ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

ASTORYENGINE: LINGUAGGIO E INTERPRETE PER LIBRIGAME

Elaborata nel corso di: Linguaggi e Modelli Computazionali LM

Relazione di: ANDREA ARNOFFI

ANNO ACCADEMICO 2012–2013

Indice

Introduzione

Un librogame è un'opera narrativa che invece di essere letta linearmente dall'inizio alla fine presenta alcune possibili alternative mediante l'uso di paragrafi o pagine numerate. Lettori diversi (o la stessa persona in occasione di una rilettura) potranno compiere scelte diverse e ciò condizionerà lo svolgimento e la fine della trama. Questo genere, nato su supporto cartaceo, normalmente si presenta come un libro diviso in sezioni numerate, che terminano con una serie di opzioni che presentano le scelte che il lettore può prendere in quel punto; ad ogni opzione corrisponde il numero del paragrafo con cui si dovrà proseguire la lettura. In molti casi lo svolgimento della storia è determinato non solo dalle scelte compiute dal lettore, ma anche da opportuni punteggi che possono variare nel corso della storia. In base al valore di questi punteggi, alcune scelte possono essere inibite o selezionabili. Spesso, per conferire maggiore imprevedibilità alla storia, il valore di questi punteggi viene stabilito attraverso il lancio di un dado, o un altro meccanismo analogo che permetta di estrarre un valore casuale. Il lettore raggiunge infine un paragrafo conclusivo che porta a termine la storia.

Si può quindi pensare ad uno strumento che consenta di realizzare con facilità librigame, fruibili su computer. A tal fine sarà necessario progettare:

1. un **Linguaggio** per poter scrivere il librogame
2. un **Parser** che si assicuri che il librogame sia stato scritto correttamente secondo il linguaggio
3. un **Checker** che controlli che il librogame sia eseguibile senza problemi
4. un **Interprete** in grado di elaborare il racconto e permettere al lettore di interagirvi

Capitolo 1

Linguaggio

In questo capitolo ci si occuperà di progettare un linguaggio che permetta facilmente di realizzare librigame. In particolare verranno esaminati i costrutti necessari per realizzare un librogame, affinché si possa progettare un linguaggio che permetta di realizzare librigame senza particolari limitazioni. Infine verrà individuata una grammatica formale in grado di rappresentare in modo chiaro il linguaggio progettato.

1.1 Elementi fondamentali

Un librogame è fondamentalmente una storia composta da diverse sezioni, che possono essere astratte nel concetto di **pagina**. Inoltre ogni pagina contiene del **testo** che descrive l'evolversi della storia, e può essere collegata ad altre pagine attraverso un **collegamento**.
Questi sono gli elementi fondamentali per poter realizzare un librogame, che può quindi essere visto come un grafo in cui ad ogni nodo corrisponde una pagina. In più è necessario specificare la **pagina iniziale** (da cui iniziare il librogame), mentre le pagine terminali sono inevitabilmente quelle che non contengono collegamenti ad altre pagine.

La figura 1.1 esplica quanto appena detto. Si può notare come i collegamenti non siano soggetti ad alcun tipo di limitazione e ogni nodo possa puntare a qualsiasi altro nodo o a nessuno (anche a se stesso). Nell'immagine, *Page1* è il nodo di partenza da cui comincia la storia, mentre *Page8*, *Page9* e *Page10* corrispondono a nodi terminali, in cui la storia si conclude (quindi con possibili finali alternativi).

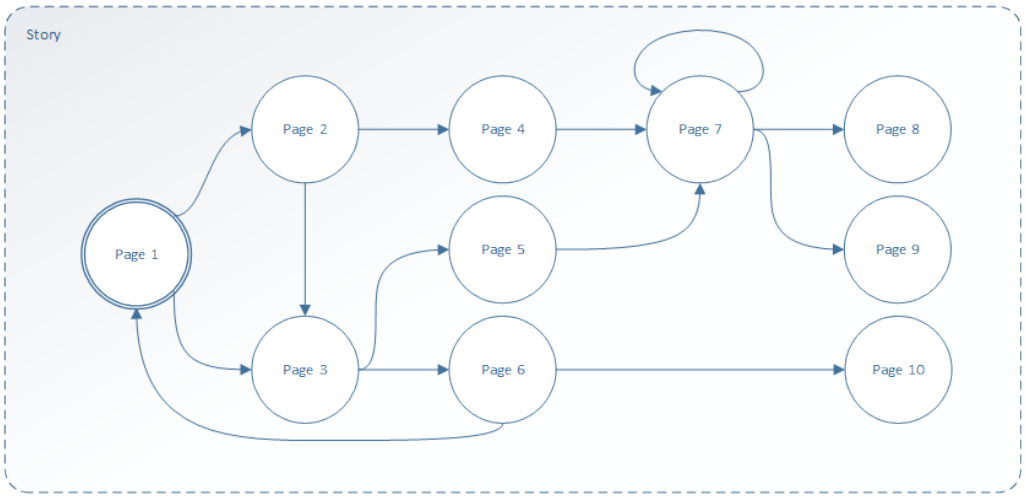

Figura 1.1: Struttura elementare di un librogame

A questo punto è possibile abbozzare una prima versione del linguaggio:

```
story Storia1 {
    start Pagina1;

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina";
    }

    page Pagina2 {
        text "Testo seconda pagina";
        link Fine "Continua";
    }

    page Fine {
        text "Ultima pagina";
    }
}
```

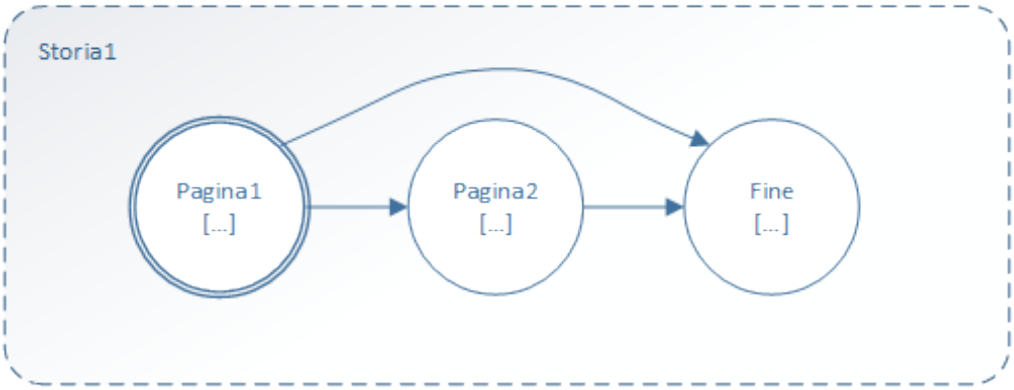

Figura 1.2: Modello con gli elementi fondamentali

Come si può notare, è presente un contenitore *story* che racchiude diversi elementi *page*, ognuno caratterizzato da un nome univoco, e l'istruzione *start*, che specifica la pagina iniziale della storia.
Ogni pagina è provvista di istruzioni *text* o *link*: la prima specifica il testo da visualizzare nella pagina, mentre la seconda specifica i collegamenti verso altre pagine (con il testo attraverso il quale il lettore potrà interagirvi).

Il modello dell'esempio è riportato in figura 1.2. Rispetto allo schema precedente, viene evidenziato come ogni pagina contenga del codice.

1.2 Estensione con variabili

Gli elementi elencati in precedenza sono la base per qualsiasi librogame, ma per poter realizzare racconti più complessi è necessario introdurre il concetto di **variabili**. Attraverso le variabili è possibile personalizzare lo svolgimento del librogame, ad esempio per implementare un sistema di punteggio che può indicare quantitativamente al lettore quanto le sue scelte siano state efficaci.
Ciò può essere realizzato mediante una variabile che viene incrementata ad ogni navigazione in una pagina a cui corrisponde una scelta corretta. Questa variabile potrà quindi essere visualizzata al lettore nella pagina finale, per informarlo del risultato conseguito.

È possibile arricchire l'esempio precedente con l'uso delle variabili:

```
story Storia1 {
    start Pagina1;

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina";
    }

    page Pagina2 {
        text "Testo seconda pagina";
        var $punteggio = $punteggio + 1;
        link Fine "Continua";
    }

    page Fine {
        text "Ultima pagina\n";
        text "Hai totalizzato $punteggio punti";
    }
}
```

In questo caso la scelta della seconda pagina provoca l'incremento di una *variabile* di nome punteggio. Questa variabile può poi essere visualizzata al lettore in un messaggio, sempre mediante l'istruzione *text*.
L'assegnazione di una variabile avviene attraverso l'istruzione *var*, che associa ad una variabile il valore di un'espressione numerica.

L'aggiunta delle variabili non modifica lo schema, per cui si può continuare a prendere come riferimento la figura 1.2.

In virtù di questo esempio si possono fare una serie di considerazioni.
In primo luogo le variabili non sono tipizzate, ma sono tutte di tipo numerico (intero). Inoltre le variabili non devono essere preventivamente dichiarate e tutte le nuove variabili hanno valore iniziale pari a 0. Infine si può osservare come le variabili abbiano un contesto globale a livello di storia e non di pagina: infatti una variabile definita in una pagina conserva il proprio valore anche se richiamata all'interno di un'altra pagina.
Come ultima considerazione, si può notare come in una pagina possano essere presenti più istruzioni di tipo *text*. In questo caso il testo da visualizzare viene accodato man mano che queste istruzioni vengono elaborate. Se il testo da visualizzare contiene delle variabili, queste dovranno per prima cosa essere sostituite dal loro valore.

1.2.1 Espressioni numeriche

L'introduzione delle variabili comporta inevitabilmente anche quello delle espressioni numeriche. Queste possono essere formate dalla composizione di variabili o numeri attraverso operazioni quali:

- +
- –
- *
- /
- %
- [Min,Max]

Le prime 5 sono operazioni binarie che corrispondono alla somma, sottrazione, moltiplicazione, divisione intera e modulo di due numeri (o variabili). L'ultima invece è una funzione che restituisce un valore casuale intero, compreso tra Min e Max.
L'uso delle parentesi tonde permette la scrittura di espressioni complesse.

1.3 Estensione dei collegamenti

Oltre alle espressioni matematiche contenute in una pagina, è possibile ipotizzare l'associazione di queste espressioni direttamente ai collegamenti. Questo può essere utile nel caso in cui la variazione di un valore dipenda dalla scelta effettuata e non direttamente dalla pagina di destinazione.

Un collegamento potrà contenere al suo interno espressioni numeriche (quindi istruzioni di tipo *var*) e queste espressioni numeriche verranno elaborate solo nel caso in cui il lettore scelga quel collegamento. Potrà contenere anche istruzioni di tipo *if-else* (descritte più avanti), ma non istruzioni di tipo *text* o di tipo *link*, che comportano la visualizzazione di testo o l'interazione con il lettore.

Un esempio può essere:

```
story Storia1 {
    start Pagina1;

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina"{
            var $punteggio = 5;
        };
    }

    page Pagina2 {
        text "Testo seconda pagina";
        var $punteggio = $punteggio + 1;
        link Fine "Continua";
    }

    page Fine {
        text "Ultima pagina\n";
        text "Hai totalizzato $punteggio punti";
    }
}
```

In questo modo, arrivare all'ultima pagina da Pagina1 permetterà di totalizzare 5 punti, mentre arrivarci da Pagina2 ne farà ottenere soltanto uno.

Il modello in questo caso cambia leggermente ed è riportato in figura 1.3. Si può notare la presenza di un collegamento, da *Pagina1* a *Fine*, che contiene del codice.

1.4 Estensione con il costrutto IF-ELSE

Per poter vincolare il contenuto di una pagina al valore corrente delle variabili è necessaria l'introduzione del costrutto IF-ELSE. Ciò consente allo scrittore di condizionare l'elaborazione della pagina in base al rispetto o meno di condizioni specificate.

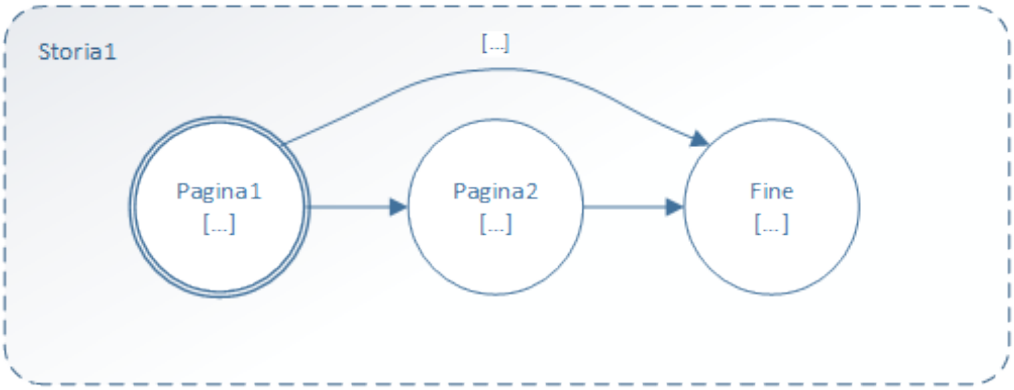

Figura 1.3: Modello con l'estensione dei collegamenti

Un esempio può essere:

```
story Storia1 {
    start Pagina1;

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina"{
            var $punteggio = 5;
        };
    }

    page Pagina2 {
        text "Testo seconda pagina";
        var $punteggio = $punteggio + 1;
        if ($punteggio > 0) {
            link Pagina2 "Ricarica la seconda pagina"{
                var $punteggio = -1;
            };
        }
        else {
            link Fine "Continua";
        }
    }

    page Fine {
        text "Ultima pagina\n";
```

```
27          if ($punteggio > 0) {
28              text "Hai totalizzato $punteggio punti";
29          }
30      }
31  }
```

In questo esempio, arrivare all'ultima pagina da Pagina1 permetterà di visualizzare il messaggio che avvisa della totalizzazione di 5 punti.
Nel caso si decida andare nella seconda pagina invece, verrà visualizzata solo la scelta Ricarica la seconda pagina; quindi, al nuovo ricaricamento, verrà visualizzata solo la scelta per andare all'ultima pagina. In questo caso, arrivarci da Pagina2 non permetterà di visualizzare il messaggio sui punti totalizzati, poiché saranno zero.

Lo schema in questo caso può essere descritto attraverso la figura 1.4. I collegamenti non sono più necessariamente stabili, ma alcuni possono essere aggiunti o rimossi a seconda dello script contenuto nelle pagine. Questa aleatorietà viene descritta attraverso le frecce tratteggiate, che indicano come la presenza di quei collegamenti possa dipendere dall'evolversi della storia o dalle scelte del lettore.

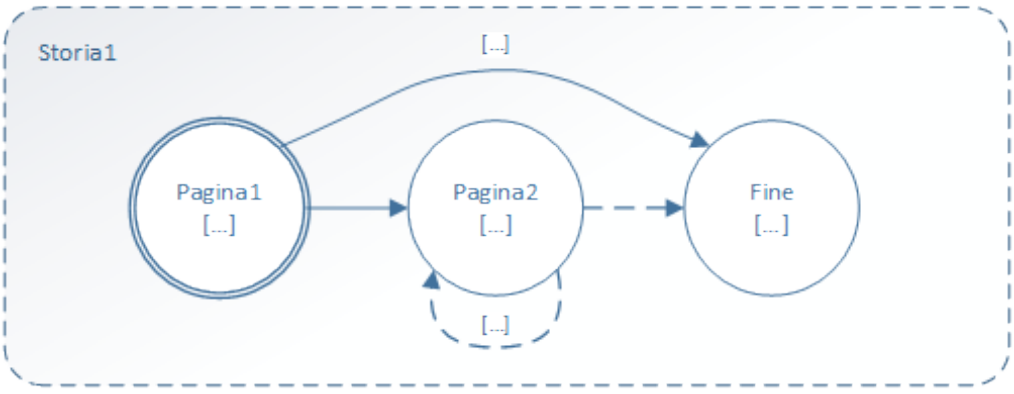

Figura 1.4: Modello con l'estensione con il costrutto If-Else

1.4.1 Espressioni logiche

Anche in questo caso è necessaria l'introduzione di espressioni, ma in questo caso si tratta di espressioni logiche, basate sulla logica booleana. Poiché le variabili hanno tutte un valore numerico, una variabile è considerata falsa se il suo valore è 0, altrimenti viene considerata vera.
Come per le espressioni numeriche, anche le espressioni logiche si ricavano dalla composizione di variabili o valori, mediante operatori logici quali:

- not
- and
- or
- ==
- !=
- <
- <=
- >
- >=

Il significato degli operatori dovrebbe essere chiaro. Sono tutti operatori binari, tranne il primo che è unario.
L'uso delle parentesi tonde permette la scrittura di espressioni complesse.

1.5 Altre estensioni

Infine è possibile estendere ulteriormente il linguaggio, per caratterizzare meglio il racconto. In particolare, oltre a *start* si può pensare di aggiungere un'istruzione *version* in previsione di evoluzioni del linguaggio, l'istruzione *title* per specificare il nome dell'avventura e l'istruzione *info* per aggiungere ulteriori informazioni al librogame.
In più, per agevolare lo scrittore, è possibile aggiungere il supporto ai commenti.

L'esempio completo perciò diventa:

```
/*
    Commento normale
*/
story Storia1 {     // Commento di linea
    start Pagina1;
    version "1.0";
    title "LibroGame N°1";
    info "LibroGame scritto da NomeAutore";

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina"{
            var $punteggio = 5;
        };
    }

    page Pagina2 {
        text "Testo seconda pagina";
        var $punteggio = $punteggio + 1;
        if ($punteggio > 0) {
            link Pagina2 "Ricarica la seconda pagina"{
                var $punteggio = -1;
            };
        }
        else {
            link Fine "Continua";
        }
    }

    page Fine {
        text "Ultima pagina\n";
        if ($punteggio > 0) {
            text "Hai totalizzato $punteggio punti";
        }
    }
}
```

Non sono presenti variazioni significative, perciò lo schema di riferimento rimane quello della figura 1.4.

1.6 Definizione di una grammatica formale

Ora che sono stati definiti tutti gli elementi, è possibile definire una grammatica formale BNF per descrivere il linguaggio.

Dato `eps` come il simbolo di produzione vuota e date le produzioni fondamentali, ossia quelle relative ai nomi (`Name`), numeri (`Number`) e stringhe di testo (`Text`), la grammatica con scopo `S` è:

```
Var ::= $Name
NumValue ::= Number | Var

S ::= story Name { ElementStory StoryBodyRest }
StoryBodyRest    ::= eps | ElementStory StoryBodyRest
ElementStory     ::= InstructionStory | Page

InstructionStory ::=
    start Name ; |
    version "Text" ; |
    title "Text" ; |
    info "Text" ;

Page ::= page Name { ElementPage PageBodyRest }
PageBodyRest ::= eps | ElementPage PageBodyRest
ElementPage ::=
    text "Text" ; |
    var NumericExpr ; |
    link Name "Text" Link ; |
    if LogicExpr { ElementPage PageBodyRest } ElsePage
ElsePage ::= eps | else { ElementPage PageBodyRest }

Link ::= eps | { ElementLink LinkBodyRest }
LinkBodyRest ::= eps | ElementLink LinkBodyRest
ElementLink ::=
    var NumericExpr ; |
    if LogicExpr { ElementLink LinkBodyRest } ElseLink
ElseLink ::= eps | else { ElementLink LinkBodyRest }
```

Quindi occorre definire le produzioni per le espressioni numeriche:

```
NumericExpr ::= Var = NumExprBody
NumExprBody ::= NumMoltDivElem SumSubOp
SumSubOp    ::= SumRest | SubRest
SumRest     ::= eps | + NumExprBody
SubRest     ::= eps | - NumExprBody

NumMoltDivElem ::= UnaryOp MulDivModOp
MulDivModOp    ::= MulRest | DivRest | ModRest
MulRest ::= eps | * NumExprBody
DivRest ::= eps | / NumExprBody
ModRest ::= eps | % NumExprBody

UnaryOp  ::= NumValue | ( NumExprBody ) | [NumValue,NumValue]
```

Come si può notare vengono rispettate le normali convenzioni di priorità degli operatori aritmetici, e la funzione di generazione di numeri casuali viene trattata alla stregua di un numero.

Infine occorre definire le produzioni per le espressioni logiche:

```
LogicExpr ::= LogicValue AndOrOp
AndOrOp   ::= eps | and LogicExpr | or LogicExpr
LogicValue ::=
        not LogicValue       |
        NumValue CompOp      |
        ( LogicExpr ) CompOp
CompOp ::= eps | == LogicExpr | != LogicExpr | <  LogicExpr
               | <= LogicExpr | >  LogicExpr | >= LogicExpr
```

Si può evincere come le operazioni di confronto siano le operazioni più prioritarie, seguite dal `not` e quindi da `and` e `or`.

Come si può notare, la grammatica formale individuata non prende in considerazione i commenti. La loro gestione verrà delegata a un opportuno *tokenizer*, che provvederà a restituire solo gli elementi significativi del linguaggio per il *parser*, come si potrà vedere nel capitolo successivo.

Capitolo 2

Parsing

In questo capitolo ci si occuperà di tutte le operazioni necessarie per il parsing della grammatica definita nel capitolo precedente. Per la loro realizzazione verranno sfruttate le potenzialità offerte da *tuProlog* e dalla libreria *DCGLibrary*.

Per prima cosa verrà realizzato un tokenizer che, dato un file di testo con il codice del librogame, restituirà una lista di token da fornire al parser. Quindi verrà realizzato il parser vero e proprio, che controllerà la correttezza del codice scritto e permetterà di ottenere l'abstract syntax tree in grado di rappresentare il librogame.

2.1 Tokenizer

Il primo passo per riconoscere la grammatica è dato da un tokenizer in grado di convertire un file di testo, col codice del librogame, in una lista di token che sia comoda per rappresentare il contenuto del file testuale.
Il compito di un tokenizer si può dividere in due parti: ricerca delle parti di testo da scartare (come spazi, byte di andata a capo, commenti...) e ottenimento dei token relativi alle altre parti del file. Perciò è possibile cominciare a delineare il tokenizer per soffermarsi in seguito a quali parti scartare e quali invece memorizzare in forma di token.

Il codice prolog del nostro tokenizer quindi comincerà con:

```
:-load_library('alice.tuprolog.lib.DCGLibrary').
% riceve i simboli e restituisce i token
tokenizer(L,Out):-
    atom_codes(L,Codes),
    tokenizer(t,skip,Codes,[],Out).
% cerca caratteri da scartare
tokenizer(Symb,Skip,LI,LO,Out):-
    phrase(Skip,LI,LO1),!,
    tokenizer(Symb,Skip,LO1,LO,Out).
% ottiene un token
tokenizer(Symb,Skip,LI,LO,[A|Out]):-
    P =.. [Symb,A],
    phrase(P,LI,LO1),!,
    tokenizer(Symb,Skip,LO1,LO,Out).
% termina la tokenizzazione
tokenizer(_,_,L,L,[]).
```

2.1.1 Simboli da scartare

A questo punto occorre indicare i codici ASCII dei caratteri da scartare. In questo caso, oltre a spazi, tabulazioni e comandi di andata a capo è necessario scartare tutti i commenti dal codice del programma, che ai fini del linguaggio non forniscono contenuto informativo per l'interprete. Il codice del tokenizzatore perciò proseguirà con:

```
skip --> [32]. % Spazio ' '
skip --> [13]. % CR     \r
skip --> [10]. % NL     \n
skip --> [9].  % TAB    \t
skip --> [12]. % FF     \f formfeed
% Linea di commento:    // ...
skip --> {atom_codes('//',X)},X,X,lineComment.
lineComment --> [10].
lineComment --> [X],lineComment.
% Blocco di commento:   /* ... */
skip --> {atom_codes('/*',X)},X,blockComment.
blockComment --> [X],{atom_codes('*',[X])},blockComment2.
blockComment --> [X],blockComment.
blockComment2 --> [X],{atom_codes('/',[X])}.
blockComment2 --> [X],blockComment.
```

Per rendere più efficiente il tokenizzatore, conviene ricavare preventivamente i codici dei caratteri da scartare, per evitare che vengano ricalcolati ad ogni esecuzione, attraverso il predicato `atom_codes`. In questo modo il codice precedente diverrà:

```
17 skip --> [32]. % Spazio ' '
18 skip --> [13]. % CR      \r
19 skip --> [10]. % NL      \n
20 skip --> [9].  % TAB     \t
21 skip --> [12]. % FF      \f formfeed
22 % Linea di commento:     // ...
23 skip --> [47],[47],lineComment.
24 lineComment --> [10].
25 lineComment --> [X],lineComment.
26 % Blocco di commento:    /* ... */
27 skip --> [47,42],blockComment.
28 blockComment --> [42],blockComment2.
29 blockComment --> [X],blockComment.
30 blockComment2 --> [47].
31 blockComment2 --> [X],blockComment.
```

2.1.2 Token da ottenere

Come ultima cosa, bisogna specificare al tokenizer quali sono i token da ricavare dal codice scritto del librogame. In questo caso si può anche distinguere tra le *parole riservate* del linguaggio da riconoscere e il resto del linguaggio, composto ad esempio da *nomi*, *valori* e *stringhe*.
Le parole riservate sono:

- else
- if
- info
- link
- page
- start
- story
- text
- title
- var
- version

```
32 % Parole riservate all'interno del codice sorgente
33 keywords(['else','if','info','link','page','start','story','text
       ','title','var','version']).
34 t(TOK) --> {keywords(Ts),member(TOK,Ts),atom_codes(TOK,X)},X.
35
36 % Espressioni logiche
37 t(eq)  --> {atom_codes('==',X)},X.  % ==
38 t(neq) --> {atom_codes('!=',X)},X.  % !=
39 t(gte) --> {atom_codes('>=',X)},X.  % >=
40 t(gt)  --> {atom_codes('>',X)},X.   % >
41 t(lte) --> {atom_codes('<=',X)},X.  % <=
42 t(lt)  --> {atom_codes('<',X)},X.   % <
43 t(not) --> {atom_codes('not',X)},X. % not
44 t(and) --> {atom_codes('and',X)},X. % and
45 t(or)  --> {atom_codes('or',X)},X.  % or
46
47 % Espressioni aritmetiche
48 t(sqlbra) --> {atom_codes('[',X)},X. % [
49 t(sqrbra) --> {atom_codes(']',X)},X. % ]
50 t(comma)  --> {atom_codes(',',X)},X. % ,
51 t(dot)    --> {atom_codes('.',X)},X. % .
52 t(plus)   --> {atom_codes('+',X)},X. % +
53 t(minus)  --> {atom_codes('-',X)},X. % -
54 t(times)  --> {atom_codes('*',X)},X. % *
55 t(div)    --> {atom_codes('/',X)},X. % /
56 t(mod)    --> {atom_codes('%',X)},X. % %
57 t(is)     --> {atom_codes('=',X)},X. % =
58
59 % Parole composte da lettere e numeri
60 t(str(S))    --> string(S).     % Stringa di testo
61 t(var(VAR))  --> {atom_codes('$',X)},X,literal(VAR). % Variabile
62 t(num(NUM))  --> numb(NUM).     % Numero
63 t(name(LIT)) --> literal(LIT).  % Nome identificativo
64
65 % Altri simboli e punteggiatura
66 t(colon) --> {atom_codes(';',X)},X. % ;
67 t(lbra)  --> {atom_codes('(',X)},X. % (
68 t(rbra)  --> {atom_codes(')',X)},X. % )
69 t(begin) --> {atom_codes('{',X)},X. % {
70 t(end)   --> {atom_codes('}',X)},X. % }
```

Come si può notare, ogni variabile è una parola che inizia con il carattere $ ed è presente un token specifico per ogni parola riservata del linguaggio. Entrando più nel dettaglio, si può esaminare la definizione delle parole composte da lettere e numeri.

```
% Numero
numb(N)    --> digit(D),numbz(L),{atom_codes(A,[D|L]),
               num_atom(N,A)}.
digit(N)   --> [N],{N>47,N < 58}. % [0..9]
numbz([D|L]) --> digit(D),numbz(L). % Sequenza di cifre
numbz([]) --> [].

% Letterale
char(N) --> [N],{N>96, N<123}. % [a..z]
char(N) --> [N],{N>64, N<91}. % [A..Z]
literal(A)  --> char(N),lit(L),
                {atom_codes(A,[N|L])}. % Sequenza di lettere
lit([N|L2]) --> char(N),lit(L2).
% Un letterale può contenere anche numeri
lit([N|L2]) --> digit(N),lit(L2).
lit([]) --> [].

% Stringa
string(T) --> {atom_codes('"',X)},X,strRest(Y),
              {atom_codes(T,Y)}.
% Carattere di escape per valutare correttamente una stringa
strRest([Y,X|T]) --> {atom_codes('\',[Y]),atom_codes('"',[X])},
                     [Y],[X],strRest(T).
strRest([Y,X|T]) --> {atom_codes('\',[Y]),atom_codes('§',[X])},
                     atom_codes("'",[Z])},[Z],strRest(T).
strRest([])      --> {atom_codes('"',X)},X.
strRest([X|T])   --> [X],strRest(T).
```

Si può notare come un numero sia una sequenza di cifre, un letterale una sequenza di lettere o cifre (che cominciano però con una lettera) e una stringa invece sia una sequenza di caratteri delimitata da una coppia di virgolette: ¨.
Il carattere ¨ può essere inserito all'interno di una stringa mediante il carattere di escape backslash: \. Il carattere ' all'interno delle stringhe invece viene automaticamente convertito in \§.
Il tokenizzatore provvederà quindi a interpretare correttamente la stringa specificata.

Analogamente a prima, per rendere più efficiente il tokenizzatore è possibile ricavare preventivamente i codici dei caratteri da esaminare. In questo modo il codice precedente diverrà:

```
% Parole riservate all'interno del codice sorgente
t(else)    --> [101,108,115,101].
t(if)      --> [105,102].
t(info)    --> [105,110,102,111].
t(link)    --> [108,105,110,107].
t(page)    --> [112,97,103,101].
t(start)   --> [115,116,97,114,116].
t(story)   --> [115,116,111,114,121].
t(text)    --> [116,101,120,116].
t(title)   --> [116,105,116,108,101].
t(var)     --> [118,97,114].
t(version) --> [118,101,114,115,105,111,110].

% Espressioni logiche
t(eq)  --> [61,61].       % ==
t(neq) --> [33,61].       % !=
t(gte) --> [62,61].       % >=
t(gt)  --> [62].          % >
t(lte) --> [60,61].       % <=
t(lt)  --> [60].          % <
t(not) --> [110,111,116]. % not
t(and) --> [97,110,100].  % and
t(or)  --> [111,114].     % or

% Espressioni aritmetiche
t(sqlbra) --> [91]. % [
t(sqrbra) --> [93]. % ]
t(comma)  --> [44]. % ,
t(dot)    --> [46]. % .
t(plus)   --> [43]. % +
t(minus)  --> [45]. % -
t(times)  --> [42]. % *
t(div)    --> [47]. % /
t(mod)    --> [37]. % %
t(is)     --> [61]. % =

% Parole composte da lettere e numeri
t(str(S))    --> string(S).     % Stringa di testo
t(var(VAR))  --> [36],literal(VAR). % Variabile
t(num(NUM))  --> numb(NUM).     % Numero
t(name(LIT)) --> literal(LIT).  % Nome identificativo
```

```
73
74 % Altri simboli e punteggiatura
75 t(colon) --> [59].  % ;
76 t(lbra)  --> [40].  % (
77 t(rbra)  --> [41].  % )
78 t(begin) --> [123]. % {
79 t(end)   --> [125]. % }
80
81 % Numero
82 numb(N)    --> digit(D),numbz(L),{atom_codes(A,[D|L]),
83                 num_atom(N,A)}.
84 digit(N)  --> [N],{N>47,N < 58}. % [0..9]
85 numbz([D|L]) --> digit(D),numbz(L). % Sequenza di cifre
86 numbz([]) --> [].
87
88 % Letterale
89 char(N) --> [N],{N>96, N<123}. % [a..z]
90 char(N) --> [N],{N>64, N<91}. % [A..Z]
91 literal(A)  --> char(N),lit(L),
92                 {atom_codes(A,[N|L])}. % Sequenza di lettere
93 lit([N|L2]) --> char(N),lit(L2).
94 % Un letterale può contenere anche numeri
95 lit([N|L2]) --> digit(N),lit(L2).
96 lit([]) --> [].
97
98 % Stringa
99 string(T) --> [34],strRest(Y),{atom_codes(T,Y)}.
100 % Carattere di escape per valutare correttamente una stringa
101 strRest([92,34|T])  --> [92],[34],strRest(T).
102 strRest([92,167|T]) --> [39],strRest(T).
103 strRest([])         --> [34].
104 strRest([X|T])      --> [X],strRest(T).
```

Il riconoscimento di numeri negativi e/o decimali viene delegato al parser, che ad esempio si occuperà perciò di discriminare l'uso del simbolo - per indicare un numero negativo e l'uso dello stesso simbolo per esprimere l'operazione di sottrazione.

2.1.3 Applicazione del tokenizer

Per testare le capacità del tokenizer, è possibile ricorrere all'esempio precedente. Al fine di rendere l'esempio più completo, nella pagina *Fine* è stata aggiunta un'istruzione per poter verificare la corretta tokenizzazione della funzione di generazione di numeri casuali `[Min,Max]`.

```
/*
    Commento normale
*/
story Storia1 {     // Commento di linea
    start Pagina1;
    version "1.0";
    title "LibroGame N°1";
    info "LibroGame scritto da NomeAutore";

    page Pagina1 {
        text "Testo prima pagina";
        link Pagina2 "Vai alla seconda pagina";
        link Fine "Vai all'ultima pagina"{
            var $punteggio = 5;
        };
    }

    page Pagina2 {
        text "Testo seconda pagina";
        var $punteggio = $punteggio + 1;
        if ($punteggio > 0) {
            link Pagina2 "Ricarica la seconda pagina"{
                var $punteggio = -1;
            };
        }
        else {
            link Fine "Continua";
        }
    }

    page Fine {
        text "Ultima pagina\n";
        if ($punteggio > 0) {
            text "Hai totalizzato $punteggio punti";
            var $punteggio = [0,5];
        }
    }
}
```

L'esecuzione del tokenizer su questo esempio produrrà come risultato la seguente lista di token (indentata analogamente al codice sorgente):

```
story,name('Storia1'),begin,
    start,name('Pagina1'),colon,
    version,str('1.0'),colon,
    title,str('LibroGame N°1'),colon,
    info,str('LibroGame scritto da NomeAutore'),colon,

    page,name('Pagina1'),begin,
        text,str('Testo prima pagina'),colon,
        link,name('Pagina2'),str('Vai alla seconda pagina'),colon,
        link,name('Fine'),str('Vai all'ultima pagina'),begin,
            var,var(punteggio),is,num(5),colon,
        end,colon,
    end,

    page,name('Pagina2'),begin,
        text,str('Testo seconda pagina'),colon,
        var,var(punteggio),is,var(punteggio),plus,num(1),colon,
        if,lbra,var(punteggio),gt,num(0),rbra,begin,
            link,name('Pagina2'),str('Ricarica la seconda pagina'),begin,
                var,var(punteggio),is,minus,num(1),colon,
            end,colon,
        end,
        else,begin,
            link,name('Fine'),str('Continua'),colon,
        end,
    end,

    page,name('Fine'),begin,
        text,str('Ultima pagina\n'),colon,
        if,lbra,var(punteggio),gt,num(0),rbra,begin,
            text,str('Hai totalizzato \$punteggio punti'),colon,
            var,var(punteggio),is,sqlbra,num(0),comma,num(5),sqrbra,colon,
        end,
    end,
end
```

2.2 Parser

Una volta realizzato il tokenizer, si può procedere alla progettazione del parser, che si occuperà di verificare il rispetto della grammatica formale e provvederà a realizzare l'AST relativo al codice sorgente esaminato.
L'implementazione del parser segue molto da vicino la definizione della grammatica formale:

```
1  parsegen(T,A):-phrase(s(A),T),!.
2
3  % Storia
4  s(story(N,[H|T])) --> [story],name(N),[begin],
5                        elementStory(H),storyBodyRest(T),[end].
6  storyBodyRest([])     --> [].
7  storyBodyRest([H|T]) --> elementStory(H),storyBodyRest(T).
8  elementStory(X)    --> instructionStory(X).
9  elementStory(X)    --> page(X).
10
11 % Istruzioni della storia
12 instructionStory(start(X))    --> [start],name(X),[colon].
13 instructionStory(version(X)) --> [version],text(X),[colon].
14 instructionStory(title(X))    --> [title],text(X),[colon].
15 instructionStory(info(X))     --> [info],text(X),[colon].
16
17 % Pagine e istruzioni delle pagine
18 page(page(N,[H|T]))    --> [page],name(N),[begin],
19                            elementPage(H),pageBodyRest(T),[end].
20 pageBodyRest([])        --> [].
21 pageBodyRest([H|T])     --> elementPage(H),pageBodyRest(T).
22 elementPage(text(T))    --> [text],text(T),[colon].
23 elementPage(var(V,E))   --> [var],numericExpr(V,E),[colon].
24 elementPage(link(N,T,L))    --> [link],name(N),text(T),
25                                 link(L),[colon].
26 elementPage(if(C,[H|T],E)) --> [if],logicExpr(C),[begin],
27                                 elementPage(H),pageBodyRest(T),
28                                 [end],elsePage(E).
29 elsePage([])      --> [].
30 elsePage([H|T]) --> [else,begin],elementPage(H),pageBodyRest(T),
31                     [end].
32 link([])      --> [].
33 link([H|T]) --> [begin],elementLink(H),linkBodyRest(T),[end].
34 linkBodyRest([])      --> [].
35 linkBodyRest([H|T]) --> elementLink(H),linkBodyRest(T).
36
```

```
37 elementLink(var(V,E))        --> [var],numericExpr(V,E),[colon].
38 elementLink(if(C,[H|T],E)) --> [if],logicExpr(C),[begin],
39                                 elementLink(H),linkBodyRest(T),
40                                 [end],elseLink(E).
41 elseLink([])      --> [].
42 elseLink([H|T]) --> [else,begin],elementLink(H),linkBodyRest(T),
43                     [end].
44
45 % Espressioni numeriche
46 numericExpr(V,E) --> [var(V),is],numExprBody(E).
47 numExprBody(E)    --> numMoltDivElem(H),sumSubOp(H,E).
48 sumSubOp(H,E) --> sumRest(H,E).
49 sumSubOp(H,E) --> subRest(H,E).
50 sumRest(X,X)   --> [].
51 sumRest(A,B)   --> [plus],numExprBody(C),sumRest(add(A,C),B).
52 subRest(X,X)   --> [].
53 subRest(A,B)   --> [minus],numExprBody(C),subRest(sub(A,C),B).
54 numMoltDivElem(E) --> unaryOp(H),mulDivModOp(H,E).
55 mulDivModOp(H,E)  --> mulRest(H,E).
56 mulDivModOp(H,E)  --> divRest(H,E).
57 mulDivModOp(H,E)  --> modRest(H,E).
58 unaryOp(N) --> numValue(N).
59 unaryOp(E) --> [lbra],numExprBody(E),[rbra].
60 unaryOp(rand(A,B)) --> [sqlbra],numValue(A),[comma],numValue(B),
61                        [sqrbra].
62 mulRest(X,X) --> [].
63 mulRest(A,B) --> [times],numExprBody(C),mulRest(mul(A,C),B).
64 divRest(X,X) --> [].
65 divRest(A,B) --> [div],numExprBody(C),divRest(div(A,C),B).
66 modRest(X,X) --> [].
67 modRest(A,B) --> [mod],numExprBody(C),modRest(mod(A,C),B).
68
69 % Espressioni logiche
70 logicExpr(E)   --> logicValue(A),andOrOp(A,E).
71 andOrOp(X,X)   --> [].
72 andOrOp(A,B)   --> [and],logicExpr(C),andOrOp(and(A,C),B).
73 andOrOp(A,B)   --> [or],logicExpr(C),andOrOp(or(A,C),B).
74 logicValue(not(E)) --> [not],logicValue(E).
75 logicValue(L) --> numValue(N),compOp(N,L).
76 logicValue(L) --> [lbra],logicExpr(E),[rbra],compOp(E,L).
77 compOp(X,X)    --> [].
78 compOp(A,L)    --> [eq],logicExpr(B),compOp(eq(A,B),L).
79 compOp(A,L)    --> [neq],logicExpr(B),compOp(neq(A,B),L).
80 compOp(A,L)    --> [gte],logicExpr(B),compOp(gte(A,B),L).
81 compOp(A,L)    --> [gt],logicExpr(B),compOp(gt(A,B),L).
```

```
82 compOp(A,L)     --> [lte], logicExpr(B),compOp(lte(A,B),L).
83 compOp(A,L)     --> [lt], logicExpr(B),compOp(lt(A,B),L).
84
85 % Altri elementi del linguaggio
86 name(N) --> [name(N)].
87 text(T) --> [str(T)].
88 numValue(num(R)) --> [minus,num(N),dot,num(M)],% Reale negativo
89                      {atom_chars(X,['-',N,'.',M]),
90                      num_atom(R,X)}.
91 numValue(num(R)) --> [plus,num(N),dot,num(M)], % Reale positivo
92                      {atom_chars(X,[N,'.',M]),num_atom(R,X)}.
93 numValue(num(N)) --> [plus,num(N)].              % Intero positivo
94 numValue(num(R)) --> [minus,num(N)],             % Intero negativo
95                      {atom_chars(X,['-',N]),num_atom(R,X)}.
96 numValue(num(R)) --> [num(N),dot,num(M)],        % Reale positivo
97                      {atom_chars(X,[N,'.',M]),num_atom(R,X)}.
98 numValue(num(N)) --> [num(N)].                   % Intero positivo
99 numValue(var(V)) --> [var(V)].
```

Si può notare come i numeri decimali e i numeri negativi vengano opportunamente costruiti unendo i diversi simboli memorizzati dal tokenizer.

2.2.1 Applicazione del parser

A questo punto si può verificare il risultato del parser, ad esempio sfruttando la lista di token ricavata dall'esempio precedente. L'esecuzione del parser su questo esempio produrrà come risultato il seguente AST (indentato analogamente al codice sorgente):

```
story('Storia1',[
    start('Pagina1'),
    version('1.0'),
    title('LibroGame N°1'),
    info('LibroGame scritto da NomeAutore'),
    page('Pagina1',[
        text('Testo prima pagina'),
        link('Pagina2','Vai alla seconda pagina',[]),
        link('Fine','Vai all'ultima pagina',[
            var(punteggio,num(5))
        ])
    ]),
    page('Pagina2',[
        text('Testo seconda pagina'),
        var(punteggio,add(var(punteggio),num(1))),
        if(gt(var(punteggio),num(0)),[
            link('Pagina2','Ricarica la seconda pagina',[
                var(punteggio,num(-1))
            ])
        ],[
            link('Fine','Continua',[])
        ])
    ]),
    page('Fine',[
        text('Ultima pagina\n'),
        if(gt(var(punteggio),num(0)),[
            text('Hai totalizzato $punteggio punti'),
            var(punteggio,rand(num(0),num(5)))
        ],[])
    ])
])
```

Ultimato l'AST si potrebbe già procedere con l'implementazione dell'interprete per librigame, ma prima è necessario effettuare alcuni controlli, per verificare anche la correttezza semantica del programma scritto. Questa fase di controllo verrà affrontata nel capitolo successivo.

Capitolo 3

Checking

In questo capitolo ci si occuperà dei controlli da effettuare sull'AST restituito dal parsing, per verificare che il librogame scritto sia non solo sintatticamente, ma anche semanticamente corretto.

Il superamento di tutti i test permetterà di concludere con un ragionevole grado di sicurezza che il librogame scritto può essere interpretato correttamente. Tuttavia, non è possibile averne la certezza, poichè alcuni controlli possono essere effettuati solamente durante l'esecuzione (come ad esempio i test che dipendono dal valore attuale delle variabili).
Verranno perciò descritti i vari test e per ognuno verrà fornita la rispettiva implementazione.

3.1 Presenza e unicità di elementi

Ogni librogame scritto, contiene degli elementi fondamentali per la sua corretta esecuzione. In questa sezione verranno esaminati i test dedicati a controllarne la presenza.
Inoltre ci sono alcune istruzioni che non possono comparire più di una volta per ogni librogame, per cui ci si occuperà anche di verificare questa condizione.

3.1.1 Presenza dell'istruzione `start`

Questo test verifica la presenza dell'istruzione `start`, indispensabile per specificare all'interprete la pagina iniziale.

```
1 checkStart(A):-member(start(_),A),!.
```

3.1.2 Unicità di elementi specifici

Questo test verifica che le istruzioni `start`, `version`, `title` e `info` siano presenti al massimo una volta all'interno dell'AST. Ciò viene fatto perché simili istruzioni hanno un significato globale, a livello di librogame, e perciò se comparissero più di una volta potrebbero portare ad ambiguità.

```
checkISTV(A):-
    findall(info(I),member(info(I),A),LI),
        length(LI,NLI),NLI < 2,
    findall(start(S),member(start(S),A),LS),
        length(LS,NLS),NLS < 2,
    findall(title(T),member(title(T),A),LT),
        length(LT,NLT),NLT < 2,
    findall(version(V),member(version(V),A),LV),
        length(LV,NLV),NLV < 2.
```

Inoltre bisogna assicurarsi che non vi siano più pagine con lo stesso nome definite all'interno del medesimo librogame.

```
checkDupPages(A):-
    findall(page(X),member(page(X,_),A),Pages),length(Pages,NP),
    setof(page(X),member(page(X),Pages),SPag),!,length(SPag,NP).
```

3.2 Correttezza dei collegamenti

Per la corretta interpretazione di un librogame, è necessario che i collegamenti tra le pagine siano corretti, e quindi che le pagine puntate dalle istruzioni `start` e `link` effettivamente siano state definite.

```
checkLinks(A):-
    member(start(S),A),!,member(page(S,_),A),!,
    findall(I,member(page(_,I),A),Pages),
    concatLinks(Pages,Links),checkLink(A,Links).

concatLinks(Pages,Out):-concatLinks(Pages,Out,[]).
concatLinks([],Out,O):-concatLists(O,Out).
concatLinks([H|T],Out,R):-getLinks(H,L),(L=[],!,
    concatLinks(T,Out,R);concatLinks(T,Out,[L|R])).
```

```
23
24 getLinks([],[]).
25 getLinks([if(Cond,Body,Else)|T1],T2):-!,
26     append(Body,Else,IF),append(IF,T1,T),getLinks(T,T2).
27 getLinks([link(Page,Name,Body)|T1],[link(Page,Name,Body)|T2]):-
28     !,getLinks(T1,T2).
29 getLinks([H|T1],T2):-getLinks(T1,T2).
30
31 concatLists([],[]).
32 concatLists([L|T],T2):-list(L),!,append(L,T,TN),
33     concatLists(TN,T2).
34 concatLists([H|T],[H|T2]):-concatLists(T,T2).
35
36 checkLink(A,[]).
37 checkLink(A,[link(H,_,_)|T]):-member(page(H,_),A),!,
38     checkLink(A,T).
```

Per prima viene controllato che sia presente la pagina puntata dall'istruzione `start`, quindi per ogni pagina presente nel librogame viene estratta una lista contenente tutte le istruzioni `link` presenti nelle pagine: per ogni pagina vengono estratte le suddette istruzioni e le istruzioni `if` vengono esplorate (a prescindere dalla validità delle loro condizioni) per estrarre anche `link` eventualmente contenuti nel loro interno.
Ogni pagina genera quindi una lista, contenente i propri link, e queste liste vengono poi concatenate per ottenere un'unica lista con tutti i link di tutte le pagine, su cui poter lavorare.

A questo punto ogni link della lista può essere controllato, per verificare che nel librogame esista una pagina indirizzata da quell'istruzione.

3.3 Correttezza delle variabili embedded

La presenza di variabili nei testi può generare errori, nel caso la variabile abbia un nome non valido. Le variabili contenute nei testi non vengono verificate dal parser perciò, per scongiurare questo, vanno controllate prima di procedere con l'interpretazione.
Poiché è possibile utilizzare anche variabili che non siano state precedentemente dichiarate, l'unico controllo che viene effettuato è assicurarsi che i nomi delle variabili siano compatibilmente corretti.

```
50 checkTexts(A):-
51     findall(I,member(page(_,I),A),Pages),
52     concatTexts(Pages,Texts),
53     checkText(A,Texts).
54
55 concatTexts(Pages,Out):-concatTexts(Pages,Out,[]).
56 concatTexts([],Out,O):-concatLists(O,Out).
57 concatTexts([H|T],Out,R):-getTexts(H,L),
58     (L=[],!,concatTexts(T,Out,R);concatTexts(T,Out,[L|R])).
59
60 getTexts([],[]).
61 getTexts([if(Cond,Body,Else)|T1],T2):-!,
62     append(Body,Else,IF),append(IF,T1,T),getTexts(T,T2).
63 getTexts([link(Page,Text,Body)|T1],[link(Page,Text,Body)|T2]):-
64     !,getTexts(T1,T2).
65 getTexts([text(Tx)|T1],[text(Tx)|T2]):-!,getTexts(T1,T2).
66 getTexts([H|T1],T2):-getTexts(T1,T2).
67
68 checkText(A,[]).
69 checkText(A,[text(Tx)|T]):-atom_chars(Tx,Chars),
70     checkVarsInText(Chars),checkText(A,T).
71 checkText(A,[link(_,Tx,_)|T]):-atom_chars(Tx,Chars),
72     checkVarsInText(Chars),checkText(A,T).
73
74 checkVarsInText([]).
75 checkVarsInText(['\','$'|Chars]):-!,checkVarsInText(Chars).
76 checkVarsInText(['$'|Chars]):-!,
77     getLiteral(Chars,Literal,Rest),checkVarsInText(Rest).
78 checkVarsInText([C|Chars]):-checkVarsInText(Chars).
79
80 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
81     Cod > 96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
82 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
83     Cod > 64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
84 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
85     Cod > 96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
86 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
87     Cod > 64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
88 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
89     Cod > 47,Cod < 58,!,getLiteral2(Chars,Lit,Rest).
90 getLiteral2(Chars,[],Chars).
```

Come si può vedere, il controllo riprende da vicino la stessa struttura di quello precedente. In questo caso però la lista comprende sia le istruzioni `text` che le istruzioni `link`, poiché in entrambe le istruzioni possono esserci variabili embedded nel testo.
Una volta costruita la lista, il testo di ogni istruzione viene esaminato e si controlla che ogni nome di variabile corrisponda ad un letterale valido.

3.4 Avvisi

Mentre i controlli precedenti sono invalidanti, poiché il loro mancato superamento può pregiudicare la corretta esecuzione del librogame, i controlli seguenti invece sono da considerarsi semplicemente degli avvisi al programmatore. Il mancato superamento di uno di questi test non implica in alcun modo l'errata fruizione del librogame, ma può indicare al programmatore di fare attenzione a specifiche situazioni.

Non ci si preoccuperà di fornire un'implementazione di questi test, ma ci si limiterà alla loro presentazione.

3.4.1 Controllo delle pagine orfane

Un librogame è formato da diverse pagine. Mentre è necessario che ad ogni collegamento corrisponda una pagina, nulla vieta che vi siano pagine non puntate da alcun collegamento. Questo però può essere dovuto ad una svista del programmatore, perciò è utile una segnalazione.

Il controllo può essere effettuato in due modi:

- Controllo semplice:
  Si tratta di ottenere una lista di tutte le pagine e controllare, per ognuna di esse, se è presente almeno un'istruzione `start` o `link` che punti a loro.

- Controllo approfondito:
  Il controllo semplice non assicura assenza di pagine orfane, poichè potrebbero esserci istruzioni `link` che non vengono mai eseguite in virtù delle istruzioni `if` e dal valore delle variabili presenti nelle loro condizioni. Per questo motivo occorrerebbe un esplorazione dell'intero albero d'esecuzione del librogame, ma poiché possono essere realizzati cicli, tra le diverse pagine, non si puàssicurare che il controllo termini.

Pertanto tale controllo presenta dei limiti e può essere effettuato solo mediante tecniche di model checking: ad esempio tramite l'esame delle variabili nelle condizioni e un'analisi della loro evoluzione nel corso dell'esecuzione.

3.4.2 Controllo dell'uso di variabili non dichiarate

Come è già stato scritto, le variabili non devono essere necessariamente dichiarate prima di essere utilizzate. Questo però può portare ad errori di programmazione, pertanto dovrebbe essere segnalata ogni variabile che viene letta senza che le sia stato mai assegnato un valore.

- Controllo semplice:
  Si tratta di controllare tutte le istruzioni che possono contenere variabili in lettura e di verificare, per ognuna di queste variabili, se è presente almeno un'istruzione `var` che ne definisce il valore.

- Controllo approfondito:
  Il controllo semplice non garantisce che tutte le variabili impiegate in lettura abbiano un valore assegnato, poiché potrebbe essere prima usata una variabile in lettura non dichiarata precedentemente, e poi quella stessa variabile potrebbe essere ridefinita in un punto successivo dell'esecuzione. Pertanto sarebbe necessario esplorare tutto l'albero di esecuzione, per assicurarsi che le variabili incontrate siano effettivamente state dichiarate prima.
  Questo però porta ai medesimi problemi del controllo approfondito precedente.

Il controllo approfondito degli avvisi, seppur dia maggiori garanzie, comporta diversi problemi. Una scelta può essere quella di portare i suddetti controlli a tempo d'esecuzione, e quindi integrarli invece nell'interprete. Ciò non fornirà certezze sull'assenza dei problemi sopra citati, ma provvederà a segnalarli all'utente, con un avviso puntuale al loro verificarsi.

Capitolo 4

Esecuzione

In questo capitolo verrà trattata l'esecuzione vera e propria del librogame, con le operazioni da realizzare per processare ogni pagina da restituire all'utente, in base alle scelte effettuate durante la lettura.

Verrà quindi realizzato un interprete in grado di eseguire ed esplorare un librogame.

4.1 Modello di esecuzione

L'interpretazione di un librogame comincia da una pagina iniziale e continua fino ad arrivare ad una pagina terminale. Si può perciò considerare come una sequenza di pagine che vengono processate in base alle scelte effettuate nel corso della lettura.
Poiché il valore delle variabili può rendere dinamica una pagina, è necessario il concetto di *stato*, che rappresenti la configurazione del sistema in un particolare istante.
Ogni pagina contiene un insieme di istruzioni che, opportunamente combinate, permettono di definire la pagina da visualizzare. L'esecuzione di ognuna di queste istruzioni determina l'individuazione della prossima istruzione da eseguire (istruzione `IF-ELSE`) o la variazione dello stato finale.

In figura 4.1 è rappresentato il modello d'esecuzione. Può essere identificato come una macchina Prolog che, dato uno stato iniziale e un'istruzione da eseguire, restituisce un nuovo stato. Quindi viene impiegato il nuovo stato ottenuto, come input per l'esecuzione di una nuova istruzione.

Lo stato rappresenta il progresso durante la fruizione del librogame, e tiene traccia della pagina in cui ci si trova, del valore delle variabili usate fino a quel momento, del testo da visualizzare (in relazione all'interpretazione della pagina) e le scelte disponibili.

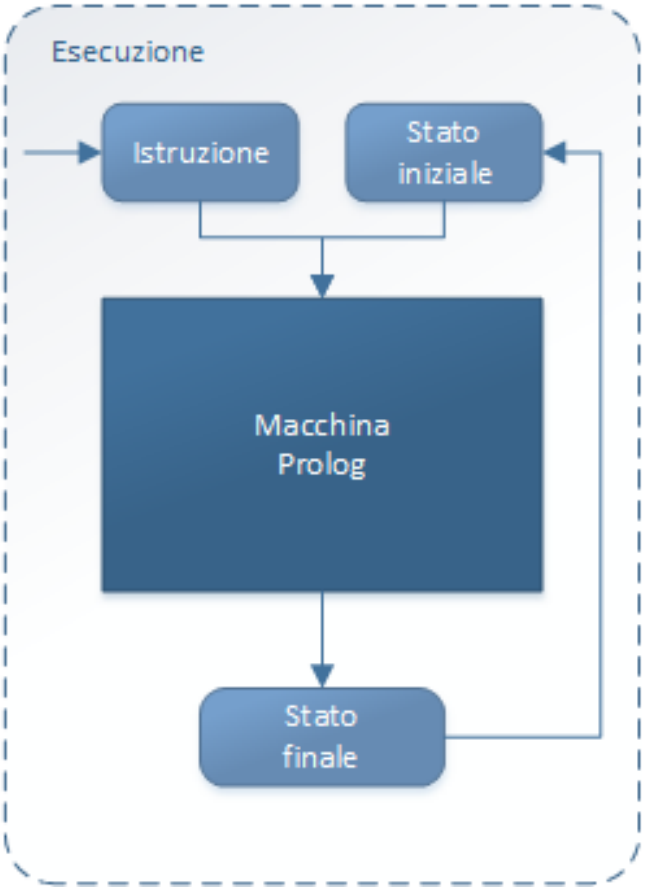

Figura 4.1: Modello di esecuzione

4.2 Esecuzione del librogame

A questo punto è necessario analizzare ogni elemento del linguaggio, esaminando ogni istruzione che l'interprete potrà incontrare. Per prima cosa però bisogna definire la pagina che l'interprete dovrà processare per prima.

4.2.1 Avvio del librogame

```
1 startStory(story(Name, Instructions)) :-
2     retractall(state(_,_,_,_)),
3     member(start(S), Instructions),!,
4     assert(state(S,[],[],[])),
5     exec(page(S), story(Name, Instructions)).
```

La clausola prende in ingresso l'AST del librogame e per prima cosa rimuove ogni eventuale stato della lettura presente in precedenza. Quindi viene individuata l'istruzione `start` all'interno dell'AST, che contiene la pagina da interpretare per prima.
A questo punto si ottiene il primo stato della lettura, che contiene solamente la pagina corrente, ossia quella appena individuata. Si può perciò procedere con l'interpretazione delle istruzioni presenti in quella pagina.

4.2.2 Interpretazione di una pagina

```
1 exec(page(P), story(Name, I)) :-
2     member(page(P, PI), I),!,
3     execP(PI, Texts, Links),
4     retract(state(_, Vars, _, _)),!,
5     assert(state(P, Vars, Texts, Links)).
```

Data una pagina da dover interpretare, per prima cosa viene estratta la sequenza di istruzioni che quella pagina contiene e quindi viene eseguita quella sequenza di istruzioni, ottenendo in questo modo il testo da visualizzare e la lista delle scelte a disposizione dell'utente.
Una volta fatto, viene sostituito il vecchio stato con un nuovo stato, formato dalla pagina appena processata (che diventa la pagina corrente), la lista delle variabili ricavate dall'esecuzione precedente.

4.2.3 Interpretazione delle istruzioni

La lista delle istruzioni di una pagina viene processata finché quella lista non si esaurisce. Come risultato viene restituita la sequenza del testo e lista delle scelte da visualizzare all'utente.

```
execP([],[],[]).
execP([text(T)|PI],[text(Tx)|Texts],Links):-
    bindVarsInText(T,Tx),
    execP(PI,Texts,Links).
execP([link(P,T,LI)|PI],Texts,[link(P,Tx,LI)|Links]):-
    bindVarsInText(T,Tx),
    execP(PI,Texts,Links).
execP([var(V,Exp)|PI],Texts,Links):-
    evalExp(Exp,Result),
    setVar(V,Result),
    execP(PI,Texts,Links).
execP([if(LExp,Body,Else)|PI],Texts,Links):-
    evalLExp(LExp,Result),
    ((  Result =\= 0,!,
        append(Body,PI,NewPI),
        execP(NewPI,Texts,Links)
        );(
        append(Else,PI,NewPI),
        execP(NewPI,Texts,Links)
    )).
```

Per quanto riguarda le istruzioni di tipo `text`, il loro scopo è di aggiungere una stringa alla lista di stringhe del risultato dell'esecuzione della pagina. Prima di farlo però è necessario sostituire le variabili contenute nel testo con il loro valore.
Quando viene individuata una variabile nel testo, si accede allo stato del gioco corrente e si controlla se esiste una variabile con quel nome. In caso affermativo viene restituito il suo valore, altrimenti viene restituito il valore `0`.

```
bindVarsInText(TIn,TOut):-
    atom_chars(TIn,CharsIn),
    searchVarsInText(CharsIn,[],CharsOut),
    atom_chars(TOut,CharsOut).
searchVarsInText([],T,T).
searchVarsInText(['\','$'|Chars],In,TOut):-!,
    append(In,['$'],NewIn),searchVarsInText(Chars,NewIn,TOut).
searchVarsInText(['$'|Chars],In,TOut):-!,
    getLiteral(Chars,Literal,Rest),atom_chars(Var,Literal),
    getVar(Var,Value),text_term(StrValue,Value),
    atom_chars(StrValue,V),append(In,V,NewIn),
    searchVarsInText(Rest,NewIn,TOut).
searchVarsInText([C|Chars],In,TOut):-
    append(In,[C],NewIn),searchVarsInText(Chars,NewIn,TOut).
```

```
15
16 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
17     Cod > 96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
18 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
19     Cod > 64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
20 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
21     Cod > 96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
22 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
23     Cod > 64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
24 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),
25     Cod > 47,Cod < 58,!,getLiteral2(Chars,Lit,Rest).
26 getLiteral2(Chars,[],Chars).
27
28 getVar(V,Value):-
29     state(_,Vars,_,_),
30     ((member(var(V,Value),Vars),!);
31     Value = 0).
```

L'istruzione `link` è analoga alla precedente e aggiunge una scelta alla lista delle scelte del risultato. Anche in questo caso vengono sostituite le variabili eventualmente presenti all'interno del testo delle scelte.

L'istruzione `var` invece processa un'espressione matematica e memorizza il risultato in una variabile. Questa variabile poi viene memorizzata nello stato: se non era già stata memorizzata viene inserita, altrimenti il vecchio elemento viene sostituito da uno nuovo.

```
1  evalExp(num(N),N).
2  evalExp(var(Name),V)    :- getVar(Name,V).
3  evalExp(add(A,B),Out)   :- evalExp(A,OutA),evalExp(B,OutB),
4                             Out is OutA+OutB.
5  evalExp(sub(A,B),Out)   :- evalExp(A,OutA),evalExp(B,OutB),
6                             Out is OutA-OutB.
7  evalExp(mul(A,B),Out)   :- evalExp(A,OutA),evalExp(B,OutB),
8                             Out is OutA*OutB.
9  evalExp(div(A,B),Out)   :- evalExp(A,OutA),evalExp(B,OutB),
10                            Out is OutA div OutB.
11 evalExp(mod(A,B),Out)   :- evalExp(A,OutA),evalExp(B,OutB),
12                            Out is OutA mod OutB.
13 evalExp(rand(A,B),Out) :- evalExp(A,OutA),evalExp(B,OutB),
14     ((
15         OutA < OutB,!,
16         NewOutB is OutB-OutA+1,rand_int(NewOutB,R),Out is R+OutA
17     );(
18         NewOutA is OutA-OutB+1,rand_int(NewOutA,R),Out is R+OutB
19     )).
```

```
20
21 setVar(V,Value):-
22     retract(state(Page,Vars,Texts,Links)),!,
23     ( (member(var(V,_),Vars),!,
24         findall(var(X,Y),(member(var(X,Y),Vars),X\=V),L),
25         assert(state(Page,[var(V,Value)|L],Texts,Links)));
26     assert(state(Page,[var(V,Value)|Vars],Texts,Links))).
```

L'istruzione `if-else` invece serve unicamente per vincolare la prossima istruzione da eseguire al risultato di un'espressione logica.

```
1  evalLExp(num(N),N).
2  evalLExp(var(Name),V):- getVar(Name,V).
3  evalLExp(and(A,B),1):- evalLExp(A,OutA),OutA =\= 0,
4                         evalLExp(B,OutB),OutB =\= 0,!.
5  evalLExp(and(A,B),0).
6  evalLExp(or(A,B),0) :- evalLExp(A,X),evalLExp(B,X),!,X=0.
7  evalLExp(or(A,B),1).
8  evalLExp(not(A),1)  :- evalLExp(A,0),!.
9  evalLExp(not(A),0).
10 evalLExp(eq(A,B),1) :- evalLExp(A,Out),evalLExp(B,Out),!.
11 evalLExp(eq(A,B),0).
12 evalLExp(neq(A,B),0):- evalLExp(A,Out),evalLExp(B,Out),!.
13 evalLExp(neq(A,B),1).
14 evalLExp(gte(A,B),1):- evalLExp(A,OutA),evalLExp(B,OutB),
15                        OutA >= OutB,!.
16 evalLExp(gte(A,B),0).
17 evalLExp(gt(A,B),1) :- evalLExp(A,OutA),evalLExp(B,OutB),
18                        OutA > OutB,!.
19 evalLExp(gt(A,B),0).
20 evalLExp(lte(A,B),1):- evalLExp(A,OutA),evalLExp(B,OutB),
21                        OutA =< OutB,!.
22 evalLExp(lte(A,B),0).
23 evalLExp(lt(A,B),1) :- evalLExp(A,OutA),evalLExp(B,OutB),
24                        OutA < OutB,!.
25 evalLExp(lt(A,B),0).
```

4.2.4 Esecuzione di altri comandi

Un librogame è dotato anche di altre direttive, come `version`, `title` e `info`. Si tratta di comandi descrittivi che possono essere richiesti dall'utente per ottenere maggiori informazioni sul librogame che sta leggendo. Per eseguire questi comandi sono necessarie ulteriori clausole prolog che restituiscono le informazioni contenute in quelle direttive:

```
1 getVersion(story(Name, Instructions), Version) :-
2     member(version(Version), Instructions), !.
3 getTitle(story(Name, Instructions), Title) :-
4     member(title(Title), Instructions), !.
5 getInfo(story(Name, Instructions), Info) :-
6     member(info(Info), Instructions), !.
```

4.3 Considerazioni sull'esecuzione

La generazione dei numeri casuali, realizzata in questo modo, mal si sposa con il salvataggio dello stato. Questo perchè a fronte di risultati casuali sfavorevoli, nel corso della lettura, è sufficiente ricaricare lo stato per ottenere potenzialmente risultati più favorevoli.
Per inibire questa possibilità si può ridefinire una funzione pseudo-casuale che generi un valore partendo da un *seed*, ossia da un numero, generato casualmente al caricamento del librogame. Questo *seed* dovrà poi essere aggiornato ad ogni nuova generazione random nel corso della lettura. Il *seed* dovrà essere quindi salvato nello stato del gioco, per garantire la stessa sequenza di generazioni nel corso della medesima lettura, a prescindere di caricamenti o ripristini dello stato.

A questo punto il codice per l'interprete è completo. Non rimane altro da fare se non provvedere a realizzare un sistema in grado di far interagire l'utente con il librogame in maniera agevole.
Nel prossimo capitolo verrà presentata una semplice applicazione di esempio, in grado di far interagire un lettore con il librogame attraverso un'interfaccia grafica.

Capitolo 5

aStory Player

In questo capitolo verrà proposta una semplice applicazione in Java, capace di interpretare ed eseguire i librigame scritti mediante il linguaggio definito, e quindi in grado di permettere ad un lettore di usufruire del librogame.
Verrà presentata l'applicazione, soffermandosi principalmente sulle sue funzionalità principali e indicandone i punti salienti per l'integrazione tra Prolog e Java.

5.1 Descrizione dell'applicazione

L'applicazione può aprire file `*.astory`, contenti il linguaggio sorgente del librogame da eseguire, oppure file `*.astoryAST` contenenti un *AST* già generato. I secondi possono essere intesi come librigame già testati e pronti per essere distribuiti.
Il librogame viene caricato, quindi la prima pagina viene processata e poi visualizzata all'utente. L'utente a questo punto ha la possibilità di effettuare delle scelte, attraverso dei pulsanti, per guidare il proseguo della lettura e decidere perciò la prossima pagina del librogame da visualizzare.
Il *Player* permette anche di ricominciare la lettura dall'inizio, oppure di memorizzare lo stato della lettura per poi richiamarlo successivamente.
È possibile anche visualizzare le informazioni del librogame aperto.

In figura 5.1 è rappresentata l'applicazione, con caricato un librogame di esempio.

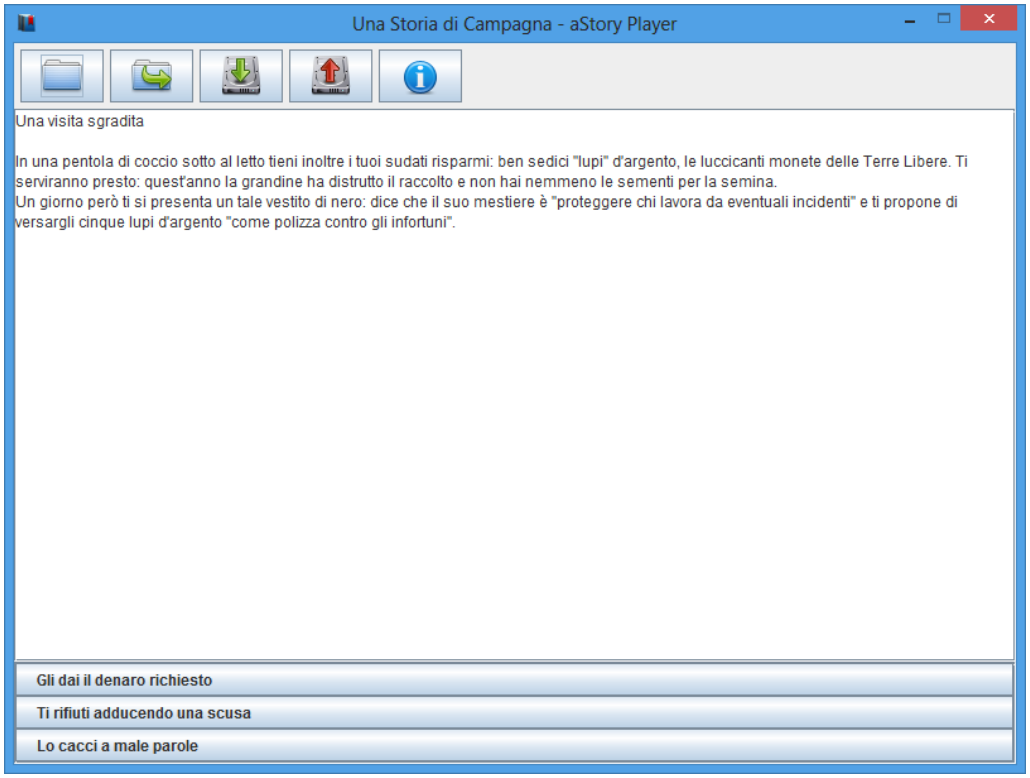

Figura 5.1: AStory Player

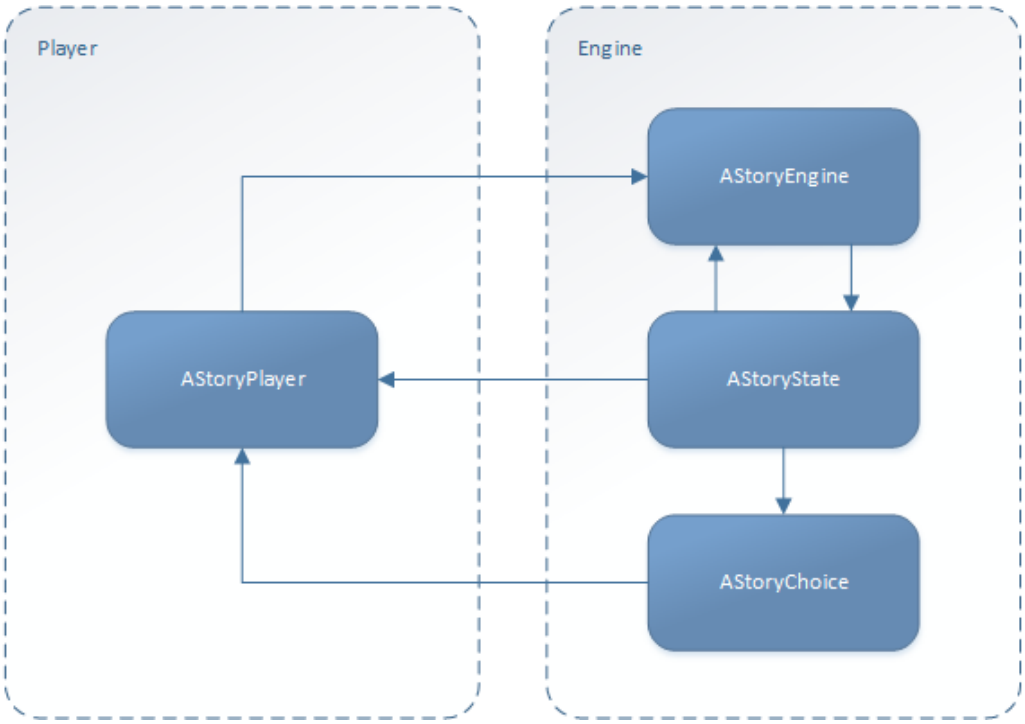

Figura 5.2: Schema di funzionamento

5.2 Schema di funzionamento

Lo schema di funzionamento dell'applicazione è riportato in figura 5.2.

Come si può vedere, l'applicazione è divisa in due parti: la parte *Player* e la parte *Engine*. Il *Player* comanda l'*AStoryEngine*, che rappresenta tutta l'elaborazione compiuta per i librigame. Ad ogni esecuzione, *AStoryEngine* elabora un nuovo *AStoryState*, che rappresenta lo stato della lettura dopo ogni passo di esecuzione (ossia dopo l'esecuzione di ogni pagina). Questo stato viene restituito al *Player*, comprese le scelte possibili per il proseguo della lettura (attraverso *AStoryChoice*), che poi provvederà a fornire la visualizzazione della pagina all'utente.
Lo stato del librogame viene anche impiegato per indicare all'*AStoryEngine* che istruzioni eseguire al nuovo passo di esecuzione e, durante le operazioni di ripristino dello stato del gioco, per ripristinare un progresso salvato in precedenza.

5.3 Integrazione tra Prolog e Java

L'applicazione impiega sia codice Java che codice Prolog. Mentre il primo si occupa principalmente dell'interazione con l'utente (interpretando i suoi comandi e restituendo i risultati a video), il secondo è dedicato all'elaborazione vera e propria per l'esecuzione del librogame.
Tutto ciò si traduce in un'applicazione Java che, mediante il componente *AStoryEngine*, incapsula il codice Prolog da integrare.

L'Engine, alla sua creazione, carica la teoria prolog definita nei precedenti capitoli:

```
1     Theory theory;
2     Prolog engine;
3
4     public AStoryEngine() {
5         [...]
6         engine = new Prolog();
7         try {
8             theory = new Theory(
9                 new FileInputStream("prolog/astory.pl"));
10            engine.setTheory(theory);
11        } catch (IOException | InvalidTheoryException e) {
12            [...]
13        }
14    }
```

5.3.1 Caricamento di un librogame

Una volta avviata, l'applicazione quindi rimane in attesa della scelta dell'utente di caricare un file `*.astory` di un librogame, che viene realizzata attraverso il metodo:

```
1     Term astStory;
2     public AStoryState loadStory(File story){
3         astStory = null;
4         try{
5             SolveInfo siAst = null;
6             if (story.getName().toLowerCase().
7                 endsWith(".astory")){
8                 SolveInfo siTokens = solve(
9                         "text_from_file('"+
10                            story.getParent()+"/"+
11                            story.getName()+"',L),
12                        tokenizer(L,T)."
13                    );
```

```
14                  if (!siTokens.isSuccess()) return null;
15                  Term tokens = siTokens.getTerm("T");
16                  siAst = solve(new Struct(
17                          "parsegen",tokens,new Var("O")
18                      ));
19                  SolveInfo siCheck = solve(new Struct(
20                          "check",siAst.getTerm("O")
21                      ));
22                  if (!siCheck.isSuccess()) return null;
23              }
24              else if (story.getName().toLowerCase().
25                       endsWith(".astoryast")){
26                  siAst = solve(
27                          "retractall(astoryAST(_)),"
28                          + "consult('"+story.getParent()+
29                              "/"+story.getName()+"'),"
30                          + "astoryAST(O)."
31                      );
32              }
33              if (siAst != null && !siAst.isSuccess())
34                  return null;
35
36              astStory = siAst.getTerm("O");
37          } catch (Exception e){
38              [...]
39          }
40          return startStory();
41      }
```

Come si può vedere, il caricamento avviene differentemente a seconda che si cerchi di caricare un file sorgente di tipo `*.astory` o un AST già pronto per essere eseguito.
Nel primo caso vengono effettuati tutti i passaggi e il contenuto del file sorgente viene passato al *tokenizer* e quindi al *parser*, che provvede a generare l'*AST* del librogame. Prima di passare all'esecuzione vera e propria, il *checker* effettua i controlli sul risultato precedente. Quindi viene caricata la prima pagina, attraverso il metodo `startStory()`, e il risultato viene restituito per la visualizzazione.
Nel secondo caso invece si procede direttamente con l'esecuzione.

```
1     public AStoryState startStory(){
2         if (astStory == null) return null;
3         SolveInfo start = solve(new Struct(
4                 "startStory",astStory
5             ));
6         if (!start.isSuccess()) return null;
7         SolveInfo state = solve(
8                 "state(Page,Vars,Texts,Links)."
9             );
10        if (!state.isSuccess()) return null;
11        try {
12            return new AStoryState(
13                    ""+state.getTerm("Page"),
14                    state.getTerm("Vars"),
15                    state.getTerm("Texts"),
16                    state.getTerm("Links"));
17        } catch (NoSolutionException |
18                 UnknownVarException e) {
19            [...]
20        }
21        return null;
22    }
```

Una volta richiamato il predicato Prolog `startStory`, viene esaminato lo stato prodotto e restituito all'ambiente Java mediante un oggetto `AStoryState`, che ne offre una rappresentazione Java e che si occupa di effettuare alcune operazioni di pulitura delle stringhe Prolog ricevute. A questo punto, l'ambiente Java (e quindi l'*AStoryPlayer*) è pronto per visualizzare all'utente la prima pagina del librogame.

5.3.2 Esecuzione di una scelta

Quando viene selezionata una scelta, nel caso sia necessario, per prima cosa viene eseguito il metodo `doActions(Term actions)`, per eseguire operazioni causate dalla scelta selezionata.

```
    public AStoryState doActions(Term actions){
        SolveInfo execP = solve(new Struct(
                "execP",actions,new Var("T"),new Var("L")
            ));
        if (!execP.isSuccess()) return null;
        SolveInfo state = solve(
                "state(Page,Vars,Texts,Links)."
            );
        if (!state.isSuccess()) return null;
        try {
            return new AStoryState(
                ""+state.getTerm("Page"),state.getTerm("Vars"),
                state.getTerm("Texts"),state.getTerm("Links"));
        } catch (NoSolutionException | UnknownVarException e) {
            [...]
        }
        return null;
    }
```

Quindi viene eseguita la pagina corrispondente alla scelta effettuata.

```
    public AStoryState playPage(String page){
        if (astStory == null) return null;
        SolveInfo exec = solve(new Struct(
            "exec",new Struct("page",new Struct(page)),astStory
            ));
        if (!exec.isSuccess()) return null;
        SolveInfo state = solve(
                "state(Page,Vars,Texts,Links)."
            );
        if (!state.isSuccess()) return null;
        try {
            return new AStoryState(
                ""+state.getTerm("Page"),state.getTerm("Vars"),
                state.getTerm("Texts"),state.getTerm("Links"));
        } catch (NoSolutionException | UnknownVarException e) {
            [...]
        }
        return null;
    }
```

5.3.3 Salvataggio e ripristino dello stato della lettura

Le operazioni di salvataggio e ripristino dello stato di lettura, consistono in pratica nella memorizzazione del predicato prolog `state` e nella sua successiva sostituzione.
Queste operazioni vengono realizzate dai metodi `saveState()` e `restoreState()`.

```
1     AStoryState saveState;
2
3     public boolean saveState(){
4         SolveInfo state = solve(
5                 "state(Page,Vars,Texts,Links)."
6             );
7         if (!state.isSuccess()) return false;
8         try {
9             String page = ""+state.getTerm("Page");
10            Term vars   = state.getTerm("Vars");
11            Term texts  = state.getTerm("Texts");
12            Term links  = state.getTerm("Links");
13            saveState = new AStoryState(
14                                    page, vars, texts, links);
15        } catch (NoSolutionException | UnknownVarException e) {
16            [...]
17        }
18        return true;
19    }
20
21    public AStoryState restoreState(){
22        if (saveState == null) return null;
23        SolveInfo retractall = solve(
24                "retractall(state(_,_,_,_))."
25            );
26        if (!retractall.isSuccess()) return null;
27        Struct sState = new Struct(
28            "state",
29            new Struct(saveState.getPage()),
30            saveState.getVars(),
31            saveState.getTexts(),
32            saveState.getLinks()
33        );
34        SolveInfo state = solve(new Struct(
35                "assert",sState
36            ));
37        if (!state.isSuccess()) return null;
38        return saveState;
39    }
```

5.3.4 Visualizzazione delle informazioni del librogame

Come ultima cosa, l'utente ha la possibilità di visualizzare informazioni relative al librogame in esecuzione. Ciò viene effettuato mediante tre metodi analoghi, che permettono di recuperare l'informazione in questione.

```
    public String getVersion(){
        SolveInfo version = solve(new Struct(
                "getVersion",astStory,new Var("V")
            ));
        if (!version.isSuccess()) return null;
        try {
            return  (""+version.getTerm("V")).substring(1,
                    (""+version.getTerm("V")).length()-1);
        } catch (NoSolutionException | UnknownVarException e) {
            [...]
        }
        return null;
    }

    public String getTitle(){
        SolveInfo title = solve(new Struct(
                "getTitle",astStory,new Var("V")
            ));
        if (!title.isSuccess()) return null;
        try {
            return  (""+title.getTerm("V")).substring(1,
                    (""+title.getTerm("V")).length()-1);
        } catch (NoSolutionException | UnknownVarException e) {
            [...]
        }
        return null;
    }

    public String getInfo(){
        SolveInfo info = solve(new Struct(
                "getInfo",astStory,new Var("V")
            ));
        if (!info.isSuccess()) return null;
        try {
            return  (""+info.getTerm("V")).substring(1,
                    (""+info.getTerm("V")).length()-1);
        } catch (NoSolutionException | UnknownVarException e) {
            [...]
        }
        return null;
    }
```

Appendice

Viene riportato per intero il codice Prolog di AStoryEngine.

```
:-load_library('alice.tuprolog.lib.DCGLibrary').

%=========================================%
% TOKENIZER                               %
%=========================================%

% riceve i simboli e restituisce i token
tokenizer(L,Out):-
    atom_codes(L,Codes),
    tokenizer(t,skip,Codes,[],Out).
% cerca caratteri da scartare
tokenizer(Symb,Skip,LI,LO,Out):-
    phrase(Skip,LI,LO1),!,
    tokenizer(Symb,Skip,LO1,LO,Out).
% ottiene un token
tokenizer(Symb,Skip,LI,LO,[A|Out]):-
    P =.. [Symb,A],
    phrase(P,LI,LO1),!,
    tokenizer(Symb,Skip,LO1,LO,Out).
% termina la tokenizzazione
tokenizer(_,_,L,L,[]).

skip --> [32]. % Spazio ' '
skip --> [13]. % CR      \r
skip --> [10]. % NL      \n
skip --> [9].  % TAB     \t
skip --> [12]. % FF      \f formfeed
% Linea di commento:     // ...
skip --> [47],[47],lineComment.      % //
lineComment --> [10].
lineComment --> [X],lineComment.

```

```
33 % Blocco di commento:   /* ... */
34 skip --> [47],[42],blockComment.     % /*
35 blockComment --> [42],blockComment2.   % *
36 blockComment --> [X],blockComment.
37 blockComment2 --> [47].          % /
38 blockComment2 --> [X],blockComment.
39
40 % Parole riservate all'interno del codice sorgente
41 t(else)    --> [101,108,115,101].
42 t(if)      --> [105,102].
43 t(info)    --> [105,110,102,111].
44 t(link)    --> [108,105,110,107].
45 t(page)    --> [112,97,103,101].
46 t(start)   --> [115,116,97,114,116].
47 t(story)   --> [115,116,111,114,121].
48 t(text)    --> [116,101,120,116].
49 t(title)   --> [116,105,116,108,101].
50 t(var)     --> [118,97,114].
51 t(version) --> [118,101,114,115,105,111,110].
52
53 % Espressioni logiche
54 t(eq)  --> [61,61].       % ==
55 t(neq) --> [33,61].       % !=
56 t(gte) --> [62,61].       % >=
57 t(gt)  --> [62].          % >
58 t(lte) --> [60,61].       % <=
59 t(lt)  --> [60].          % <
60 t(not) --> [110,111,116]. % not
61 t(and) --> [97,110,100].  % and
62 t(or)  --> [111,114].     % or
63
64 % Espressioni aritmetiche
65 t(sqlbra) --> [91]. % [
66 t(sqrbra) --> [93]. % ]
67 t(comma)  --> [44]. % ,
68 t(dot)    --> [46]. % .
69 t(plus)   --> [43]. % +
70 t(minus)  --> [45]. % -
71 t(times)  --> [42]. % *
72 t(div)    --> [47]. % /
73 t(mod)    --> [37]. % %
74 t(is)     --> [61]. % =
75
76
77
```

```
78 % Parole composte da lettere e numeri
79 t(str(S))     --> string(S).      % Stringa di testo
80 t(var(VAR))   --> [36],literal(VAR). % Variabile
81 t(num(NUM))   --> numb(NUM).      % Numero
82 t(name(LIT)) --> literal(LIT).   % Nome identificativo
83
84 % Altri simboli e punteggiatura
85 t(colon) --> [59].  % ;
86 t(lbra)  --> [40].  % (
87 t(rbra)  --> [41].  % )
88 t(begin) --> [123]. % {
89 t(end)   --> [125]. % }
90
91 % Numero
92 numb(N)    --> digit(D),numbz(L),{atom_codes(A,[D|L]),
93                (A='0',!,num_atom(N,'-0');num_atom(N,A))}.
94 digit(N)  --> [N],{N>47,N < 58}. % [0..9]
95 numbz([D|L]) --> digit(D),numbz(L). % Sequenza di cifre
96 numbz([]) --> [].
97
98 % Letterale
99 char(N) --> [N],{N>96, N<123}. % [a..z]
100 char(N) --> [N],{N>64, N<91}. % [A..Z]
101 literal(A)  --> char(N),lit(L),
102                 {atom_codes(A,[N|L])}. % Sequenza di lettere
103 lit([N|L2]) --> char(N),lit(L2).
104 % Un letterale puo' contenere anche numeri
105 lit([N|L2]) --> digit(N),lit(L2).
106 lit([]) --> [].
107
108 % Stringa
109 string(T) --> [34],strRest(Y),{atom_codes(T,Y)}.
110
111 % Carattere di escape per valutare correttamente una stringa
112 strRest([92,34|T]) --> [92],[34],strRest(T).
113 %strRest([39,39|T]) --> [39],strRest(T).
114 strRest([92,167|T]) --> [39],strRest(T).
115 strRest([])         --> [34].
116 strRest([X|T])      --> [X],strRest(T).
117
118
119
120
121
122
```

```
123 %=============================================%
124 % PARSER                                      %
125 %=============================================%
126 parsegen(T,A):-phrase(s(A),T),!.
127
128 s(story(N,[H|T])) --> [story],name(N),[begin],elementStory(H),
        storyBodyRest(T),[end].
129 storyBodyRest([])      --> [].
130 storyBodyRest([H|T]) --> elementStory(H),storyBodyRest(T).
131 elementStory(X)    --> instructionStory(X).
132 elementStory(X)    --> page(X).
133
134 instructionStory(start(X))    --> [start],name(X),[colon].
135 instructionStory(version(X)) --> [version],text(X),[colon].
136 instructionStory(title(X))    --> [title],text(X),[colon].
137 instructionStory(info(X))     --> [info],text(X),[colon].
138
139 page(page(N,[H|T]))    --> [page],name(N),[begin],elementPage(H),
        pageBodyRest(T),[end].
140 pageBodyRest([])        --> [].
141 pageBodyRest([H|T])    --> elementPage(H),pageBodyRest(T).
142 elementPage(text(T))   --> [text],text(T),[colon].
143 elementPage(var(V,E)) --> [var],numericExpr(V,E),[colon].
144 elementPage(link(N,T,L))    --> [link],name(N),text(T),link(L),[
        colon].
145 elementPage(if(C,[H|T],E)) --> [if],logicExpr(C),[begin],
        elementPage(H),
146                                 pageBodyRest(T),[end],elsePage(E)
        .
147 elsePage([])      --> [].
148 elsePage([H|T]) --> [else,begin],elementPage(H),pageBodyRest(T)
        ,[end].
149 link([])      --> [].
150 link([H|T]) --> [begin],elementLink(H),linkBodyRest(T),[end].
151 linkBodyRest([])      --> [].
152 linkBodyRest([H|T]) --> elementLink(H),linkBodyRest(T).
153 elementLink(var(V,E))        --> [var],numericExpr(V,E),[colon].
154 elementLink(if(C,[H|T],E)) --> [if],logicExpr(C),[begin],
        elementLink(H),
155                                 linkBodyRest(T),[end],elseLink(E)
        .
156 elseLink([])      --> [].
157 elseLink([H|T]) --> [else,begin],elementLink(H),linkBodyRest(T)
        ,[end].
158
```

```
159 numericExpr(V,E) --> [var(V),is],numExprBody(E).
160 numExprBody(E)    --> numMoltDivElem(H),sumSubOp(H,E).
161 sumSubOp(H,E) --> sumRest(H,E).
162 sumSubOp(H,E) --> subRest(H,E).
163 sumRest(X,X)   --> [].
164 sumRest(A,B)   --> [plus],numExprBody(C),sumRest(add(A,C),B).
165 subRest(X,X)   --> [].
166 subRest(A,B)   --> [minus],numExprBody(C),subRest(sub(A,C),B).
167 numMoltDivElem(E) --> unaryOp(H),mulDivModOp(H,E).
168 mulDivModOp(H,E)  --> mulRest(H,E).
169 mulDivModOp(H,E)  --> divRest(H,E).
170 mulDivModOp(H,E)  --> modRest(H,E).
171 unaryOp(N) --> numValue(N).
172 unaryOp(E) --> [lbra],numExprBody(E),[rbra].
173 unaryOp(rand(A,B)) --> [sqlbra],numValue(A),[comma],numValue(B)
        ,[sqrbra].
174 mulRest(X,X) --> [].
175 mulRest(A,B) --> [times],numExprBody(C),mulRest(mul(A,C),B).
176 divRest(X,X) --> [].
177 divRest(A,B) --> [div],numExprBody(C),divRest(div(A,C),B).
178 modRest(X,X) --> [].
179 modRest(A,B) --> [mod],numExprBody(C),modRest(mod(A,C),B).
180
181 logicExpr(E)   --> logicValue(A),andOrOp(A,E).
182 andOrOp(X,X)   --> [].
183 andOrOp(A,B)   --> [and],logicExpr(C),andOrOp(and(A,C),B).
184 andOrOp(A,B)   --> [or],logicExpr(C),andOrOp(or(A,C),B).
185 logicValue(not(E)) --> [not],logicValue(E).
186 logicValue(L) --> numValue(N),compOp(N,L).
187 logicValue(L) --> [lbra],logicExpr(E),[rbra],compOp(E,L).
188 compOp(X,X)    --> [].
189 compOp(A,L)    --> [eq],logicExpr(B),compOp(eq(A,B),L).
190 compOp(A,L)    --> [neq],logicExpr(B),compOp(neq(A,B),L).
191 compOp(A,L)    --> [gte],logicExpr(B),compOp(gte(A,B),L).
192 compOp(A,L)    --> [gt],logicExpr(B),compOp(gt(A,B),L).
193 compOp(A,L)    --> [lte],logicExpr(B),compOp(lte(A,B),L).
194 compOp(A,L)    --> [lt],logicExpr(B),compOp(lt(A,B),L).
195
196 name(N) --> [name(N)].
197 text(T) --> [str(T)].
198 numValue(num(R)) --> [minus,num(N),dot,num(M)], % Reale negativo
199                      {atom_chars(X,['-',N,'.',M]),num_atom(R,X)
        }.
200 numValue(num(R)) --> [plus,num(N),dot,num(M)],  % Reale positivo
201                      {atom_chars(X,[N,'.',M]),num_atom(R,X)}.
```

```
202 numValue(num(N)) --> [plus,num(N)].                % Intero
        positivo
203 numValue(num(R)) --> [minus,num(N)],               % Intero
        negativo
204                      {atom_chars(X,['-',N]),num_atom(R,X)}.
205 numValue(num(R)) --> [num(N),dot,num(M)],          % Reale positivo
206                      {atom_chars(X,[N,'.',M]),num_atom(R,X)}.
207 numValue(num(N)) --> [num(N)].                     % Intero
        positivo
208 numValue(var(V)) --> [var(V)].
209
210
211
212 %=============================================%
213 % INTERPRETE                                  %
214 %=============================================%
215 startStory(story(Name,Instructions)) :-
216     retractall(state(_,_,_,_)),
217     member(start(S),Instructions),!,
218     assert(state(S,[],[],[])),
219     exec(page(S),story(Name,Instructions)).
220 getVersion(story(Name,Instructions),Version) :-
221     member(version(Version),Instructions),!.
222 getTitle(story(Name,Instructions),Title) :-
223     member(title(Title),Instructions),!.
224 getInfo(story(Name,Instructions),Info) :-
225     member(info(Info),Instructions),!.
226
227 exec(page(P),story(Name,I)) :-
228     member(page(P,PI),I),!,
229     execP(PI,Texts,Links),
230     retract(state(_,Vars,_,_)),!,
231     assert(state(P,Vars,Texts,Links)).
232
233 execP([],[],[]).
234 execP([text(T)|PI],[text(Tx)|Texts],Links):-
235     bindVarsInText(T,Tx),
236     execP(PI,Texts,Links).
237 execP([link(P,T,LI)|PI],Texts,[link(P,Tx,LI)|Links]):-
238     bindVarsInText(T,Tx),
239     execP(PI,Texts,Links).
240 execP([var(V,Exp)|PI],Texts,Links):-
241     evalExp(Exp,Result),
242     setVar(V,Result),
243     execP(PI,Texts,Links).
```

```
244 execP([if(LExp,Body,Else)|PI],Texts,Links):-
245     evalLExp(LExp,Result),
246     ((  Result =\= 0,!,
247         append(Body,PI,NewPI),
248         execP(NewPI,Texts,Links)
249         );(
250         append(Else,PI,NewPI),
251         execP(NewPI,Texts,Links)
252     )).
253
254 getVar(V,Value):-
255     state(_,Vars,_,_),
256     ((member(var(V,Value),Vars),!);
257     Value = 0).
258
259 setVar(V,Value):-
260     retract(state(Page,Vars,Texts,Links)),!,
261     (  (member(var(V,_),Vars),!,
262         findall(var(X,Y),(member(var(X,Y),Vars),X\=V),L),
263         assert(state(Page,[var(V,Value)|L],Texts,Links)));
264     assert(state(Page,[var(V,Value)|Vars],Texts,Links))).
265
266 bindVarsInText(TIn,TOut):- atom_chars(TIn,CharsIn),
267     searchVarsInText(CharsIn,[],CharsOut),atom_chars(TOut,
    CharsOut).
268 searchVarsInText([],T,T).
269 searchVarsInText(['\','$'|Chars],In,TOut):-!,append(In,['$'],
    NewIn),searchVarsInText(Chars,NewIn,TOut).
270 searchVarsInText(['$'|Chars],In,TOut):-!,
271     getLiteral(Chars,Literal,Rest),atom_chars(Var,Literal),
    getVar(Var,Value),text_term(StrValue,Value),atom_chars(
    StrValue,V),
272     append(In,V,NewIn),searchVarsInText(Rest,NewIn,TOut).
273 searchVarsInText([C|Chars],In,TOut):-append(In,[C],NewIn),
    searchVarsInText(Chars,NewIn,TOut).
274
275 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),Cod >
    96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
276 getLiteral([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),Cod >
    64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
277 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),Cod >
    96,Cod < 123,!,getLiteral2(Chars,Lit,Rest).
278 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),Cod >
    64,Cod < 91,!,getLiteral2(Chars,Lit,Rest).
279 getLiteral2([C|Chars],[C|Lit],Rest):-atom_codes(C,[Cod]),Cod >
```

```
    47,Cod < 58,!,getLiteral2(Chars,Lit,Rest).
280 getLiteral2(Chars,[],Chars).
281
282 evalExp(num(N),N).
283 evalExp(var(Name),V):-getVar(Name,V).
284 evalExp(add(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),Out is
    OutA+OutB.
285 evalExp(sub(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),Out is
    OutA-OutB.
286 evalExp(mul(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),Out is
    OutA*OutB.
287 evalExp(div(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),Out is
    OutA div OutB.
288 evalExp(mod(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),Out is
    OutA mod OutB.
289 evalExp(rand(A,B),Out):-evalExp(A,OutA),evalExp(B,OutB),
290     ((
291         OutA < OutB,!,
292         NewOutB is OutB-OutA+1,rand_int(NewOutB,R),Out is R+OutA
293     );(
294         NewOutA is OutA-OutB+1,rand_int(NewOutA,R),Out is R+OutB
295     )).
296
297 evalLExp(num(N),N).
298 evalLExp(var(Name),V):-getVar(Name,V).
299 evalLExp(and(A,B),1):-evalLExp(A,OutA),OutA =\= 0,evalLExp(B,
    OutB),OutB =\= 0,!.
300 evalLExp(and(A,B),0).
301 evalLExp(or(A,B),0):-evalLExp(A,X),evalLExp(B,X),!,X=0.
302 evalLExp(or(A,B),1).
303 evalLExp(not(A),1):-evalLExp(A,0),!.
304 evalLExp(not(A),0).
305 evalLExp(eq(A,B),1):-evalLExp(A,Out),evalLExp(B,Out),!.
306 evalLExp(eq(A,B),0).
307 evalLExp(neq(A,B),0):-evalLExp(A,Out),evalLExp(B,Out),!.
308 evalLExp(neq(A,B),1).
309 evalLExp(gte(A,B),1):-evalLExp(A,OutA),evalLExp(B,OutB),OutA >=
    OutB,!.
310 evalLExp(gte(A,B),0).
311 evalLExp(gt(A,B),1):-evalLExp(A,OutA),evalLExp(B,OutB),OutA >
    OutB,!.
312 evalLExp(gt(A,B),0).
313 evalLExp(lte(A,B),1):-evalLExp(A,OutA),evalLExp(B,OutB),OutA =<
    OutB,!.
314 evalLExp(lte(A,B),0).
```

```
315 evalLExp(lt(A,B),1):-evalLExp(A,OutA),evalLExp(B,OutB),OutA <
        OutB,!.
316 evalLExp(lt(A,B),0).
317
318 %=============================================%
319 % CHECKER                                     %
320 %=============================================%
321 check(story(Name,I)):-checkStart(I),checkISTV(I),checkDupPages(I
        ),checkLinks(I),checkTexts(I).
322 checkStart(A):-member(start(_),A),!.
323 checkISTV(A):-
324       findall(info(I),member(info(I),A),LI),length(LI,NLI),NLI <
        2,
325       findall(start(S),member(start(S),A),LS),length(LS,NLS),NLS <
        2,
326       findall(title(T),member(title(T),A),LT),length(LT,NLT),NLT <
        2,
327       findall(version(V),member(version(V),A),LV),length(LV,NLV),
        NLV < 2.
328 checkDupPages(A):-
329       findall(page(X),member(page(X,_),A),Pages),length(Pages,NP),
330       setof(page(X),member(page(X),Pages),SPag),!,length(SPag,NP).
331 checkLinks(A):-
332       member(start(S),A),!,member(page(S,_),A),!,
333       findall(I,member(page(_,I),A),Pages),concatLinks(Pages,Links
        ),
334       checkLink(A,Links).
335 concatLinks(Pages,Out):-concatLinks(Pages,Out,[]).
336 concatLinks([],Out,O):-concatLists(O,Out).
337 concatLinks([H|T],Out,R):-getLinks(H,L),(L=[],!,concatLinks(T,
        Out,R);concatLinks(T,Out,[L|R])).
338 getLinks([],[]).
339 getLinks([if(Cond,Body,Else)|T1],T2):-!,append(Body,Else,IF),
        append(IF,T1,T),getLinks(T,T2).
340 getLinks([link(Page,Text,Body)|T1],[link(Page,Text,Body)|T2])
        :-!,getLinks(T1,T2).
341 getLinks([H|T1],T2):-getLinks(T1,T2).
342 concatLists([],[]).
343 concatLists([L|T],T2):-list(L),!,append(L,T,TN),concatLists(TN,
        T2).
344 concatLists([H|T],[H|T2]):-concatLists(T,T2).
345 checkLink(A,[]).
346 checkLink(A,[link(H,_,_)|T]):-member(page(H,_),A),!,checkLink(A,
        T).
347
```

```
348 checkTexts(A):-
349      findall(I,member(page(_,I),A),Pages),concatTexts(Pages,Texts
         ),
350      checkText(A,Texts).
351 concatTexts(Pages,Out):-concatTexts(Pages,Out,[]).
352 concatTexts([],Out,O):-concatLists(O,Out).
353 concatTexts([H|T],Out,R):-getTexts(H,L),(L=[],!,concatTexts(T,
        Out,R);concatTexts(T,Out,[L|R])).
354 getTexts([],[]).
355 getTexts([if(Cond,Body,Else)|T1],T2):-!,append(Body,Else,IF),
        append(IF,T1,T),getTexts(T,T2).
356 getTexts([link(Page,Text,Body)|T1],[link(Page,Text,Body)|T2])
        :-!,getTexts(T1,T2).
357 getTexts([text(Tx)|T1],[text(Tx)|T2]):-!,getTexts(T1,T2).
358 getTexts([H|T1],T2):-getTexts(T1,T2).
359 checkText(A,[]).
360 checkText(A,[text(Tx)|T]):-atom_chars(Tx,Chars),checkVarsInText(
        Chars),checkText(A,T).
361 checkText(A,[link(_,Tx,_)|T]):-atom_chars(Tx,Chars),
        checkVarsInText(Chars),checkText(A,T).
362
363 checkVarsInText([]).
364 checkVarsInText(['\','$'|Chars]):-!,checkVarsInText(Chars).
365 checkVarsInText(['$'|Chars]):-!,getLiteral(Chars,Literal,Rest),
        checkVarsInText(Rest).
366 checkVarsInText([C|Chars]):-checkVarsInText(Chars).
```

Elenco delle figure