seguinte forma: *not(nome(PAR1,PAR2,...,PARn))*.
- Uma lista de efeitos gerados pelo operador, que possui a mesma estrutura da lista de pré-condições.
- O custo do operador.
- Uma lista de efeitos primários, gerados pelo operador, que possui a mesma estrutura da lista de pré-condições. Os efeitos primários são utilizados pelo gerador quando o uso deles está ativado. Nesse caso, a busca por um operador, que satisfaça uma certa pré-condição, é feita olhando-se, em cada operador, esta lista e não a lista de todos os efeitos gerados por ele. Quando um plano é reconhecido, atribui-se como sua finalidade, tanto na geração de novo plano quanto na adaptação do plano reconhecido, a geração dos efeitos contidos nesta lista.
- Uma lista de suboperadores componentes contendo, no caso de operadores complexos (planos), as duplas *(atributo, nome_do_componente(PAR1,PAR2,...,PARn))*.
- Uma lista contendo duplas de variáveis que não podem referenciar um mesmo objeto.
- Uma lista de restrições temporais, que estabelece a precedência entre os operadores componentes. Cada restrição é uma dupla contendo os atributos dos dois componentes.

A seguir é mostrado um exemplo de estrutura de operador.

```
operator([ seeker_hero_tale,
            seeker_hero_tale(VICTIM, HERO, VILLAIN)
        [
            character(VILLAIN,villain),
            character(HERO,hero),
            hero_state(HERO,strong),
            place(HERO,home),
            feels(DISPATCHER, love, VICTIM)        ],
        [
            hero_state(HERO, victorious),
            not(victim_state(VICTIM,suffering))
        ],
        10,
        [
            hero_state(HERO, victorious)
        ],
        [
            (f2, seeker_hero_quest(VICTIM,HERO,VILLAIN,DISPATCHER, ...)
            /* obs: demais componentes f1, f3 e f4 são herdados do plano tale */
        ]
        [(VICTIM,HERO), (VICTIM,VILLAIN), (HERO,VILLAIN)],
        [(f1,f2),(f2,f3),(f2,f4)] ).
```

Apêndice II. Especificação de Operadores de Repetição

Foram feitas as seguintes modificações nas definições de planos/operadores de forma a permitir a especificação de operadores que se repetem um número indefinido de vezes em um plano:

1. Quando um operador tem como componente um suboperador que se repete, o atributo correspondente ao suboperador tem, como identificação, ao invés de simplesmente um nome, um termo com uma variável a ser instanciada diferentemente em cada ocorrência do suboperador. Sendo assim, a especificação de uma dupla atributo/operador deste tipo é feita da seguinte forma: *(nome_do_atributo(VAR), nome_do_operador(...,VAR,...))*. As instanciações dessa variável diferenciam as ocorrências. No caso do enredo *repetition_test*, da seção 4.6, o atributo *f1(FH)* ocorrerá tantas vezes quanto o operador *combat_preparation_neg* aparecer com instanciações diferentes para a variável *FH*. Sendo assim, se forem fornecidas observações relativas às tentativas fracassadas do primeiro filho (*first_son*) e do segundo filho (*second_son*), aparecerão *f1(first_son)* e *f1(second_son)* .

2. Passou-se a admitir que argumentos dos operadores pudessem ser listas, que agrupam ocorrências de uma certa variável em um atributo com repetição. Se uma variável *L*, pertencente aos argumentos de um operador, agrupa as ocorrências de uma variável *VAR* em um suboperador, então a definição do suboperador deve ser feita da seguinte forma: *(nome_do_atributo(VAR), nome_do_operador(...,element_of(L,VAR),...))*. O termo *element_of(L,VAR)* aparece exatamente na posição correspondente aos valores de *VAR*. É por isso que o operador *combat_preparation_neg*, no enredo típico da seção 4.6, possui como primeiro argumento *element_of(LFH,FH)*. *FH* representa o falso herói de cada ocorrência enquanto que *LFH* é a lista que os agrupa. Se são fornecidas observações referentes a *first_son* e *second_son*, *LFH* assume o valor da lista *[first_son,second_son]*.

3. Se existe uma ordem obrigatória entre as ocorrências de um operador que se repete, essa ordem é especificada na lista de restrições temporais do plano através de um termo com a seguinte estrutura: *order(nome_do_atributo, VAR, VALUE, predicado_de_ordenação(..., VAR, ..., VALUE, ...))*. Neste termo, a variável *VAR* indica a variável que diferencia as ocorrências do atributo. A variável *VALUE* indica o valor de ordenação correspondente à variável *VAR*. O predicado de ordenação vincula *VAR* e *VALUE* e deve ter suas cláusulas especificadas no banco de dados, de forma a possibilitar a ordenação. No caso do enredo típico *repetition_test*, a ordem das duas ocorrências de *combat_preparation_neg* (atributo *f1*) é especificada, na lista de restrições temporais, pelo termo *order(f1,SON,PRECED, precedence(SON,PRECED))*. O banco de dados, por sua vez, contém os fatos relativos à precedência dos filhos: *precedence(first_son,1)* e *precedence(second_son,2)*.

4. Quando existe uma restrição temporal envolvendo um atributo repetido e um outro atributo qualquer, essa restrição é indicada normalmente na lista de restrições através da dupla *(nome_do_atributo_repetido, nome_do_outro_atributo)*, se qualquer ocorrência do atributo repetido precede a ocorrência do outro atributo, ou *(nome_do_outro_atributo, nome_do_atributo_repetido)*, quando a ordem é a inversa. No enredo típico *repetition_test*, a especificação de que a tentativa bem sucedida do filho mais novo (operador *combat_preparation*, atributo *f2*) deve ser precedida por todas as tentativas falhas dos filhos mais velhos (operador *combat_preparation_neg*, atributo *f2*) é feita com a inclusão da dupla *(f1,f2)* na lista de restrições temporais.