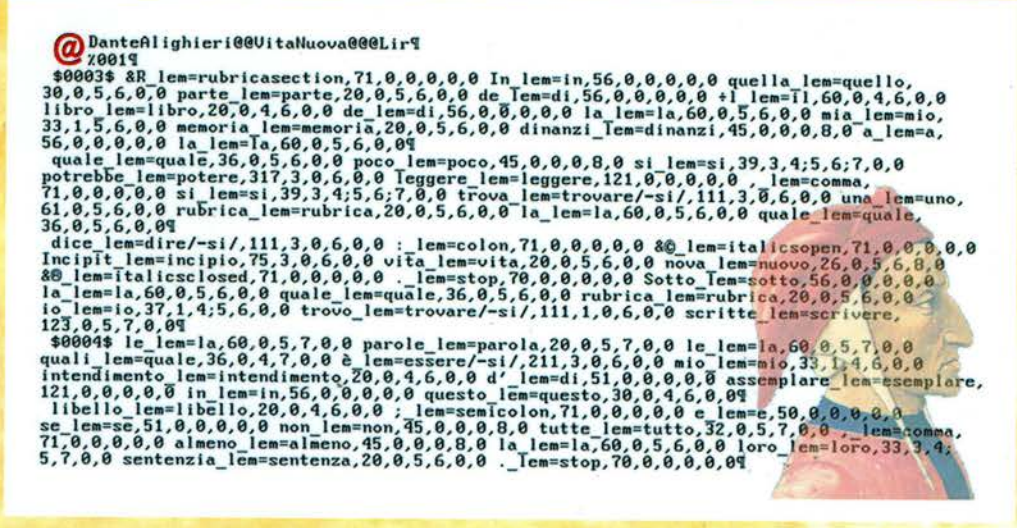
MANUEL BARBERA

Schema e storia del *"Corpus Taurinense"*

Linguistica dei *corpora* dell'italiano antico



Edizioni dell'Orso

Manuel Barbera

Schema e storia del *Corpus Taurinense*: linguistica dei corpora dell'italiano antico

Edizioni dell'Orso
Alessandria

15100 Alessandria, via Rattazzi 47
Tel. 0131.252349 - Fax 0131.257567
E-mail: info@ediorso.it
http: //www.ediorso.it

Allestimento informatico: Arun Maltese (bear.am@savonaonline.it)

ISBN 978-88-6274-139-2

21. La *release.* *Specifiche ed utilizzo del CT.*

21.0 INTRODUZIONE. In questo conclusivo capitolo vedremo di tirare riassuntivamente le fila di tutte le informazioni fondamentali sul corpus e sul suo funzionamento che abbiamo sparsamente introdotto, discusso ed approfondito nel corso di questo volume.

Questo significa, da un lato, fornire una sorta di "*quick reference guide*" (come si direbbe per altri generi di software) delle principali specifiche del corpus, cui l'utente possa fare brevemente riferimento in caso di bisogno (§ 21.1), e dall'altro un prontuario per la costruzione di query del corpus nella sintassi CQP, altrettanto breve e "pratico" (§ 21.2). Una versione web in inglese almeno del primo, tra l'altro, è da sempre stata a disposizione sulla homepage del CT.

21.1 LE SPECIFICHE DEL CT: UNA BREVE GUIDA DI RIFERIMENTO. Raccogliamo qui le informazioni di base sulla struttura e la consistenza materiale del *Corpus Taurinense* che più possa essere utile al lettore / utilizzatore del corpus avere schematicamente sott'occhio.

La versione inglese di questi materiali, disponibile online fin dal lontano 29 agosto 2000 (l'ultima revisione è stata del 1 novembre 2008) è la *CT Specifications Guide*.

21.1.0 LE CIFRE PRINCIPALI DEL CT VER. 1.8. I dati quantitativi della corrente versione del CT sono i seguenti (cfr. per una presentazione più analitica il § 21.1.5.3 *infra*):

token	257.192
type	18.883
lemmi	8.319
dimensione[1]	15,6 MB
attributi posizionali	12
attributi strutturali	10

Tav. 249: Le principali cifre del CT.

21.1.1 ATTRIBUTI STRUTTURALI E POSIZIONALI (CT VER. 1.8). L'ossatura fondamentale di un corpus in formato-CWB è data dalla struttura degli attributi che vi sono stati definiti (cfr. § 19.1.2.2; per la creazione di questa struttura di attributi, cfr. invece § 20.1.3), che ne disegnano le condizioni generali di interrogabilità.

Pertanto, liminarmente, la Tav. 250 *infra* ricapitola attributi posizionali e strutturali del CT.

21.1.2 VALORI DEGLI ATTRIBUTI RELATIVI AL TAGSET (CT-TAGSET VER. 1.4). L'altro aspetto caratteristico del CT è il POS-tagging: il CT-tagset (Ver. 1.4) era stato largamente discusso nel ¶ 4, dove già ne era stata data una FD (*Feature Declaration*), cfr. § 4.3 ecc.

La lista delle HDF (*Hierarchy Defining Features*) delle POS del CT-tagset Ver. 1.4 è ripresentata nella Tav. 251 *infra* sotto forma di lista degli attributi posizionali `kat` e `pos`, con *mapping* internotazionale[2]; la ExN è qui presente in forma non dichiarativa (ossia di ogni type è stato direttamente fornito il *value*)[3].

---

[1] Prima dell'Encode.

[2] Per la differenza tra "notazione condensata" (CdN *Condensed Notation*), "notazione breve" (ShN *Short Notation*) e "notazione estesa" (ExN *Extended Notation*) cfr. § 4.0.2.1.

[3] Per una versione dichiarativa completa cfr. il *mapping* internotazionale fornito nel § 4.3.1 (Tav.53).

attributi		*descrizione*	*esempio*	*note, valori*
P	**word**	il token	mangia	... ...
	lemma	il lemma cui il token è stato ricondotto	mangiare	cfr. lista separata[4]
	pos	il POS-tag con le sue HDF (Hierarchy-Defining Features) in ShN (Short Notation)	v.m.f.ind.pr	cfr. Tav.251
	kat	• i codici di MSF (Morphosyntactic Features) • i codici di HCF (Hierarchy-Collapsed Features)	3,0,6,0,0 111	cfr. Tav. 252 cfr. Tav. 251
	typ	il "tipo testuale" di un (-a parte di) testo: {prosa; verso; rubrica}; incrociato con *type*	/P	P V R
	corr	se la forma del token muta nel ms. o nell'edizione	y	y n
	genre	il genere letterario di un testo: {documentario, didattico, storico, narrativo, lirico}; incrociato con *genr*	nar	Doc Did Stor Nar Lir
	msform	il token inalterato che compare *de facto* nel ms.	magia	... ...
	philform	la resa filologica del token, con i diacritici consueti (parentesi tonde e quadre, corsivo)	ma(n)gia; ma[n]gia; ma¦n¦gia	cfr. § 21.1.4
	mwlword	il lemma-MW	a°÷l°postutto	-- cfr. lista separata[5]
	mwlkat	la pos-MW	45	-- cfr. Tav.251
	mwlnum	la successione dei token-MW	1	0 {1;2;n}
S	**author**	l'autore di un testo compreso nel corpus	Anonimo	cfr. § 21.1.3
	title	il titolo di un testo compreso nel corpus	Novellino	cfr. § 21.1.3
	chapter	il num. di capitolo di un testo compreso nel corpus	*n*	... ...
	par	il num. di paragrafo di un testo compreso nel corpus	*n*	... ...
	s	confine di frase	*n*	... ...
	line	il numero di riga (di una pagina) dell'edizione a stampa di un testo compreso nel corpus	*n*	... ...
	page	il numero di pagina dell'edizione a stampa di un testo compreso nel corpus	*n*	... ...
	type	rimando incrociato con *typ*	/P	P V R
	genr	rimando incrociato con *genre*	nar	Doc; Did; Stor; Nar; Lir
	mwl	confini di MW	mwl	mwl

Tav. 250: Gli attributi posizionali e strutturali del CT.

[4] Il lemmario completo del CT è presente sul sito.
[5] Il lemmario-MW del CT è del pari disponibile sul sito.

	kat	**pos** – ShN	**pos** – ExN
nome	20	n.c	noun.common
	21	n.p	noun.proper
agg.	26	adj	adjective
pronome & determinante	30	pd.dem.s	pro-det.demonstrative.strong
	31	pd.dem.w	pro-det.demonstrative.weak
	32	pd.ind	pro-det.indefinit.
	33	pd.pos.s	pro-det.possessive.strong
	34	pd.pos.w	pro-det.possessive.weak
	35	pd.int	pro-det.interrogative
	36	pd.rel	pro-det.relative
	37	pd.per.s.no	pro-det.personal.strong.nominative
	38	pd.per.s.ob	pro-det.personal.strong.oblique
	39	pd.per.w.ob	pro-det.personal.weak.oblique
	40	pd.exc	pro-det.exclamative
	41	pd.per.w.no	pro-det.personal.weak.nominative
avverbio	45	adv.gn	adverb.general
	46	adv.pc	adverb.particle
	47	adv.cnt	adverb.connective
congiunzione	50	conj.co	conjunction.coordinative
	51	conj.sb	conjunction.subordinative
adposizione	56	adp.pre	adposition.preposition
	57	adp.post	adposition.postposition
articolo	60	art.d	article.determinative
	61	art.i	article.indeterminative
numerale	64	num.car	numeral.cardinal
	65	num.ord	numeral.ordinal
interiezione	68	intj	interjection
interpunzione	70	punct.fi	punctuation.final
	71	punct.nfi	punctuation.non-final
residui	75	r.frg	residual.foreign
	76	r.abb	residual.abbreviation
	77	r.for	residual.formulae
	78	r.epe	residual.epenthesis
verbo (principale)	111	v.m.f.ind.pr	verb.main.finite.indicative.present
	112	v.m.f.ind.ipf	verb.main.finite.indicative.imperfect
	113	v.m.f.ind.pt	verb.main.finite.indicative.past
	114	v.m.f.ind.ft	verb.main.finite.indicative.future
	115	v.m.f.sub.pr	verb.main.finite.subjunctive.present
	116	v.m.f.sub.ipf	verb.main.finite.subjunctive.imperfect
	117	v.m.f.cnd.pr	verb.main.finite.conditional.present

	118	v.m.f.imp.pr	verb.main.finite.imperative.present
	121	v.m.nf.inf.pr	verb.main.non-finite.infinitive.present
	122	v.m.nf.par.pr	verb.main.non-finite.participle.present
	123	v.m.nf.par.pt	verb.main.non-finite.participle.past
	124	v.m.nf.ger.pr	verb.main.non-finite.gerunde.present
verbo (ausiliare)	211	v.a.f.ind.pr	verb.auxiliar.finite.indicative.present
	212	v.a.f.ind.ipf	verb.auxiliar.finite.indicative.imperfect
	213	v.a.f.ind.pt	verb.auxiliar.finite.indicative.past
	214	v.a.f.ind.ft	verb.auxiliar.finite.indicative.future
	215	v.a.f.sub.pr	verb.auxiliar.finite.subjunctive.present
	216	v.a.f.sub.ipf	verb.auxiliar.finite.subjunctive.imperfect
	217	v.a.f.cnd.pr	verb.auxiliar.finite.conditional.present
	218	v.a.f.imp.pr	verb.auxiliar.finite.imperative.present
	221	v.a.nf.inf.pr	verb.auxiliar.non-finite.infinitive.present
	222	v.a.nf.par.pr	verb.auxiliar.non-finite.participle.present
	223	v.a.nf.par.pt	verb.auxiliar.non-finite.participle.past
	224	v.a.nf.ger.pr	verb.auxiliar.non-finite.gerunde.present
verbo (modale)	311	v.md.f.ind.pr	verb.modal.finite.indicative.present
	312	v.md.f.ind.ipf	verb.modal.finite.indicative.imperfect
	313	v.md.f.ind.pt	verb.modal.finite.indicative.past
	314	v.md.f.ind.ft	verb.modal.finite.indicative.future
	315	v.md.f.sub.pr	verb.modal.finite.subjunctive.present
	316	v.md.f.sub.ipf	verb.modal.finite.subjunctive.imperfect
	317	v.md.f.cnd.pr	verb.modal.finite.conditional.present
	318	v.md.f.imp.pr	verb.modal.finite.imperative.present
	321	v.md.nf.inf.pr	verb.modal.non-finite.infinitive.present
	322	v.md.nf.par.pr	verb.modal.non-finite.participle.present
	323	v.md.nf.par.pt	verb.modal.non-finite.participle.past
	324	v.md.nf.ind.pr	verb.modal.non-finite.gerunde.present

Tav. 251: `kat` (HDF & MSF; CdN) e `pos` (HDF; ShN & ExN) del CT.

La lista delle MSF (*MorphoSyntactic Features*) del CT-tagset Ver. 1.4, poi, è ripresentata in Tav. 252 sotto forma di lista degli attributi `kat`, con mapping in ExN (*Extended Notation*) in versione dichiarativa e specificazione dell'ordine posizionale nel bastone di annotazione (cfr. § 6.3). Le MSF, ricordo peraltro, erano già state teorizzate nel § 4.0.3.2, discusse per il CT-tagset nel § 4.2.1 e raccolte nella prima tavola del § 4.3.1.

21.1.3 VALORI DEGLI ATTRIBUTI RELATIVI AI TESTI (CT VER. 1.8). Gli altri attributi del CT che è utile avere qui raccolti sinotticamente (Tav. 253 *infra*) sono quelli relativi ai testi costitutivi del CT: ossia i posizionali `author`, `title`, `genre` e `typ` e gli strutturali `genr` e `type`.

La lista particolareggiata dei 22 testi compresi nel CT era peraltro già stata fornita nel § 3.0.2, ed i dettagli bibliografici sono comunque presenti in bibliografia (§ 25.2).

		kat *(MSF)*	***ExN***	*ordine*
MSF	**persona**	1	pers=1	posizione 1
		2	pers=2	
		3	pers=3	
	genere	4	gend=masc	posizione 2
		5	gend=fem	
		4;5	gend=c	
	numero	6	numb=sg	posizione 3
		7	numb=pl	
		6;7	numb=n	
	grado	8	degr=pos	posizione 4
		9	degr=comp	
		10	degr=sup	
	multiword	11-19	loc=11-19	posizione 5

Tav. 252: Le MSF del CT in ShN (kat) e ExN..

author	***title***	***g/g***	***t/t***
MaestroRinuccino	Sonetti	lir	V
BonoGiamboni	LibroViziVirtù	did	P
BonoGiamboni	TrattatoViziVirtù	did	P
BrunettoLatini	Favolello	did	V
BrunettoLatini	Tesoretto	did	V
BrunettoLatini	Rettorica	did	P
Anonimi	CapitoliCompagniaSanGilio(Statuti84)	doc	P
DanteAlighieri	VitaNuova	lir	P V
Anonimi	CapitoliCompagniaMadonnaOrsanmichele(Statuti94)	doc	P
Anonimi	CapitoliCompagniaMadonnaOrsanmichele(Statuti97)	doc	P
ConsiglioDe'Cerchi	Lettera	doc	P
Consiglio&LapoDe'Cerchi	Lettera	doc	P
CastraGualfredi&c	LibroDareEdAvere	doc	P
LapoRiccomanni	LibroDareEdAvere	doc	P
Anonimo	FioreDiFilosafi	nar	P
Anonimi	LibroOrdinamentiCompagniaSMariaCarmine(Statuti80)	doc	P
Anonimo	CronicaFiorentina	stor	P
Anonimo	VolgarizzamentoDisciplinaClericalis	nar	P
Anonimo	Novellino	nar	P V
GuidoCavalcanti(?)	DueBallate(I'Vidi/SolPerPietà)	lir	P
GuidoCavalcanti	Rime	lir	V
JacopoCavalcanti	TreSonetti	lir	V

Tav. 253: Gli attributi author e title, ed i loro accoppiamenti con genre/genr e typ/type.

21.1.4 CONVENZIONI ED AVVERTENZE SPECIALI (CT VER. 1.8). Restano alcune convenzioni ed avvertenze che è bene avere ancora sott'occhio in una *quick reference guide* del CT come la presente.

In primo luogo sono da ricordare alcuni simboli usati sistematicamente nel CT, qui elencati insieme al loro "nome" tipografico internazionale, alla spiegazione del fenomeno che simboleggiano e ad un esempio dal corpus:

simboli	"nome"	*fenomeno*	*esempio (e glosse)*
¬	*logicalnot*	composti	`Mei¬di¬donna`
		fonosintassi di n\d	`nonn¬ è, foglia¬d è`
¦	*brokenbar*	corsivi filologici	`or¦o¦`
·	*periodcente*r	proclisi+assimilazione	`de· regno, Be· ll' ho`
÷	*divide*	grafoclisi	`porta ÷l ÷te ÷ne`
~	*tilde*	compendi	`ka~ agosto`
^	*caret*	ellissi	`^^^` (corrisponde all'usuale "...")
[^]	*caret tra quadr*e	lacuna	`molto [^^^^] francamente`
×	*multiply*	deperdita	`××osta` (caratteri illeggibili)
*	*asteris*k	vacua	`die *** d' aprile` (spazio bianco nel ms.)
Ø	*Oslash*	morfemi zero	`a ÷Ø demonî` ('ai demonii')
©...®	*copyright... register*	corsivi tipografici	`le credenze del © Credo in Deo ®`
(...)	*round brackets*	abbreviazioni sciolte	`mante(n)gna`
[...]	*square brackets*	integrazioni	`rag[g]io`
{...}	*brace brackets*	simboli grafici	`{SN}` ('signum notarii')

Tav. 254: Simboli convenzionali usati nel CT.

In secondo luogo è bene ricordare le seguenti quattro convenzioni:

(j) i **grafoclitici** sono trattati come token individuali marcati dal *divide* (ASCII 246 = ANSI 247) "÷";

(ij) i **lemmi** sono introdotti dalla formula `lemma=`;

(iij) **cassazioni** ed **espunzioni** non sono state incluse nel testo del CT ma solo in `msform` e/o `philform`;

(iiij) il **bastone di annotazione** è della forma "*token*_lemma=*lemma*,HDF/HCF,MSF1, MSF2,MSF3,MSF4,MSF5".

21.1.5 LE CIFRE COMPLESSIVE DEL CT SECONDO "CWB -DESCRIBE-CORPUS". Viste le cifre generali del CT (§ 21.1.0) e l'architettura complessiva dei suoi attributi (§ 21.1.1), possiamo ora fornire i dati quantitativi analitici per ciascun tipo di posizione del CT. Per fare ciò il CWB dispone di un apposito comando.

21.1.5.1 IL "CWB-DESCRIBE-CORPUS". [MT] Una volta terminata con successo la procedura di preparazione del corpus, con la relativa installazione dello stesso in un'area della

memoria di massa appositamente creata allo scopo, è infatti possibile estrarre tutte le informazioni pertinenti la struttura del corpus in oggetto, operazione di primaria importanza non fosse che per finalità di controllo della coerenza strutturale del prodotto. Per ordinare al sistema di fornire tali informazioni, l'utente dovrà fare ricorso al comando del Corpus Work-Bench `cwb-describe-corpus` che, è bene ricordare, pur facendo parte della piattaforma di gestione CWB, risulta a tutti gli effetti estraneo al sistema di indagine CQP, e pertanto irraggiungibile anche dall'interfaccia web che abbiamo predisposto per il CT online (cfr. § 20.3).

In merito al funzionamento pratico, è importante ed utile segnalare che il comando in questione consente di ottenere i dati in due formati: uno di base ed uno più dettagliato.

21.1.5.2 IL "CWB -DESCRIBE-CORPUS" BASICO. [MT] Digitando il comando senza alcun elemento aggiuntivo, verranno restituiti unicamente i dati relativi al posizionamento del corpus all'interno dell'elaboratore, la dimensione del corpus espressa in termini di quantità complessiva di token e, infine, la quantità totale di attributi posizionali e strutturali, con il relativo elenco delle voci dichiarate in fase di *encoding*.

Nel caso dell'ultima installazione della corrente versione 1.8 che abbiamo fatto in locale in bmanuel.org, questa era la schermata che il sistema restituisce:

```
============================================================
Corpus: ant2009
============================================================

description:    CT18=ItalAnt2009, Apr 2009
registry file:  /usr/local/cwb-3.0/registry/ant2009
home directory: /usr/etc/corpora/ant2009/
info file:      (none)
size (tokens):  257185

 12 positional attributes:
      word            lemma           pos             kat
      typ             corr            genre           msform
      philform        mwlword         mwlkat          mwlnum

 10 structural attributes:
      author          title           chapter         s
      line            page            par             type
      genr            mwl

  0 alignment  attributes:
```

Tav. 255: Descrizione basica del CT fornita da `cwb-describe-corpus`.

21.1.5.3 IL "CWB-DESCRIBE-CORPUS" ESPANSO. [MT] Per contro, specificando il parametro `-s` al comando di cui sopra, si otterrà l'espansione della sezione dedicata all'elenco delle voci appartenenti agli attributi strutturali e posizionali, con la relativa esplicitazione, per ogni entità presente, della quantità totale di type e token.

Nel caso dell'ultima installazione della corrente versione 1.8 che abbiamo fatto in locale questa è la schermata che il sistema, similarmente a quella presentata in Tav. 255, restituisce con il comando `cwb-describe-corpus -s`:

```
============================================================
Corpus: ant2009
============================================================

description:    CT18=ItalAnt2009, Apr 2009
registry file:  /usr/local/cwb-3.0/registry/ant2009
home directory: /usr/etc/corpora/ant2009/
info file:      (none)
size (tokens):  257185

 12 positional attributes
 10 structural attributes
  0 alignment  attributes

p-ATT word                257185 tokens,    18876 types
p-ATT lemma               257185 tokens,     7584 types
p-ATT pos                 257185 tokens,       69 types
p-ATT kat                 257185 tokens,      429 types
p-ATT typ                 257185 tokens,        4 types
p-ATT corr                257185 tokens,        3 types
p-ATT genre               257185 tokens,        7 types
p-ATT msform              257185 tokens,    19107 types
p-ATT philform            257185 tokens,    19235 types
p-ATT mwlword             257185 tokens,     1231 types
p-ATT mwlkat              257185 tokens,      155 types
p-ATT mwlnum              257185 tokens,        1 types
s-ATT author                  22 regions (with annotations)
s-ATT title                   22 regions (with annotations)
s-ATT chapter                557 regions (with annotations)
s-ATT s                     7642 regions (with annotations)
s-ATT line                 25065 regions (with annotations)
s-ATT page                  1303 regions (with annotations)
s-ATT par                   4496 regions (with annotations)
s-ATT type                  1228 regions (with annotations)
s-ATT genr                    22 regions (with annotations)
s-ATT mwl                      0 regions (with annotations)
```

Tav. 256: Descrizione analitica del CT fornita da `cwb-describe-corpus-s`.

Da una rapida analisi di questi dati, balzeranno agli occhi due apparenti anomalie:

(1) la discrepanza tra il numero dei lemmi (7.584) estratto dal “cwb-describe” e quello (8.319) ricavato dal nostro lemmario;

(2) la totale assenza, evidenziata nell’ultima riga, di aree definite come multiword a livello di attributo strutturale.

Quanto ad (1), ciò dipende dal fatto che il CWB non disambigua i lemmi, così quelli che per noi sono 5 lemmi distinti (es. 2067) per CWB sono solo 2:

[2067] `volere 2 18 n.c`
`volere 66 409 v.md`
`volere/-si/ 11 182 v.md`
`volgare 2 7 adj`
`volgare 2 15 n.c` *lemmario.*

Quanto a (2), la cosa, in realtà, non deve stupire: proprio per aggirare le problematiche descritte nei §§ 20.1 sgg., la gestione delle MW si è basata su attributi posizionali, facilmente accessibili e interrogabili, evitando di conseguenza il ricorso alle entità strutturali (che sono state previste ma non implementate).

21.2 IL LINGUAGGIO CQP: BREVE GUIDA ALLA QUERY DEL CT. Se in generale per illustrare la potenza di interrogazione del CWB dovrebbe essere sufficiente quanto raccolto nel ¶ 19, per agevolmente interrogare il CT nel CWB bisogna tenere conto almeno di due fattori: da un lato (§ 21.2.1) la specifica struttura con cui è stato costruito il corpus (per una sua illustrazione cfr. § 20.1 e sottoparagrafi) ed i valori con cui ogni tag è stato etichettato (per un riassunto sinottico cfr. § 21.1 e sottoparagrafi), e dall'altro (§ 21.2.2) le caratteristiche del linguaggio di interrogazione CQP, che, pur giocoforza limitate dall'interfaccia web, restano nondimeno amplissime.

I paragrafi seguenti vorrebbero essere una guida minimale al CQP online (la manualistica di riferimento, per cui cfr. il § 21.2.2 è infatti sempre riferita alla versione funzionante in locale e non sul web) specificamente centrata sulla struttura del *Corpus Taurinense* e pensata espressamente per i suoi utenti[6].

21.2.1 LA STRUTTURA DEL CT. Per il primo punto del § 21.2, è sufficiente ricordare che il testo del CT nel formato-CQP finale è costituito da una serie di righe in cui ad ogni token (definito `word`) sono associati 11 attributi posizionali (per la loro definizione cfr. § 19.1.2.2; per i loro valori possibili cfr. quanto riassunto nel § 21.1) direttamente interrogabili, ed accanto a questi, su righe separate, sono aperti e chiusi 10 attributi strutturali *XML-like*:

0	**1**	**2**	**3**	**3**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
<line 2>											
tu	tu	|pd.per.s.no|	|37,2,4;5,6,0,0|	P	n	Did	tu	tu	-	-	0
hai	avere	|v.a.f.ind.pr|	|211,2,0,6,0,0|	P	n	Did	hai	hai	-	-	0
fatta	fare	|v.m.nf.par.pt|	|123,0,5,6,0,0|	P	n	Did	fatta	fatta	-	-	0
mala	malo	|adj|	|26,0,5,6,8,0|	P	n	Did	mala	mala	-	-	0
venuta	venuta	|n.c|	|20,0,5,6,0,0|	P	n	Did	venuta	venuta	-	-	0
,	comma	|punct.nfi|	|71,0,0,0,0,0|	P	n	Did	,	,	-	-	0
e	e	|conj.co|	|50,0,0,0,0,0|	P	n	Did	e	e	-	-	0
se'	essere	|v.a.f.ind.pr|	|211,2,0,6,0,0|	P	n	Did	se'	se'	-	-	0
morta	morire	|v.m.nf.par.pt|	|123,0,5,6,0,0|	P	n	Did	morta	morta	-	-	0
con	con	|adp.pre|	|56,0,0,0,0,0|	P	n	Did	con	con	-	-	0
tutta	tutto	|pd.ind|	|32,0,5,6,0,0|	P	n	Did	tutta	tutta	-	-	0
tua	tuo	|pd.pos.s|	|33,2,5,6,0,0|	P	n	Did	tua	tua	-	-	0
gente	gente	|n.c|	|20,0,5,6,0,0|	P	n	Did	gente	gente	-	-	0
,	comma	|punct.nfi|	|71,0,0,0,0,0|	P	n	Did	,	,	-	-	0
</line>											

Tav. 257: Frammento in formato-CQP di testo normale.

Gli attributi posizionali nella Tav. 257 *supra* così come nelle 258-260 *infra*, che riassumono la struttura complessiva esibendo scampoli del testo-CT nella sua forma-CQP, sono: 1 `lemma`, 2 `pos`, 3 `kat`, 4 `typ`, 5 `corr`, 6 `genre`, 7 `msform`, 8 `philform`, 9 `mwlword`,

[6] È ben nota, infatti, la refrattarietà a tutto ciò che è latamente informatico o logicamente – "matematicamente!", direbbero con orrore, laddove io lo direi con sollucchero – formalizzato, caratteristica di molti ambienti umanistici, in cui pure si trovano non pochi potenziali utenti del corpus.

10 `mwlkat` ed 11 `mwlnum`; cfr. ad esempio le tavole seguenti (o quella al fondo del § 20.1.3); quanto agli attributi strutturali `author`, `title`, `chapter`, `par`, `s`, `line`, `page`, `type`, `genr` e `mwl` (che si possono liberamente visualizzare col comando CQP `show` o porre come restrizioni alle query col comando CQP `within`, ma non direttamente interrogare), cfr. per la loro definizione cfr. § 19.1.2.3, e per i loro valori possibili il § 21.1. Se il primo esempio è quello di una sequenza affatto normale (Tav. 257), il secondo (Tav. 258) è una sequenza con MW, il terzo (Tav. 259) un frammento con espunzioni ed integrazioni, ed il quarto (Tav. 260) uno con scioglimento di abbreviazioni e corsivi filologici.

0	1	2	3	3	5	6	7	8	9	10	11
<line 4>											
molte	molto	|pd.ind|	|32,0,5,7,0,0|	V	n	Lir	molte	molte	--	--	0
fiate	fiata	|n.c|	|20,0,5,7,0,0|	V	n	Lir	fiate	fiate	--	--	0
contro	contro	|adv.gn|	|45,0,0,0,8,0|	V	n	Lir	contro	contro	contro°a°	56,0,0,0,0,12	1
a	a	|adp.pre|	|56,0,0,0,0,0|	V	n	Lir	a	a	contro°a°	56,0,0,0,0,12	2
suo	suo	|pd.pos.s|	|33,3,4;5,6;7,0,0|	V	n	Lir	suo	suo	--	--	0
talento	talento	|n.c|	|20,0,4,6,0,0|	V	n	Lir	talento	talento	--	--	0
:	colon	|punct.nfi|	|71,0,0,0,0,0|	V	n	Lir	:	:	--	--	0
</line>											

Tav. 258: Frammento di testo in formato-CQP con MW.

0	1	2	3	3	5	6	7	8	9	10	11
<line 2>											
Beninchasa	beninchasa	|n.p|	|21,0,4,6,0,0|	P	y	Doc	Benichasa <Bent>	Beni[n]chasa	--	--	0
d'	di;da	|adp.pre|	|56,0,0,0,0,0|	P	n	Doc	d'	d'	--	--	0
Altomena	altomena	|zero|	|0,0,0,0,0,11|	P	n	Doc	Altomena	Altomena	--	--	0
notaio	notaio	|n.c|	|20,0,4,6,0,0|	P	n	Doc	notaio	notaio	--	--	0
</line>											
<line 3>											
che	che	|pd.rel|	|36,0,4;5,6;7,0,0|	P	n	Doc	che <che stava>	che	--	--	0
teneva	tenere/-si/	|v.m.f.ind.ipf|	|112,3,0,6,0,0|	P	n	Doc	teneva	teneva	--	--	0
botecha	bottega	|n.c|	|20,0,5,6,0,0|	P	n	Doc	botecha	botecha	--	--	0
soto	sotto	|adp.pre|	|56,0,0,0,0,0|	P	n	Doc	soto	soto	--	--	0
chasa	casa	|n.c|	|20,0,5,6,0,0|	P	n	Doc	chasa	chasa	--	--	0
di	di	|adp.pre|	|56,0,0,0,0,0|	P	n	Doc	di	di	--	--	0
Chavalchanti	cavalcanti	|n.p|	|21,0,4;5,6;7,0,0|	P	y	Doc	Chavachanti	Chava[l]chanti	--	--	0

Tav. 259: Frammento di testo in formato-CQP con espunzioni ed integrazioni.

0	1	2	3	3	5	6	7	8	9	10	11
<line 14>											
d'	di	|adp.pre|	|56,0,0,0,0,0|	P	n	Doc	d'	d'	--	--	0
oro	oro	|n.c|	|20,0,4,6,0,0|	P	y	Doc	o	o|ro|	--	--	0
:	colon	|punct.nfi|	|71,0,0,0,0,0|	P	n	Doc	:	:	--	--	0
ponemo	porre	|v.m.f.ind.pr|	|111,1,0,7,0,0|	P	y	Doc	po	po(nemo)	--	--	0
che	che	|conj.sb|	|51,0,0,0,0,0|	P	n	Doc	che	che	--	--	0
deono	dovere	|v.md.f.ind.pr|	|311,3,0,7,0,0|	P	y	Doc	deo	deo|no|	--	--	0
dare	dare/-si/	|v.m.nf.inf.pr|	|121,0,0,0,0,0|	P	n	Doc	dare	dare	--	--	0
innanzi	innanzi	|adv.gn|	|45,0,0,0,8,0|	P	n	Doc	innanzi <ne÷l>	innanzi	--	--	0
quatro	quattro	|num.car|	|64,0,4;5,0,0,0|	P	n	Doc	quatro	quatro	--	--	0
</line>											

Tav. 260: Frammento di testo in formato-CQP con scioglimento d'abbreviazioni e corsivi filologici.

A proposito del markup filologico, sarà forse bene ricapitolare sistematicamente il trattamento di parentesi tonde abbreviative, parentesi quadre integrative e corsivo filologico[7],

0	1	2	3	3	5	6	7	8	9	10	11
&[Di&]pravamento[8]											
Dipravamento	dipravamento	|n.p|	|21,0,4,6,0,0|	P	n	Did	[Di]pravamento	pravamento	--	--	0
&[Donati&]											
Donati	donati	|n.p|	|21,0,4;5,6;7,0,0|	P	n	Sto	[Donati]	--	--	--	0
ano&(verai&)											
anoverai	annoverare	|v.m.f.ind.pt|	|113,1,0,6,0,0|	P	n	Doc	ano(verai)	ano	--	--	0
c&(entinaio&)											
centinaio	centinaio	|n.c|	|20,0,4,6,0,0|	P	n	Doc	c(entinaio)	c	--	--	0
&|giungno&|											
giungno	giugno	|n.c|	|20,0,4,6,0,0|	P	n	Doc	|giungno|	--	--	--	0
percha&|mena&|											
perchamena	pergamena	|n.c|	|20,0,5,6,0,0|	P	y	Doc	percha|mena|	percha	--	--	0
passavano <per> <la> <via>											
passavano	passare	|v.m.f.ind.ipf|	|112,3,0,7,0,0|	P	n	Did	passavano <per la via>	passavano	--	--	0

Tav. 261: Il trattamento del markup filologico nella forma-CQP.

Tutti questi attributi, opportunamente codificati in formato binario e compressi, sono gli oggetti che, materialmente, maneggia il CQP, il motore di ricerca ed al contempo linguaggio di interrogazione del CWB.

21.2.2 IL LINGUAGGIO DI INTERROGAZIONE CQP: GENERALITÀ. Quanto al linguaggio in cui formulare le query, ossia il secondo punto di § 21.2, bisogna comunque avvertire che è disponibile presso il sito dell'IMS (nel CWB Users' Corner), e linkata dalla homepage del CT, una ampia documentazione, che spazia dal formato minimo della "quick reference" alla più ampia manualistica tecnica, come il fondamentale, anche se ormai non aggiornato, Christ et alii 1999[9] ed il più recente Evert 2005; tra questi materiali v'è anche un efficace, anche se un poco datato, prontuario di interrogazione del CT (Heid 2000). Ma, stante quanto si diceva preliminarmente nel § 21.2, non guasterà forse darne anche qui qualche elementare ragguaglio *sub specie CT*, tanto più se limitato alle funzionalità di base del linguaggio, e comunque alle sole utilizzabili nella versione web.

Innanzitutto bisogna infatti ribadire che il CQP, oltre che il motore di ricerca del Corpus WorkBench, è anche un vero e proprio linguaggio di interrogazione, basato su un sottoinsieme dello standard *POSIX*[10], in cui pertanto sono possibili espressioni regolari (Reg-

[7] In corsivo è data la forma-CT corrispondente.

[8] Premetto, in corsivo, per ogni categoria la forma-CT corrispondente.

[9] E tutta la trattazione seguente (che si limita solo alle caratteristiche fondamentali del linguaggio di interrogazione) si basa largamente, anche quando non esplicitamente detto, proprio su Christ et alii 1999. Evert 2005, in genere, descrive tecniche spesso utilissime ma troppo avanzate per l'utente medio del nostro corpus, e comunque sempre non disponibili via Web.

[10] «POSIX [...] or "Portable Operating System Interface" is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system, although the standard can apply to any operating system. Originally, the name stood for IEEE Std 1003.1-1988, which, as the name suggests, was released in 1988. The family of POSIX standards is formally designated as IEEE 1003 and the international standard name is ISO/IEC 9945. The standards emerged from a project

Exp)[11] con operatori booleani: qui ne vedremo giusto i principali operatori, le principali espressioni, ed alcune applicazioni alla specifica struttura del CT.

21.2.3 QUERY ELEMENTARI ATTRIBUTO="VALORE". In generale una query in CQP consiste al minimo nella affermazione di un particolare valore (*value*; cfr. § 19.1.4.1), normalmente formulato con una RegExp, per un attributo posizionale del corpus (cfr. § 19.1.2.2): in breve, un *match*.

```
attributo="valore"
```

21.2.4. TIPO DIRETTO E "STRING MATCHING". Tale valore, nel tipo più semplice e diretto di query, viene posto tra virgolette, doppie (`"valore"`, come si è sempre fatto in questo volume), od indifferentemente singole (`'valore'`); se non viene specificato l'attributo cui tale specifica viene riferita, si intende di default il token (`word`), pertanto la query 2068a è in tutto equivalente alla "completa" 2068b ed entrambe dànno risultati come 2068c:

[2068a] `"topo"` *query CQP*,
[2068b] `word="topo"` *query CQP*,
[2068c] `Poco stante , vidde entrare uno` **`topo`** `per la finestrella , che traeva a ÷ll' odore .` *Novellino*, lxxxij.1, p. 335[12].

Ed il medesimo tipo di semplice query dalla mera forma `attributo="valore"` funziona parzialmente[13] anche per il solo attributo `lemma`, mentre ogni altro attributo posizionale deve porsi tra parentesi quadre, come qualsiasi espressione booleana complessa (cfr. *infra* § 21.2.12).

---

that began near 1985. Formerly known as IEEE-IX, the term POSIX was suggested by Richard Stallman in response to an IEEE request for a memorable name» (WikipediaEN, *s.v.*)

[11] Le espressioni regolari sono uno degli strumenti più potenti a disposizione del linguista computazionale e del programmatore. Nei termini più generali: «A regular expression, often called a **pattern**, is an expression that describes a set of strings. They are usually used to give a concise description of a set, without having to list all elements. For example, the set containing the three strings "*Handel*", "*Händel*", and "*Haendel*" can be described by the pattern `H(ä|ae?)ndel` (or alternatively, it is said that the pattern *matches* each of the three strings).» (WikipediaEN, *s.v.*).

«The seeds of regular expressions were planted in the early 1940s by two neurophysiologists, Warren McCulloch and Walter Pitts, who developed models of how they believed the nervous system worked at the neuron level. Regular expressions became a reality several years later when mathematician Stephen Kleene [...] formally described these models in an algebra he called *regular sets*. He devised a simple notation to express these regular sets, and called them regular expressions» (Friedl 1997/2006 p. 85). Dopo essere state ampiamente studiate ed utilizzate in logica formale e matematica negli anni Cinquanta e Sessanta, alla fine degli anni Sessanta le espressioni regolari hanno fatto il loro ingresso nelle scienze informatiche, fino a diventarne uno degli strumenti più potenti (per la storia di questa fase, cfr. Friedl 1997/2006 pp. 85-91).

La bibliografia è, prevedibilmente, vastissima; ci limitiamo a segnalare l'essenziale: oltre al classico Friedl 1997/2006 ed alla sua versione "pratica" ed abbreviata Stubblebine 2003, cfr. anche Good 2004 e Goyvaerts 2006. Segnaliamo anche l'utilità del software RegexPal 0.1.4, un tester di espressioni regolari in JavaScript liberamente disponibile (licenza "Lesser GPL").

[12] Dato il particolare taglio di questa trattazione, nel presente capitolo riporteremo i risultati delle query nel formato-CQP restituito dall'interfaccia web del corpus online, anziché nel consueto formato-CT che abbiamo sempre usato nel rimanente del volume; per le medesime ragioni, anche il contesto, che normalmente diamo sempre pieno, sarà in questo capitolo ristretto alla finestra minima.

[13] O meglio non dà errore, ma non dà neppure tutti i risultati voluti, sicché in realtà è meglio evitare questa sintassi e scrivere anche questa espressione tra quadre: i suoi risultati, infatti, non sono ben prevedibili, e sembrano piuttosto corrispondere non ad un vero `[lemma="topo"]` ma piuttosto ad un `"topo.*"`, come se il valore fosse tacitamente riportato all'attributo di default, `word`.

[2069a] `lemma="topo"` *query CQP parzialmente erronea*;
[2069b] `genre="Did"` *query CQP erronea*;
[2069c] `[genre="Did"]` *query CQP*;
[2069d] `pos="intj"` *query CQP erronea*;
[2069e] `[pos=".*intj.*"]` *query CQP*;
[2069f] `« Oi cattivo ! » disse la femina .` *Novellino*, xxxviij.7, p. 216.

La query 2069a, infatti, produce, tra gli altri, il medesimo risultato 2068b delle query 2068ab, laddove le query omostrutturali 2069b e 2069d con altri attributi posizionali (rispettivamente *genre* e *pos*) non funzionano (il laconico commento del CQP è «no matches»); funzionano invece con l'introduzione delle quadre 2069c e2069e, che producono regolarmente i risultati attesi: cfr. tra gli ottanta match della 2069e l'esempio 2069f.

La medesima struttura della query può essere, naturalmente, usata con qualsiasi attributo posizionale (tutti gli attributi posizionali, infatti, sono "direttamente" interrogabili, e questo è il modo più elementare per farlo), come ad esempio con `philform`, `msform` o `mwlword`:

[2070a] `[philform="<.*"]` *query CQP*;
[2070b] `[msform="--"]` *query CQP*;
[2071a] `[mwlword=".*onde.*"]` (10)[14] *query CQP*;
[2071b] `che ne ÷' mie' forti guai m' affanna` **`là ond'`** `i' prendo ogni valore .` Cavalcanti, *Rime*, xxxij.4, bal. *Quando di morte*, v. 30, p. 537,
[2071c] `[mwlword=".*onde.*" & lemma="onde"]` (5) *query CQP*.

La query 2070a, ad esempio, trova tutte le parole cassate od espunte del testo (che non sono presenti a livello di `word`: cfr. § 20.2), mentre la 2070b tutte le parole integrate (cfr. § 20.1.3). La query 2071a, invece, trova tutte le multiword con *onde*; in realtà però trova propriamente "tutte le parole che fanno parte di una multiword con *onde*", quindi l'es. 2071b qui sopra viene di fatto preso due volte, sicché il totale di 10 risultati fornito dalla query non è indicativo; per fare sì, invece, che ogni MW faccia match una sola volta bisogna specificare qualche condizione aggiuntiva, come ad esempio in 2071c, dove si è fatto ricorso alla congiunzione (cfr. § 21.2.6 *infra*) per garantire che il match non contenga semplicemente un `mwlword` *onde* (che è espresso su tutti i costituenti della multiword) ma presenti lui stesso un `lemma` *onde* (condizione soddisfatta da uno solo dei costituenti della MW che cerchiamo[15]): ed i 5 risultati corrispondono questa volta alla realtà

In quasi tutti gli esempi di query precedenti, inoltre, abbiamo usato il metodo più semplice dello *string matching*, ossia abbiamo richiesto che il match del valore dell'attributo posizionale fosse una determinata stringa di caratteri. In pochi casi (query 2069e, 2070 e 2071a), però, abbiamo dovuto abbandonare questa semplice strategia introducendo per la prima volta un *pattern matching*, ossia un match effettuato su una struttura, nella fattispecie quella specificata da due *wildcharacters*; ed in un caso, addirittura abbiamo collegato con una congiunzione un *pattern matching* con uno *string matching* sulla medesima posizione cercata (query 2071c).

21.2.5 "PATTERN MATCHING" SEMPLICI ED ESPRESSIONI REGOLARI. Il valore di un attributo può essere infatti espresso, oltre che da un perfetto match di stringa col corpus, anche da espressioni regolari, che permettono un match di struttura, ossia un *pattern matching*.

[14] Dò tra tonde, quando per qualche ragione interessante, come già ho altrove fatto, il numero dei match.
[15] A meno che, naturalmente, la MW non contenga due volte la parola *onde*: ma ciò non avviene né in italiano antico né in moderno.

In genere, le espressioni regolari si compongono, oltre che dei caratteri che costituiscono la stringa da trovare, di una batteria di una dozzina (con modeste variazioni a seconda del linguaggio utilizzato, ad es. Unix di base, Perl, ecc., od appunto CQP) di *metacaratteri* o caratteri speciali.

21.2.5.1 "WILDCHARACTERS" QUANTIFICATI. Il primo gruppo di questi, che si suppone ormai universalmente noto (dato il loro uso diffuso nell'uso dei computer), è quello dei cosiddetti *wildcharacters*,

.	*dot*	qualsiasi singolo carattere (escluso *newline* e *zero*)
*****	*star*[16]	qualsiasi numero di ripetizioni incluso lo 0 (*zero*)
+	*plus*	qualsiasi numero di ripetizioni escluso lo 0 (*zero*)

Tav. 262: *wildcharacters* semplici.

normalmente combinati per essere appropriatamente quantificati

.*	*dot-star*	qualsiasi carattere presente od assente (il *punto* fa match con qualsiasi carattere, e la *stella* permette al punto di essere ripetuto per qualsiasi numero di volte, incluso lo zero)
.+	*dot-plus*	qualsiasi carattere per forza presente (il *punto* fa match con qualsiasi carattere, ed il *più* permette al punto di essere ripetuto per qualsiasi numero di volte, escluso lo zero)

Tav. 263: *wildcharacters* combinati.

Così la query 2068a potrebbe anche essere espressa da

[2072a] `"t.po"` (3) *query CQP*,
[2072b] `"to.o"` *query CQP*,

dato che non esistono nel corpus *tipi* e *tori* (o parole di simile struttura); ma entrambe le query 2072ab

[2073a] `"top."` (9) *query CQP*,
[2073b] `".opo"` (60) *query CQP*,
[2073c] `Et ciò sappiate , che de ÷l decto facto non è da maravigliare , perciò che si truova che in certe terre , dove l' uomo è morso da ÷l leopardo , i` **`topi`** `incontanente in quella parte abondano , e tucti gli pisciano adosso , sì cche quasi vi fanno un lagho ; per la qual soççura sì ne seguita a questo huomo la morte .`
Cronica fiorentina, mlv, p. 85,
[2073d] `Uno re fu ne ÷lle parti di Egitto , lo quale avea un suo figliuolo primogenito , lo quale dovea portare la corona de ÷l reame` **`dopo`** `lui .`
Novellino, iiij.1, p. 134,

prendono un numero di risultati più ampi dei semplici 3 delle query equivalenti 2068ab e 2072ab, rispettivamente 9 e 60, perché l'una fa match anche con i plurali di *topo* (es.

[16] Tipograficamente un asterisco, è in realtà derivato dall'operatore noto in logica matematica come *Kleene star* o *Kleene closure*, che descrive una funzione ad una sola variabile applicata ad un insieme di stringhe o di caratteri; è così chiamato dal nome del suo πρῶτος εὑρετής Stephen Cole Kleene (1909-1994, pron. ['kleɪni]), il grande matematico e logico americano fondatore della teoria della recursione e padre, *ipso facto*, del concetto medesimo di *espressione regolare* (cfr. *supra* n. 11 § 21.2.2). Nel gergo degli utenti di CQP è, pertanto, correttamente, chiamato *star* anziché *asterisco*.

2073c), e l'altra perché fa match anche[17] con 46 *dopo* (es. 2073d), 9 *Dopo*[18], 2 *uopo* e 3, appunto, *topo*.

Se invece si volesse che invece di 'qualsiasi singolo carattere' il *punto* stesse ricorsivamente per 'qualsiasi numero di caratteri', dovremmo quantificare tale ricursione usando il *più* o la *stella* (cfr. *supra*). Così se 2068a e2072a avevano solo 3 risultati, 2074ab ne dànno ben 350, prendendo indifferentemente *troppo* (con `.=rop`), *tempo* (con `.=em`, cfr. es. 2074c), *tenpo* (con `.=en`), ecc.:

[2074a] `"t.*po"` (331) *query CQP*;
[2074b] `"t.+po"` (331) *query CQP*;
[2074c] `Fu ÷lli contato come nodrido era stato con savi e con uomini di` **`tempo`** `, lungo da ogni fanciullezza .` *Novellino*, iiij.5, p. 135.
[2074d] `De ÷l mese di giungno , Guilglelmino , vescovo d' Arezzo , chon Ubertini e Pazzi di Valdarno e con Bonconte filgluolo de ÷l conte Guido di Montefeltro , chon Uberti e Lanberti ed altri sbanditi di Firenze , di notte` **`tempora`** `, entraro de ÷l mese di giungno inn Arezzo , e cacciaro fuori tutti i Guelfi .` *Cronica fiorentina*, mcclxxxvij, p. 131.

Inaspettatamente, però, nelle query 2074ab vengono prese anche forme come *tenpora* (cfr. es. 2074d), *tem|nporale|i* e *trassportate*, che forse non corrispondono precisamente a quello che volevamo (probabilmente 'qualsiasi parola che inizia con *t-* e finisce con *-po*'): perché? Perché la portata delle espressioni con quantificazione ricorsiva è potenzialmente infinita per cui può facilmente sfuggire di mano (di fatto l'espressione che abbiamo usato in realtà valeva 'qualsiasi parola che inizia con *t-* e contenga *-po-* nella stringa di caratteri che seguono'), e va usata pertanto con molta cautela. Se l'obiettivo era quello che supponevamo qui di sopra, meglio sarebbe stato ricorrere ad una RegExp che specificasse espressamente le condizioni di inizio e fine della stringa, ma purtroppo il linguaggio CQP è privo dei due matacaratteri appositi,

^	*caret*	metacarattere per l'inizio di una stringa
$	*dollar*	metacarattere per la fine di una stringa

Tav. 264: Metacaratteri assenti in CQP.

Si noti inoltre fin da sùbito la differenza tra i due quantificatori *stella* e *più*. Nel caso delle query qui sopra (2074a e 2074b) la scelta dell'uno o dell'altro non faceva differenza alcuna, perché non esistono nell'italiano antico e nel moderno parole del tipo *tpo* (ossia con zero al posto del punto, come la query 2074a con la stella consentirebbe, a differenza della 2074b con il più, che la escluderebbe). Ma la differenza emerge immediatamente se quantifichiamo una query come la seguente:

[2075a] `"topo.*"` (3) *query CQP*,
[2075b] `"topo.+"` (0) *query CQP*:

2075a dà gli stessi 3 risultati (*topo*) di 2068ab e 2072ab (ammettendo la possibilità che *topo* non sia seguito da nessun carattere), mentre 2075b non dà risultati (imponendo che *topo* debba obbligatoriamente essere seguito da un solo altro carattere: il che non avviene).

---

[17] Come facilmente controllabile con la query 2073e, per la cui sintassi cfr. *infra*.

[2073e] `[word=".opo" & lemma != "dopo"]` *query CQP*.

[18] Il CQP, come i linguaggi ad ispirazione Unix in genere, è *case sensitive*; ma si può rendere una query *case insensitive* aggiungendo all'espressione del valore desiderato il parametro (flag) `%c`: cfr. *infra* § 21.2.8.3.

Accanto a questi 3 metacaratteri “classici” (comuni a quasi tutti i linguaggi che usano RegExp) il CQP ha anche un’altra espressione, semanticamente analoga al punto ed ugualmente quantificabile,

[]	*match any*	un token avente qualsiasi stringa di caratteri,

Tav. 265: Un metacarattere caratteristico di CQP, il *match any*.

ma con diversa sintassi, in quanto usualmente non serve in posizione iniziale di query, dove anzi usato da solo catturerebbe tutto il corpus

[2076] `[]` *query CQP*,

ma tipicamente si adopera nelle ricerche per cooccorrenze per ammettere la possibilità di qualsiasi parola tra le due specificate. Così la query 2077a[19] pesca i 3 esempi di cooccorrenze di *per* e *tempo* a contatto (cfr. es. 2077e), ma 2077b cattura il gruppo, più numeroso e forte di 13 attestazioni, di cooccorrenze di *per* e *tempo* con una parola qualsiasi (*alcun*, *neun*, *alquanto*, *lo*, *quanto*, *anticho*) in mezzo (cfr. es. 2077f),

[2077a] `"per" "tempo"` *query CQP*,
[2077b] `"per" [] "tempo"` *query CQP*,
[2077c] `"per" []* "tempo"` *query CQP*,
[2077d] `"per" []+ "tempo"` *query CQP*,
[2077e] `Allora trovo ÷e una molto bella canzonetta , e la mattina` **`per tempo`** `salìo in su ÷e lo pergamo .` *Novellino*, lxiiij.21, p. 274,
[2077f] `Questo libro tratta d' alquanti fiori di parlare , di belle cortesie e di be' risposi e di belle valentie e doni , secondo che` **`per lo tempo`** `passato hanno fatto molti valenti uomini .` *Novellino*, iiij.5, p. 135,
[2077g] `Parti ÷si un dì di subito , et ando ÷nne` **`per una gran foresta , e trovava orsi e leoni et assai fiere pessime : tutte le squarciava et uccidea con la sua forza ; e non trovò niuna bestia sì forte , che da lui si difendesse . E' stette in questa foresta gran tempo`** `; poi tornò a casa a ÷lla moglie co ÷' panni tutti squarciati , con pelli di leoni a dosso .` *Novellino*, lxx.5, p. 289.

La ulteriore quantificazione del *match any* con *stella* e *più* è affatto possibile, ma forse, nuovamente, di potenza eccessiva: 2077c e 2077d, una con 120 match e l’altra con 117 (in quanto non prende i 3 individuati da 2077a poiché non ammette la possibilità che fra *per* e *tempo* non ci sia alcuna parola), prendono anche moltissimi esempi linguisticamente di utilità nulla come 2077g: la ricorsività galoppa. Per fortuna, come vedremo tra poco (cfr. §§ 21.2.8.1-2) esistono anche altri, più accurati, modi di quantificare i *wildcharacters* che nelle iperpermissive 2077cd, come ad es. quelli di 2086d, 2088g e 2088h, *infra*.

21.2.5.2 IL METACARATTERE LETTERALE. Un ultimo particolare carattere speciale è quello che serve per usare un metacarattere come carattere normale:

****	*backslash*	forza il valore letterale al carattere che segue

Tav. 265: Un metacarattere di CQP, il *backslash*.

Non è infatti strano, anche nel CT, voler cercare parole con un apostrofo o segni di interpunzione altrimenti adibiti a metacaratteri (come punti esclamativi od interrogativi, ecc.);

[19] Che implicitamente usa la concatenazione, per cui cfr. *infra* §§ 21.2.6 e 21.2.12.

così una query come 2078a pescherà correttamente[20] tutte le 35 forme di *ove* apostrofate presenti nel corpus (cfr. es. 2078c); e del pari la query 2078b farà match con tutti i 107 punti esclamativi del CT (cfr. es. 2078d).

[2078a] `"ov\'"` *query CQP*,
[2078b] `"\!"` *query CQP*,
[2078c] `« Femmina , non ho di che ti sovenire d' altro , ma fa' così : mena ÷mi a ÷lla carcere` **`ov'`** `è 'l tuo figliuolo » .` *Novellino*, xv.2, p. 161,
[2078d] `Siano morti i traditori !` *Cronica fiorentina*, mcclxxxxvj, p. 145.

Si badi, inoltre che la query 2079a pescherà regolarmente tutti i caratteri espressi dal metacarattere *punto* (cfr. § 21.2.5.1), cioè tutti i token costituiti da un solo, qualsiasi, carattere (tutti i segni di interpunzione, *e*, *a*, ecc.)[21], ma per cercare effettivamente solo i token costituiti unicamente dal punto (un sottoinsieme ben preciso, ossia, dei risultati della query precedente, che prende indifferentemente gli ess. 2079b e 2079d) bisognerà adottare la query 2079c che usa il metacarattere letterale (tra i cui risultati sarà l'es. 2079d ma non più l'es. 2079b):

[2079a] `"."` *query CQP*,
[2079b] **`,`** `cominciò in cotal modo` **`a`** `parlare` **`:`** Bono, *Libro Vizi*, v.1, p. 11,
[2079c] `"\."` *query CQP*,
[2079d] `[...] t' avea dato` **`.`** `Ond' è tempo [...]` Bono, *Libro Vizi*, v.1, p. 11.

21.2.6 OPERATORI LOGICI DI BASE. A questo punto, dopo averne visto implicitamente (e talora inconsapevolmente) in azione alcuni, possiamo meglio esplicitare gli operatori che costituiscono lo scheletro del linguaggio CQP. Accanto ai *wildcharacters* che abbiamo già definito, infatti, i metacaratteri di base usati nelle RegExp di CQP comprendono i fondamentali operatori logici: congiunzione (*e*), disgiunzione (*o*), identità (*uguale*) e negazione (*non*), ossia:

&	*and*	congiunzione (*e*)
|	*or*	disgiunzione (*o*)
=	*value statement*	identità (*uguale*)
!=	*value negation*	negazione di attributo (*non vale "x"*)
!	*not*	negazione (*non*)
<*space*>	*concatenation*	concatenazione tra più match od espressioni
(...) *x* (...)	*round brackets*	specifica il dominio di un operatore o di un parametro (*grouping*)
[...]	*square brackets*	delimita una RegExp od una espressione booleana

Tav. 267: Gli operatori logici di base in CQP.

Come precisato in Christ et alii 1999, le proprietà di precedenza dei principali operatori logici sono date dalla lista " `=, !=, !, &, |` ", «i.e. if operator x is listed before operator y, operator x has precedence over y» (Christ et alii 1999 cit.).

Di questi, l'*and* (congiunzione, cioè intersezione di due insiemi) è solo usato in espressioni booleane su match, e lo esamineremo pertanto separatamente, prima introducendo accanto alle query su una singola posizione del corpus con *pattern matching* di match

[20] In entrambi i casi, in realtà, il mancato uso del "metacarattere letterale" non dà necessariamente luogo ai consueti avvertimenti di «Syntax error» per ragioni non ben chiarite; ma è comunque più sicuro abituarsi all'uso corretto.
[21] Insieme ampio, certo, ma *tutti* i token del corpus invece vengono presi dalla query 2076, cfr. sopra.

semplice (§ 21.2.5) quelle con match complesso, composto di due *patterns* congiunti (§ 21.2.7), e poi contrastandolo (§ 21.2.10), logicamente e praticamente, con l'*or* (disgiunzione, cioè unione di due insiemi).

Il *value statement*, così come la concatenazione per mera giustapposizione, li abbiamo più o meno tacitamente già usati, e non meritano ormai ulteriore attenzione; l'*or* e la negazione, invece, hanno una latitudine d'uso diversa ma comunque assai vasta, non senza alcune idiosincrasie, sicché li esamineremo partitamente in séguito (risp. §§ 21.2.10 e 21.2.11).

Pochi cenni introduttivi, invece, sull'uso delle parentesi, le cui minute applicazioni figureranno dove del caso nei paragrafi seguenti. Come consueto, le parentesi tonde sono usate per specificare il preciso dominio di applicazione di un operatore (a qualsiasi livello) od a particolari espressioni complesse; le quadre, invece, delimitano tipicamente delle espressioni booleane semplici (al minimo: una coppia attributo-valore, come abbiamo già visto ad es. nel gruppo di query 2069) o complesse (come presto vedremo) e sono applicabili ad una singola RegExp solo nel caso speciale delle "quadre di alternanza" (cfr. § 21.2.9).

21.2.7 "PATTERN MATCHING" CONGIUNTI SU POSIZIONE SINGOLA. Presentando, inizialmente, il tipo più elementare di query possibile in CQP, ossia quello su una singola posizione del corpus (specificata in una data espressione booleana), abbiamo esaminato prima il tipo base costituito da un match di stringa semplice (§ 21.2.4), e poi quello appena più booleanamente sofisticato costituito da un match di *pattern* (§ 21.2.5). Come abbiamo già anticipato nell'es. 2071c, però, l'espressione booleana che specifica il match con la posizione del corpus, oltre che modificabile al suo interno (ad esempio con gli operatori presentati nei §§ 21.2.5.1-2), può anche essere coordinabile al suo esterno con altre espressioni booleane che specificano diverse condizioni per la stessa posizione (per le loro concatenazioni esterne su più posizioni cfr. oltre § 21.2.12). Quando ciò avviene, il caso più semplice e frequente è quello con congiunzione (cfr. § 21.2.6), ed è questo l'oggetto del presente paragrafo.

Data la grande ricchezza nel CT di metadata espressi da attributi posizionali, quindi direttamente interrogabili dal sistema, è evidente l'utilità di potere specificare per una data posizione del corpus un insieme congiunto[22] di più attributi.

Si può andare dalla combinazione più semplice di due soli attributi, come nelle query 2080bd dove la specificazione della `pos` accanto al `word` consente di dividere nei vari sottoinsiemi nominali (query 2080a, cfr. es. 2080e), aggettivali (query 2080c, cfr. es. 2080f) ed avverbiali (query 2080d, cfr. es. 2080g) l'insieme di tutti i `word` *piano* (da questo punto di vista la query semplice 2080a corrisponde infatti alla query disgiunta 2080h, cfr. § 21.2.10),

[2080a] `[word="piano"]` (19) *query CQP*,
[2080b] `[word="piano" & pos=".*n.c.*"]` (9) *query CQP*,
[2080c] `[word="piano" & pos=".*adj.*"]` (6) *query CQP*,
[2080d] `[word="piano" & pos=".*adv.*"]` (4) *query CQP*,
[2080e] `ne ÷l` **`piano`** `di Morici a ÷l castello de ÷l Bosso`
Cronica fiorentina, mcclxx, p. 121,
[2080f] `Ed era` **`piano`** `in costumi , grazioso in donare .`
Fiore Filosafi, xxij.1, p. 171,
[2080g] `non andare troppo piano` *Fiore Filosafi*, xx.15, p. 156,

[22] La differenza concettuale tra un insieme *congiunto* ed uno *disgiunto* è ampiamente spiegata nel § 21.2.10, dove anche la differenza pratica tra l'uso della congiunzione e della disgiunzione è altrettanto diffusamente esemplificata.

[2080h] `[word="piano" & (pos=".*adv.*" | pos=".*n.c.*" | pos=".*adj.*")]` *query CQP*,

ad esempi ancora più complessi che possono comporre molte asserzioni di attributo sulla medesima posizione: ad esempio, se la query 2081a mi troverà tutti i lemmi del verbo *dare*, la query 2081b, con congiunto un valore di `pos`, me li limiterà all'indicativo, la query 2081c, a sua volta, con congiunto un valore di `kat`, me li limiterà al solo singolare, la query 2081d, poi, con congiunto un valore di `genre`, me li limiterà ai soli testi di genere lirico, e quella 2081e, infine, con congiunto un valore di `typ`, me li limiterà ulteriormente ai soli esempi in verso, passando da 890 esempi ai soli 14 (tra cui l'es. 2081f) che, ipoteticamente, cercavo:

[2081a] `[lemma="dare.*"]`[23] (890) *query CQP*,
[2081b] `[lemma="dare.*" & pos=".*ind.*"]` (303) *query CQP*,
[2081c] `[lemma="dare.*" & pos=".*ind.*" & kat=".*6.*"]` (255) *query CQP*,
[2081d] `[lemma="dare.*" & pos=".*ind.*" & kat=".*6.*" & genre="Lir"]` (19) *query CQP*,
[2081e] `[lemma="dare.*" & pos=".*ind.*" & kat=".*6.*" & genre="Lir" & typ="V"]` (14) *query CQP*;
[2081f] `membrar mi` **`dà`** `orrore`
Dante, *Vita nuova*, iij.11, son. *A ciascun' alma presa*, v. 8, p. 15.

Tutte le combinazioni, inoltre, di attributi posizionali sono teoricamente possibili, anche le più peregrine, come ad es. la query 2082a, che vuole trovare tutti gli emendamenti attuati a forme di *fare* all'indicativo singolare, che consistono poi nell'unico es. 2082b,

[2082a] `[lemma="fare.*" & pos=".*ind.*" & kat=".*6.*" & corr="y"]` (1) *query CQP*;
[2082b] `ch' ogn' altro spiritel face` (msform `fa` / philform `fa[ce]`) `gentile`
Cavalcanti, *Rime*, xxviij.1, son. *Pegli occhi fere*, v. 4, p. 530.

o la catena di esempi seguente,

[2083a] `[lemma="a" & mwlword=".*a.*"]` (577) *query CQP*;
[2083b] `[lemma="a" & mwlword=".*a.*" & mwlkat=".*56.*"]` (293) *query CQP*;
[2083c] `[lemma="a" & mwlword=".*a.*" & mwlkat=".*56.*" & mwlnum=".*2.*"]` (234) *query CQP*;
[2083d] **`a guisa`** `d' albore piantato`
Rinuccino, *Rime*, iiij.1, son. *Amore à nascimento*, v. 2, p. 45,
[2083e] `vogliendo porre la sua sedia` **`allato a`** `la mia` Bono, *Libro vizi*, vj.9, p. 16,
[2083f] `[lemma="a" & mwlword=".*a.*" & mwlkat=".*56.*" & mwlnum=".*2.*" & genre="Doc"]` (66) *query CQP*;
[2083g] `andare` **`dinanzi a`** `÷l vescovo` *Capitoli S. Gilio*, ij.26, p. 51,

dove la query 2083a (come già 2071c) trova tutte le multiword con *a*, mentre 2083b le limita quelle di natura preposizionale; la query 2083c poi impone la restrizione alle MW preposizionali con la *a* in seconda posizione, ammettendo solo esempi tipo 2083e ad esclusione di quelli come 2083d ammessi dalla query 2083b precedente; posso poi, per verificare collocazioni specialistiche, restringere ulteriormente i miei risultati, con la query 2083, alle sole attestazioni documentarie ottenendo i soli 66 esempi come 2083g.

La flessibilità di questi match complessi è ulteriormente aumentata dalla possibilità di esprimere anche insiemi disgiunti (con il connettivo *or*) e soprattutto condizioni negative (con il connettivo negativo `!`): ma l'uso di questi due connettivi necessita d'alcune avver-

[23] Le ragioni per usare l'espressione `"dare.*"` anziché solo `"dare"` saranno spiegate nel § 21.2.15.

tenze, sicché preferiamo rimandarlo a più avanti (rispettivamente § 21.2.10 e 21.2.11), ritornando ora, prima di affrontare altre query composite, ad alcune ulteriori precisazioni sulla parametrizzazione (§ 21.2.8 e sottoparagrafi) delle espressioni booleane che ancora dovevamo introdurre.

21.2.8 PARAMETRI: GENERALITÀ. Abbiamo già constatato che il match (od i metacaratteri che contiene) può essere ulteriormente parametrizzato in vari modi: già la quantificazione stessa effettuata sul punto "`.`" o sul *match any* "`[]`" da *stella* e *più*, che abbiamo già presentato nel § 21.2.4.3, si può ben considerare una parametrizzazione di questo tipo. E modificazioni analoghe si possono applicare anche a livelli diversi, a seconda degli operatori usati.

Per semplificare, considereremo tutti questi diversi "modificatori" come "parametri".

21.2.8.1 OPERATORE DI FACOLTATIVITÀ. Il dominio di applicazione (lo *scoop*, si direbbe in logica) di *stella* e *più*, si diceva, è il (meta)carattere precedente; ed il medesimo dominio ha anche un altro importante metacarattere,

`?`	"question"	rende facoltativo il carattere precedente (operatore di facoltatività),

Tav. 268: L'operatore di facoltatività in CQP.

per la cui definizione il CQP si discosta dalla maggior parte dei linguaggi che fanno uso di RegExp, perché in questi il `?` è usualmente un quantificatore come il *più* o la *stella* (e vale 'qualsiasi singolo carattere incluso lo *zero*'), e non un operatore di facoltatività.

Comechessia, il suo dominio di applicazione normale risulta chiaro dagli esempi seguenti; ipotizzando di volere indagare in italiano antico la grafia della nasale palatale in una famiglia di parole adatta per frequenza e fonetica, potremmo infatti ricorrere a query del tipo di 2084abc:

[2084a] `"sin?gn.*"` (175) *query CQP*,
[2084b] `"sign.*"` (150) *query CQP*,
[2084c] `"singn.*"` (25) *query CQP*,
[2084d] `In questo tempo l' ordine de ÷' si`**`gn`**`ori Tempieri , che sono decti cavalieri de ÷l Tempio , si conminciò .`
Cronica fiorentina, mcxviiij, p. 96,
[2084e] `Questo Otto fu ÷e poi dispossto de ÷lla si`**`ngn`**`oria , perké cadde in briga co ÷lla Chiesa e non observava la fedaltade .`
Cronica fiorentina, mclxxxxviiij, p. 114.

Ora, la query 2084b ci permette di cogliere i 150 casi di grafie <gn> alla moderna e la 2084c i 25 casi di grafie <ngn> all'antica; ma la 2084a, impostando la facoltatività sulla prima *n*, ci permette di cogliere entrambe le grafie in una volta sola (i suoi match sono infatti 175).

Il dominio di applicazione del `?` può essere ampliato a piacere da quello tipico (carattere precedente) ad una più articolata espressione regolare sempre interna al match, a tutto il match medesimo, od addirittura ad una espressione booleana.

La prima di queste strategie deve fare ricorso, ovviamente, alle parentesi tonde:

[2085a] `"fa(re)?"` *query CQP*,
[2085b] `"far?e?"` *query CQP*,
[2085c] `« Più mi nuoce tuo nome che non` **`fa`** `la tua prodezza »`
Novellino, xxxxv.4, p. 228,

[2085d] « Di ÷mmi , Balaam : che è ciò , che li miei nemici sono assai meno poderosi di me , e io non posso **fare** loro nullo danno ? » *Novellino*, xxxvj.2, p. 210,

[2085e] Allora suo marito le 'mpromise , de ÷l primo guadagno ch' e' prendesse , di **far** ÷le una bella cotta . *Novellino*, xxv.5, p. 188.

Si noti inoltre che la query 2085a, in quanto la facoltatività è applicata a tutto il blocco tra parentesi preso come un unico oggetto, coglie solo le forme *fa* (es. 2085c) e *fare* (es. 2085d) ad esclusione di *far*, che invece è incluso (es. 2085d) nei risultati della 2085b (915 match), dove la facoltatività è applica singolarmente ad ogni carattere del gruppo *-re*.

La seconda strategia, altrettanto ovviamente, deve fare ricorso alle virgolette, che sono l'usuale delimitatore di match. Si consideri, infatti, una query su cooccorrenze come la 2077: volendo, analogamente, valutare il valore collocazionale dell'articolo nell'espressione *per X tempo* si potrebbe così procedere:

[2086a] "per" "tempo" *query CQP*,

[2086b] "per" "lo" "tempo" *query CQP*,

[2086c] "per" "lo"? "tempo" *query CQP*,

[2086d] "per" []? "tempo" *query CQP*,

[2086e] Riscaldato d' ira , la mattina **per tempo** si levò e mise ÷si sotto le pelli una spada rugginosa e venne in capo de ÷l ponte ; [...] . *Novellino*, lxxxxvj.15, p. 343,

[2086f] E , chi avrà cuore nobile et intelligenzia sottile , sì li potrà simigliare **per lo tempo** che verrà per innanzi et argomentare e dire e raccontare (in quelle parti dove avranno luogo) , a prode et a piacere di coloro che non sanno e disiderano di sapere . *Novellino*, 0.2, p. 118.

Laddove la query 2086a prende 3 esempi (cfr. 2086e) e la 2086b 6 (cfr. 2086f), la 2086c, in cui la facoltatività è espressa su tutto il secondo match, ne cattura l'intersezione, cioè 9 esempi; si noti peraltro che una query come 2086d in cui la facoltatività è espressa sul *match any* **non** è equivalente alla 2077c con la *stella*: quella catturava (inutilmente!) anche tutte le sequenze di più parole tra le due impostate nel primo e nel terzo match, laddove questa solo le 15 con una parola (l'operatore di facoltatività non consente la proliferazione ricorsiva della *stella*!), e risponde probabilmente meglio alle intenzioni linguistiche che l'avevano dettata.

La terza strategia, infine, del pari prevedibilmente, farà ricorso alle parentesi quadre, che sono il normale delimitatore di espressione booleana. Consideriamo una query su cooccorrenze, ancora una volta mirata a valutare il valore collocazionale della presenza o meno dell'articolo in una coppia V+N:

[2087a] [lemma="dare"] [pos=".*n.c.*"] (42) *query CQP*,

[2087b] [lemma="dare"] [pos=".*art.*"] [pos=".*n.c.*"] (15) *query CQP*,

[2087c] [lemma="dare"] [pos=".*art.*"]? [pos=".*n.c.*"] (57) *query CQP*,

[2087d] Quando furo a ÷lla porta , li compagni de ÷ll' altro non li **diedero briga** , ché no 'l conobbero : [...] . *Novellino*, lxxxxviiij.8, p. 349,

[2087e] E , non accorgendo ÷si de ÷lla beffa , sì lli **diè un danaio** e tolse una medaglia , et ando ÷nne consolato *Novellino*, lxxxxvj.20, p. 344.

Come nel caso precedente, se la query 2087a prende 42 esempi (cfr. es. 2087d) e la 2087b 15 (cfr. es. 2087e), la 2087c, in cui la facoltatività è espressa su tutta la seconda espressione, ne cattura l'intersezione, cioè 57 esempi.

21.2.8.2 MOLTIPLICATORE. Un altro parametro semanticamente analogo ai quantificatori è il cosiddetto *moltiplicatore* (*multiplier*) od *operatore di intervallo* (*interval operator*)

{}	"braces"	indica la quantità di ripetizioni del pattern alla sua sinistra

Tav. 269a: L'operatore di intervallo o moltiplicatore in CQP.

che, a differenza del *punto* e della *stella*, precisa in modo accurato e definito la quantità delle ricursioni della espressione quantificata.

{*n*}	***n*** indica esattamente il numero di ricursioni della struttura quantificata
{*n*,}	indica ***n*** *o più* ripetizioni della struttura quantificata
{*n*,*m*}	indica a*l minimo* ***n*** *ed al massimo* ***m*** ripetizioni della struttura quantificata

Tav. 269b: Sintassi dell'operatore di intervallo.

Si presti, tra l'altro, attenzione che nella sua formulazione binomiale

{*0*,*1*}	equivale a	**?**	assenza o 1 occorrenza del quantificato
{*0*,}	equivale a	*****	assenza o illimitate ripetizioni del quantificato
{*1*,}	equivale a	**+**	almeno 1 occorrenza od illimitate ripetizioni del quantificato.

Tav. 269c: Equivalenze tra l'operatore di intervallo e gli altri quantificatori.

Pertanto le query che avevamo già introdotto come 2086d, 2077c e 2077d, e che ripropongo qui come 2088abc, sono rispettivamente riformulabili come 2088def,

[2088a] `"per" []? "tempo"` *query CQP* = 2086d,
[2088b] `"per" []* "tempo"` *query CQP* = 2077c,
[2088c] `"per" []+ "tempo"` *query CQP* = 2077d,
[2088d] `"per" []{0,1} "tempo"` *query CQP*,
[2088e] `"per" []{0,} "tempo"` *query CQP*,
[2088f] `"per" []{1,} "tempo"` *query CQP*,
[2088g] `"per" []{0,3} "tempo"` *query CQP*,
[2088h] `"per" []{1,3} "tempo"` *query CQP*.

Non solo, dato che avevamo (al fondo del § 21.2.5.1) rilevato l'eccessiva potenza della ricorsività di 2077c=2088b e 2077d=2088c, potremmo evitare l'inconveniente limitandone la portata, ad esempio, a 3 sole ripetizioni, come in 2088g (che ammette l'assenza del quantificato) e 2088h (che non ammette l'assenza del quantificato).

L'applicabilità del moltiplicatore è amplissima, potendosi esercitare praticamente su qualsiasi tipo di dominio possibile in CQP: su carattere (`n`: query 2089a); su metacarattere (*punto*: query 2089c); su RegExp interna tra tonde (`(nd-`*punto*`)`: query 2089e); su alternanza interna tra quadre (`[Tt]`: query 2089g); su un match tra (doppi) apici (`".e"`: query 2089i); su un match tra quadre (`[pos="..."]`: query 2089k); su concatenazione di espressioni tra quadre incassate in tonde (`([...] [...])`: query 2089m).

In generale, tuttavia, molte delle possibilità sopra evidenziate possono forse non parere linguisticamente granché utili; comunque le query sopra presentate (sempre con espressione diretta del numero di ripetizioni richieste) possono servire rispettivamente per trovare: tutte le parole con *n* doppia nel corpus (cfr. es. 2089b); tutte le parole (167) con due caratteri tra *da-* ed *-o* (cfr. es. 2089d); forme (49) gerundiali con reduplicazione (cfr. es. 2089f); forme (38) con raddoppiamento fonosintattico di *t* (cfr. es. 2089h); assimilazioni (314) nel-

le serie monosillabe (cfr. es. 2089j); verbi (51) seriali, causativi, ecc.[24] (cfr. es. 2089l); parallelismi, dittologie od endiadi (cfr. es. 2089n):

[2089a] `".*n{2}.*"` *query CQP*,
[2089b] « Che cos' a tirannìa è bellore di **donna** ! » *Novellino*, xiij.5, p. 159,
[2089c] `"da.{2}o"` *query CQP*,
[2089d] `[...] ; e fece grande grandissimo` **danno** . *Cronica fiorentina*, mcclxxxxvj, p. 146,
[2089e] `".*(nd.){2}.*"` *query CQP*,
[2089f] `La volpe` **andando** `per un bosco sì trovò un mulo : [...] .` *Novellino*, xiij.5, p. 159,
[2089g] `"[Tt]{2}.*"` *query CQP*,
[2089h] `Prese ÷la e cominciò a` **ttirare** . *Novellino*, xviij.15, p. 169,
[2089i] `".e"{2}` *query CQP*,
[2089j] `Il signor` **se ne** `fece gabbo .` *Novellino*, lxxviiij.3., p. 308,
[2089k] `[pos=".*v.m.*"] {3}` *query CQP*,
[2089l] `Domenedio , veggendo che non li ÷le` **potea fare dire** `, increbbe ÷li di lui .` *Novellino*, lxxv.17, p. 300,
[2089m] `([kat=".*50.*"] [pos=".*v.m.*"]) {2}` *query CQP*,
[2089n] `[...] ;` **e prese e vinse** `Guanto , Bruggia e Lilla e tutta la contea di Fiandra .` *Cronica fiorentina*, mcclxxxxvij, p. 150.

Tipicamente, si noti tra l'altro, espressioni come

[2090a] `[word=".{15}"]` (63) *query CQP*,
[2090b] `[word="[A-z]{15}"]` (52) *query CQP*,
[2090c] `[word="[0-9]{15}"]` (0) *query CQP*,

possono essere usate per indicare qualsiasi sequenza obbligatoriamente di *tot* caratteri, ma non sono equivalenti, poiché in realtà solo la prima comprende davvero *qualsiasi* carattere nella sequenza, la seconda ne esclude numeri e simboli (cfr. la differenza nel numero dei risultati, dovuta principalmente all'aver escluso o meno le parole con l'apostrofo), la terza lettere e simboli (ed infatti non trova match nel CT con le restrizioni imposte).

Altro discorso è la applicazione del moltiplicatore al *match any* `[]`, che è particolarmente importante, vuoi perché, dato che il *match any* ricorre perlopiù in query con strutture multiple concatenate, qui assume un valore speciale, quello di "operatore di intervallo", vuoi perché linguisticamente è molto frequente in tutte le query su cooccorrenze, di interesse collocazionale o sintattico, dove abbisogni avere un controllo fine sulla larghezza della finestra su cui avviene il match.

Cosa vogliamo dire con *operatore di intervallo* sarà immediatamente chiaro se riprendiamo gli esempi di query col *match any* del gruppo 2088 qui sopra: dire che in `"per" []{1,3} "tempo"` il secondo match concatenato deve ricorrere almeno 1 ed al massimo 3 volte equivale a dire che il primo ed il terzo match possono avere un intervallo da una a massimo tre parole tra di loro. Normalmente, ossia

x **`[]{n,m}`** *y* indica l'intervallo max. e min. di token tra struttura *x* ed *y*.

Tav. 270: Combinazione di *match any* ed operatore di intervallo.

[24] Si badi, che stante la forma del Tagset Ver. 1.4, la stringa `v.m` nella POS può stare tanto per verbi principali (`main`) quanto modali (`modal`), ma non ausiliari.

21.2.8.3 "FLAGS". Nel linguaggio CQP sono chiamati *flags* tre parametri che modificano alcune RegExp; tali parametri sono introdotti dal metacarattere `%`, ed hanno peraltro una distribuzione assai limitata: modificano solo strutture delimitate dal doppio apice `"..."`:

%d	flag-d	ignora diacritici (`a` fa match indifferentemente su `à á â a`)
%c	flag-c	*case insensitive* (`a` fa match indifferentemente su `A a`)
%l	flag-l	uso letterale

Tav. 271a: I flag in CQP.

I flag sono però combinabili tra loro, così ad es.

%dc	ignora diacritici e maiuscole (`a` fa match indifferentemente su `A À à á â a` ecc.)

Tav. 271b: Combinazione di flag in CQP.

Pochi esempi, credo, basteranno per chiarirne funzioni e sintassi.

La query 2091a col flag `%d` coglie l'intersezione (almeno) delle due query 2091b (cfr. es. 2091d) e 2091c (cfr. es. 2091d), ed una ricerca mirata a cogliere la consistenza dei *ché* come lemma autonomo rispetto a *che* potrebbe ben usarla come query di controllo. Si noti peraltro che esistono alternative, sintatticamente più libere, per esprimere le medesime condizioni, come le quadre di alternanza (per cui cfr. § 21.2.9): la query 2091f è infatti un perfetto equivalente della 2091a.

[2091a] `"Che" %d` (149)[25] *query CQP*,
[2091b] `"Che"` (132) *query CQP*,
[2091c] `"Ché"` (17) *query CQP*,
[2091d] `« Lassa , ch' io non so leggere ! **Ché** molto lo saprei volentieri » .` *Novellino*, lxxxxvj.6, p. 337,
[2091e] `**Che** pensiero ti mosse ?` *Novellino*, vij.22, p. 144,
[2091a] `"Ch[eé]"` (149) *query CQP*.

Analogamente, la query 2092a col flag `%d` coglie l'intersezione delle due query 2092b (cfr. es. 2092d) e 2092c (cfr. es. 2092e); query di questo tipo (al di là del semplice scopo lessicale di avere tutte le attestazioni di un token, maiuscolo o minuscolo che sia) potrebbero ad esempio servire a controllare quanto gli editori abbiano "personificato" entità astratte, come nella psicomachia di Bono. E si noti come anche in questo caso esistano alternative, sintatticamente più libere, per esprimere le medesime condizioni, sempre basate sulle quadre di alternanza (per cui cfr. § 21.2.9): la query 2092f è infatti un perfetto equivalente della 2092a.

[2092a] `"fede" %c` (234) *query CQP*,
[2092b] `"Fede"` (165) *query CQP*,
[2092c] `"fede"` (69) *query CQP*,;
[2092d] `De l' albergheria de la **Fede** Cristiana .` Bono, *Libro Vizi*, xv.10, p. 31,
[2092e] `Inn amistade nè in **fede** non ricevere uomo folle .` *Fiore Filosafi*, xv.2, p. 142,
[2092f] `"[Ff]ede"` (234) *query CQP*.

Il flag letterale, infine, è una alternativa al metacarattere letterale (cfr. § 21.2.5.2) il cui uso, si badi bene, è diversamente distribuito, come si può constatare confrontando i risultati delle due query 2093a e 2093b, che magari immagineremmo essere affatto equivalenti:

[25] Quando opportuno per verificare le intersezione della varie query, ne riporto di séguito il numero di risultati in Times e tra tonde.

[2093a] `".*+.*" %l` (“no matches”) *query CQP non funzionante*;
[2093b] `".*\+.*"` (46) *query CQP*;
[2093c] `[...] ; e più di` **`V+M~`** `persone afogaro .` *Cronica fiorentina*, mccxxxviiij, p. 127.

La diversità di risultato è dovuta al diverso dominio di applicazione della “letteralità”, che col flag è attribuita a tutta la query (sicché CQP cerca *letteralmente* la stringa di caratteri “`.*\+.*`”, e naturalmente non la trova), mentre col metacarattere è attribuita al solo carattere alla sua destra (e quindi viene preso come *carattere letterale* il solo “+”, mentre tutto il resto della query viene letto come espressione regolare).

La query 2093b, peraltro, che potrebbe parere di scarso interesse linguistico, in realtà una sua utilità ce l’ha in un corpus di italiano antico: quella di trovare i numerali additivi rovesciati (*cinque e mille*, nell’es. 2093c).

Per concludere, vorrei ancora sottolineare la limitata disponibilità di questi parametri, sostanzialmente ristretta, come dicevamo, solo alle espressioni tra doppio apice: se, ad esempio, la query 2091a è perfettamente legale (ed infatti fornisce risultati come 2091de) e perfettamente legale è 2094b (con 72 match che comprendono tutte le congiunzioni subordinanti che inizino con *C* maiuscola e finiscano con *-e*, come *Che*, *Come* e *Chome*, come gli ess. 2094cd, ma non *Ché*), non così è 2094a (che catturerebbe anche *Ché*), alla cui proposta il CQP reagisce con «syntax error, unexpected FLAG»; uno può sempre, però, formulare la query equivalente 2094e, che cattura anche i *Ché*, come quello dell’es. 2091d qui sopra.

[2094a] `[kat=".*51.*" & word="C.*e"] %d` *query CQP erronea*,
[2094b] `[kat=".*51.*" & word="C.*e"]` (72) *query CQP*,
[2094c] **`Come`** `s' inpongano le penitenze .` *Capitoli San Gillio*, ij.22, p. 50,
[2094d] **`Che`** `÷' capitani visitino li 'nfermi .` *Capitoli San Gillio*, ij.21, p. 49,
[2094e] `[kat=".*51.*" & word="C.*[eé]"]` (89) *query CQP*.

21.2.9 REGEXP INTERNE: QUADRE DI ALTERNANZA. Alcune porzioni di stringa di una affermazione di "*valore*" (già in sé una RegExp) possono essere ulteriormente trasformate o modificate in un *pattern* da espressioni più complesse (potremmo *grosso modo* pensarle come delle RegExp incassate) ma più precise dei *wildcharacters* (cfr. § 21.2.4.3), la cui potenza, a volte eccessiva, avevamo già avuto occasione di osservare. Si tratta, perlopiù, di quelle che chiameremo le *quadre di alternanza*.

Le parentesi quadre, infatti, oltre che (a) delimitare un match semplice ma non su `word` (cfr. § 21.2.4), (b) delimitare un match complesso formato da una espressione booleana (cfr. § 21.2.12), e (c) costruire il *match any* (cfr. § 21.2.5.1), servono ad esprimere alcuni *patterns* di alternanza interni al match. Alcuni ne abbiamo già introdotti man mano nei paragrafi precedenti, quando ve n’era bisogno, ma li ripresenteremo qui organicamente; le strutture base, comunque si possono riassumere in tre tipi,

`[tT]`	‘***t*** maiuscola o minuscola’
`[abc...]`	‘o ***a*** *o* ***b*** o ***c*** ...’
`[a-d]`	‘qualsiasi carattere da ***a*** a ***d*** (cioè ***a***, ***b***, ***c*** e***d***)’

Tav. 272a: Strutture base delle quadre di alternanza in CQP.

in tutti i quali si ha propriamente una disgiunzione, ossia un’annotazione abbreviata di quello che, per esteso, implicherebbe il connettore *or* (cfr. § 21.2.6 e soprattutto 21.2.10):

`t|T`	‘***t*** maiuscola o minuscola’
`a|b|c|...`	‘o ***a*** *o* ***b*** o ***c*** ..., ecc.’

Tav. 272b: Corrispondenti con *or* delle quadre di alternanza.

Il primo tipo, come avevamo già notato (§ 21.2.8.3), equivale al flag *case*, cfr. il gruppo di query ed esempi 2092, rispetto al quale è indubbiamente più flessibile, cfr. ad es. la versione quantificata nella query 2089g (con l'es. 2089h) nel § 21.2.8.2. Sicché non serve, spero, aggiungere altro.

E mentre il terzo tipo in un corpus come il CT non ha molte applicazioni, è invece il secondo che manifesta la massima flessibilità d'impiego, in parte compensando anche per la assenza, nel linguaggio CQP, delle comodissime (e comunissime) RegExp

/d	*digit*	[0-9]
/w	*word character*	[A-Za-z0-9_]

Tav. 273: RegExp assenti in CQP.

In particolare si presti attenzione alle seguenti possibili query:

[2095a] `"[09]"` *query CQP*,
[2095b] `"[1-9]"` *query CQP*,
[2095c] `( [...] , dì **10** di maggio )` *Cronica fiorentina*, mlxxiiij, p. 89,
[2095d] `"[aeiou].{10}"` *query CQP*,
[2095e] `"[AEIOU].{10}"` *query CQP*,
[2095f] `"v[ei]"` *query CQP*,
[2095g] `E però ti vo' qui **ammaestrare** [...]` Bono, *Libro Vizi*, xiiij.9, p. 31,
[2095h] `; ano&(verò&) **Inchilberto** in mia mano .` *Libro Riccomanni*, xxxxviiij.1, p. 552,
[2095i] `; e da sezzo vo' che **vi** segnoreggi la Morte [...] .` Bono, *Libro Vizi*, vj,11, p. 17,
[2095j] `[...]non vo' che **ve** ne crucciate [...] .` Bono, *Libro Vizi*, vj,12, p. 17,
[2095k] `"[bcdfghjklmnpqrstvxzBCDFGHJKLMNPRSTVXZ]{4}.*"` *query CQP*,
[2095l] `[word="[bcdfghjklmnpqrstvxzBCDFGHJKLMNPRSTVXZ]{4}.*" & pos !=".*num.*"]` *query CQP*,
[2095m] `"[bcdfghjklmnpqrstvxzBCDFGHJKLMNPRSTVXZ]{1,3}[aeiou]{2}"` *query CQP*,
[2095n] `"[bcdfghjklmnpqrstvxzBCDFGHJKLMNPRSTVXZ]{1,3}[aeou]i"` *query CQP*,
[2095o] `"[bcdfghjklmnpqrstvxz]{1,3}u[bcdfghjklmnpqrstvxz]{1,3}u.*"` *query CQP*,
[2095p] `, non fa mistiere di **sscrivere** qui ,` Consiglio e Lapo de' Cerchi, *Lettera*, j.6, p. 601,
[2095q] `**Die** aiuta !` Bono, *Libro Vizi*, lxxv.3, p. 117,
[2095r] `, veggio c' **hai** male di paura ,` Bono, *Libro Vizi*, lxxv.3, p. 117,
[2095s] `, i seguagi de ÷l decto Amerigo ongni sozza cosa liberamente faceano in ongni **bruttura** di pecchato .` *Cronica fiorentina*, mccvij, p. 116.

Le query 2095ab servono a trovare i caratteri numerici nel corpus, la prima tutti in modo indifferenziato e la seconda solo quelli intieri e non multipli di 10, e nel corpus, peraltro ve ne è solo una occorrenza (es. 2095c). Le query 2095df usano la RegExp per 'tutte le vocali': 2095d per trovare tutte le parole di 10 caratteri che iniziano con vocale minuscola (es. 2095g), 2095e per trovare tutte quelle con vocali maiuscole (es. 2095h), 2095f per pescare tutti i clitici di seconda plurale sia che abbiano vocale *e* che vocale *i* (ess. 2095ij). Le query 2095k-o lavorano intorno alla RegExp per 'tutte le consonanti': 2095k cerca tutte le parole che iniziano con gruppi di 4 grafemi consonantici (il risultato, però è assai sporco perché più del 90% dei 441 match sono cifre latine; per avere cifre più pulite bisogna ricorrere alla query complessa 2095l che esprime anche una condizione negativa, cfr. § 21.2.11, sulla POS, ottenendo i 15 match pertinenti, tutti di natura grafematica), cfr. l'es. 2095p; 2095m trova

tutti i monosillabi in dittongo, cfr. es. 2095q, mentre 2095n limita la ricerca a quelli con *-i* seconda parte di dittongo, cfr. es. 2095r; 2095o può essere usata per cercare effetti fonosimbolici relativi alla ripetizione della vocale *u* in due sillabe della parola, cfr. es. 2095s.

Comunque, all'infuori da queste espressioni in qualche modo *figées* con le quadre di alternanza, sono normalmente le tonde a delimitare RegExp interne, incassate in un match, ed a permetterne quantificazione e parametrizzazione, cfr. risp. i §§ 21.2.5.1-2 e 21.2.8.1-2, dove ne abbiamo già presentato gli esempi più significativi.

21.2.10 DISGIUNZIONE. Tra gli operatori elencati nel § 21.2.6 la disgiunzione è uno dei più potenti e versatili, ma anche a volte dei più problematici e difficili da usare.

|	*or*	disgiunzione (*o*)

Tav. 274: La disgiunzione in CQP.

Infatti si può applicare praticamente a tutti i livelli possibili nel linguaggio CQP: da quello di carattere (come abbiamo già visto in molti dei paragrafi precedenti), a quello di match (ossia di struttura interna di una espressione booleana semplice) a quello di espressione booleana complessa (ossia delle concatenazioni di più match), cfr. § 21.2.12.

A livello di carattere, in primo luogo, tutte le espressioni con le quadre di alternanza (cfr. § 21.2.9), sia quelle maiuscolo/minuscolo (query 2096b = 2096a, già 2092f, cfr. ess. 2092de) sia quelle con normali alternanze (query 2096d = 2096c, già 2095f, cfr. ess. 2095ij), sono in realtà delle notazioni abbreviate per espressioni con "or", che comunque puossono essere impiegate per query più elaborate, come ad es. la 2096e, dove si noti il ruolo determinante delle parentesi tonde, che cerca tutte le grafie *ct/z_i* in almeno bisillabi o più (cfr. ess. 2096fg), o come la già esaminata 2089e (cfr. *supra*).

[2096a] `"[Ff]ede"` *query CQP*,
[2096b] `"(F|f)ede"` *query CQP*,
[2096c] `"v[ei]"` *query CQP*,
[2096d] `"v(e|i)`[26]`"` *query CQP*,
[2096e] `".{5}((ct)|z)i.*"` (533) *query CQP*.
[2096f] « Domando ÷ti donde se' e di che **condizione** » *Novellino*, vij.14, p. 143.
[2096g] Che si faccia **correctione** due volte in sei mesi .
Capitoli S. Gillio, j.37, p. 41.

A livello di match (espressione booleana semplice) serve normalmente a disgiungere due diversi valori del medesimo attributo[27]; in tal caso il match completo è di solito tra quadre (cfr. query 2098a) , a meno che non si tratti di due valori alternativi dell'attributo `word`,

[26] Si noti, tra l'altro, che per una certa liberalità di formalismo del CQP, oltre alle notazioni abbreviate `[Ff]`, `[ei]`, ecc. ed estese `(F|f)`, `(e|i)`, ecc. funzionano anche le meno corrette notazioni ibride `[F|f]`, `[e|i]`, ecc.

[27] Disgiungere due diversi attributi sul medesimo match, invece, sarebbe operazione perlomeno curiosa e dalle finalità dubbie; e comunque non sarebbe davvero possibile in CQP, poiché una query come la 2097a seguente,
[2097a] `[kat=".*50." | word="Italia"]` *query CQP*,
a qualsiasi cosa mai possa servire, equivarrebbe semplicemente a
[2097b] `"Italia"` *query CQP*,
per via della proprietà logica della disgiunzione di CQP che si trova illustrata poco oltre, in fondo al paragrafo. Una query complessa su una sola posizione e con disgiunzione sul medesimo attributo è invece ben possibile e verosimile, e ne avevamo già visto un esempio con la 2080h nel § 21.2.7.

come in 2098d, che peraltro è un ennesimo equivalente di 2096d = 2096c, già 2095f (cfr. ess. 2095ij):

[2098a] `[lemma="canto" | lemma="cantare"] (54 matches)` *query CQP*,
[2098b] `[lemma="canto"] (8 matches)` *query CQP*,
[2098c] `[lemma="cantare"] (46 matches)` *query CQP*,
[2098d] `[lemma="canto" & lemma="cantare"] (No matches)` *query CQP*,
[2098e] `"ve" | "vi"` *query CQP*,
[2098f] `; e l' altra si mise in uno` **`canto`** `de ÷lla casa` *Novellino*, lxxj.2, p. 290,
[2098g] `« Io non` **`cantero`** `÷e mai s' io non ho pace da mia donna »` *Novellino*, lxiiij.16, p. 273.

Si noti peraltro che il risultato della disgiunzione è l'unione di due insiemi (il risultato di 2098a è l'unione dell'insieme dei risultati di 2098b, cfr. es. 2098e, e 2098c, cfr. es. 2098f), laddove quello della congiunzione è la loro intersezione (il risultato di 2098d è pertanto zero, l'intersezione dei risultati di 2098b e 2098c essendo appunto tale: non v'è nel CT alcun token che sia al contempo verbo e nome, ma naturalmente avrebbe potuto esservi se la disambiguazione non fosse stata condotta a fondo come invece è stata, cfr. ¶ 9)

Inoltre, la (prevedibile) equivalenza di risultati non comporta affatto medesimo funzionamento: si noti, infatti, che nelle tre alternative 2099abc, pur identiche nei risultati, la disgiunzione in realtà si applica logicamente in modo diverso: nella prima, 2099a, la disgiunzione è tra due singoli caratteri, nella seconda, 2099b, tra due stringhe di caratteri, e nella terza, 2099c, tra due attributi; la differenza, invece tra 2099c e 2099d è puramente notazionale, l'una essendo abbreviazione dell'altra:

[2099a] `"(F|f)ede"` *query CQP*,
[2099b] `"(Fede)|(fede)"` *query CQP*,
[2099c] `"Fede" | "fede"` *query CQP*,
[2099d] `[word="Fede"] | [word="fede"]`[28] *query CQP*.

A livello, infine, di espressione booleana complessa, la disgiunzione può essere articolata su un solo match, come nell'ultimo esempio presentato (2099d), od essere articolata su più alternative ognuna composta di più match: in altri termini, l'*or* non può direttamente disgiungere due diverse posizioni nel corpus (a cosa, infatti servirebbe mai?), ma può disgiungere due query distinte, riunendole in un'unica query che costituisce l'unione delle due query disgiunte.

Si considerino, ad esempio, le tre query 2100b, 2100d e 2100f, costruite per trovare proto-combinazioni di "nome + cognome" nel CT, la prima solo le serie "nome + cognome" (cfr. es. 2100c), la seconda solo le serie "nome + *de/di/da* + cognome" (cfr. es. 2100e), e la terza quelle con anche articolo interposto (cfr. es. 2100g): la query combinata in disgiunzione 2100a coglie l'unione dei tre sottoinsiemi delle tre query 2100b, 2100d e 2100f (968 match = 606+206+156).

[2100a] `([pos=".*n.p.*"] [pos=".*n.p.*"]) | ([pos=".*n.p.*"] "d[aei]" [pos=".*n.p.*"]) | ([pos=".*n.p.*"] "d[aei]" [pos=".*art.*"] [pos=".*n.p.*"])` (968) *query CQP*,
[2100b] `[pos=".*n.p.*"] [pos=".*n.p.*"]` (606) *query CQP*,

[28] E si noti, riguardo a quanto dicevamo due capoversi sopra riguardo la differenza con la congiunzione che la sostituzione in questa query della disgiunzione con la congiunzione, ossia `[word="Fede"] & [word="fede"]`, chiederebbe una cosa logicamente impossibile (cercherebbe, ossia, una cosa che sia al contempo sé stessa ed un'altra), e coerentemente il sistema risponderebbe con «syntax error, unexpected '&'».

[2100c] , esendo podestate messer **Currado Orlandi** ,
Cronica fiorentina, mccviiij, p. 117.
[2100d] [pos=".*n.p.*"] "d[aei]" [pos=".*n.p.*"] (206) *query CQP*,
[2100e] : la detta charta fece ser **Albizo da Lancano** notaio ,
Libro Riccomanni, xxviiij.2, p. 539,
[2100f] [pos=".*n.p.*"] "d[aei]" [pos=".*art.*"] [pos=".*n.p.*"] (156)
query CQP,
[2100g] , sì come fu ÷e maestro **Piero da ÷lle Vigne** , Brunetto, *Rettorica*, j.5, p. 5,
[2100h] [pos=".*n.p.*"] "d[aei]"? [pos=".*art.*"]? [pos=".*n.p.*"]
(977) *query CQP*.

Si consideri peraltro che risultati simili a quelli della query 2100a, assai lunga ma accurata, (a) potrebbero essere ottenuti con la 2100h, molto più sintetica ma meno accurata (ha piccola perdita di precisione nei risultati); oppure (b), affidandosi alla annotazione che i curatori del corpus già hanno introdotto, facendo ricorso agli attributi multiword, in ispecie a `mwlkat`.

La caratteristica, comunque, più idiosincratica e meno naturale della disgiunzione nel linguaggio CQP, è la sua cosiddetta *first match strategy*: particolare attenzione bisogna pertanto porre nel valutare la portata *pratica* al di là di quella *logica* dei risultati di una query con disgiunzione su match. In che cosa il problema consista lo spiega assai chiaramente il manuale di riferimento del CQP (Christ et alii 1999): «whereas `CQP` enumerates all possible matches for a disjunctive regular expression over characters, `CQP` employs a (left-to-right) *first match strategy* for disjunctive regular expressions over corpus positions. This means the following: If the prefix of one of the disjuncts equals another disjunct, the longer disjunct will be ignored. I.e. in the query result, there will be no matches for the longer disjunct. – This applies only to the whole query, not to its embedded disjunctive expressions. For example, in the following query, the disjunct which encodes a sequence of three proper names», `[pos=".*n.p.*"]`, «is useless, since one of its prefixes is the other disjunct which asks only for two proper names. Hence, practically, albeit not logically, the query» 2101 «is equivalent to the query» 2100b, qui sopra; e «this kind of incompleteness which is caused by the first match strategy applies to all the operations which will be introduced subsequently!».

[2101] `([pos=".*n.p.*"] [pos=".*n.p.*"]) | ([pos=".*n.p.*"] [pos=".*n.p.*"] [pos=".*n.p.*"])` (606) *query CQP*.

21.2.11 NEGAZIONE. La negazione, uno degli operatori elencati nel § 21.2.6, non ha la medesima latitudine di applicazione della disgiunzione (cfr. § 21.2.10), ma è ugualmente assai importante e versatile.

!	*not*	negazione (*non*)

Tav. 275: La negazione in CQP.

Nei termini più generali, la negazione opera sulla espressione booleana alla sua destra: in altri termini, il suo dominio di applicazione non è mai una RegExp interna ad un match, ma solo una intiera espressione booleana, ossia un match od una espressione complessa, delimitato, pertanto, esplicitamente da una parentesi tonda (la costruzione è però possibile anche implicitamente, senza parentesi, cfr. query 2102d *infra*). Si confrontino infatti, a mo' di illustrazione, la query 2102a (finalizzata ad indagini analoghe a quelle del gruppo 2098 *supra*), ben formata (e che difatti può fare match con esempi come 2098e *supra*), a quelle 2102b (che vorrebbe trovare le forme di *avere* di prima persona non precedute da un *io*) e 2102c (che vorrebbe trovare le parole che non iniziano per vocale), entrambe mal formate, perché si applicano ad espressioni tra quadre e (nel caso di 2102c) ad una RegExp interna:

[2102a] `[word="canto" & ! (pos=".*v.m.*")]` (12) *query CQP*,
[2102b] `![word="io"] & [lemma="avere.*" & kat=.*1,0,6.*"]` *query CQP erronea*,
[2102c] `"![aeiou].*"` *query CQP erronea*,
[2102d] `[word="canto" & ! pos=".*v.m.*"]` (12) *query CQP*,
[2102e] `[word="canto" & pos!=".*v.m.*"]` (12) *query CQP*,
[2102f] `[word!="io"] [lemma="avere.*" & kat=".*1,0,6.*"]` (197) *query CQP*,
[2102g] `[word!="[aeiou].*"]` (1000+) *query CQP*.

Assai utile è la possibilità in CQP di una notazione abbreviata, in cui la negazione si applica direttamente all'operatore di uguaglianza producendo un operatore composto

!= *value negation* negazione di attributo (*non vale "x"*)

Tav. 276: La negazione di attributo in CQP,

sicché 2102a può più semplicemente scriversi 2102e; e non solo: la *value negation* ci fornisce anche dei modi semplici ed efficaci (2102f e 2102g) per formulare questa volta correttamente 2102b e 2102c.

Per il resto, è utile fare riferimento alla logica che presiede le equivalenze in tutte le espressioni booleane; in particolare, si presti attenzione che una negazione su disgiunzione equivale alla congiunzione di due negazioni, ossia, che 2103a è la stessa cosa di 2103c, e che, stante la validità della scorciatoia della "value negation", equivale in definitiva a 2103c; inoltre, nessuna delle tre query 2103a prenderà solo le parole *aiuto* che non siano verbo (istituendo un tipico controllo di transcategorizzazioni), bensì tutte le parole che non abbiano la forma *aiuto* e non siano nomi (ossia gli *aiuto* verbo più tutte le altre parole del corpus): tutti gli *aiuto* non verbi sono invece presi da una query tipo 2103d, cfr. es. 2103e, così come tutti gli *aiuto* non nome sono invece presi da una query tipo 2103f, cfr. es. 2103g, e l'unione di 2103d e 2103g equivale alla query disgiunta 2103h.

[2103a] `[!(word = "aiuto" | pos = ".*v.m.*")]` *query CQP*,
[2103b] `[!(word = "aiuto") & ! (pos = ".*v.m.*")]` *query CQP*,
[2103c] `[word != "aiuto" & pos != ".*v.m.*"]` *query CQP*,
[2103d] `[word = "aiuto" & pos != ".*v.m.*"]` (24) *query CQP*,
[2103e] `– Sí prometto co l' **aiuto** e a la speranza di Dio – .`
Bono, *Libro Vizi*, lxxvj.17, p. 119,
[2103f] `[word = "aiuto" & pos != ".n.c*"]` (1) *query CQP*;
[2103g] `Levo ÷ssi questa femina et **aiuto** ÷llo [...] .` *Novellino*, xiij.18, p. 216,
[2103h] `([word = "aiuto" & pos != ".*v.m.*"]) | ([word = "aiuto" & pos != ".n.c*"])` (25) *query CQP*.

21.2.12 STRUTTURA INTERNA DI UNA ESPRESSIONE BOOLEANA VS. CONCATENAZIONI DI ESPRESSIONI BOOLEANE. Poche osservazioni conclusive sulla macrostruttura delle query saranno ormai sufficienti.

La struttura interna di un'espressione booleana, ossia di una singola posizione del corpus, dovrebbe infatti essere ormai evidente dagli esempi precedenti. Questa si può articolare su un solo match (come ad es. in 2073a o 2099a) o su una combinazione di più match (come ad es. in 2071c, 2102a o 2102e ed in tutti casi presentati nel § 21.2.7), ed è normalmente delimitata dalle parentesi quadre.

La formulazione di più espressioni booleane, corrispondenti a più posizioni del corpus, avviene invece normalmente tramite concatenazione, ossia il semplice *blank* (spazio vuoto), come ad es. in 2100h o 2087b.

Ed un'intera espressione del primo o del secondo tipo (in breve: una query) si può ulteriormente coordinare ad un'altra query tramite l'uso delle parentesi tonde, come ad es. in 2100a o 2103h.

I singoli operatori, poi, hanno delle restrizioni, in base al tipo di espressione in cui ricorrono: ma questi si sono bastantemente discussi dove pertinente nei paragrafi precedenti.

21.2.13 USO DI ETICHETTE DI VARIABILI ("LABELS"). Una tecnica relativamente avanzata[29] di CQP disponibile anche nella versione Web, che consente di espandere la già grande flessibilità del linguaggio ancora oltre i limiti delle RegExp e delle espressioni booleane fin qui esaminate, o comunque di semplificarne almeno in qualche caso la sintassi, è quella di "etichettare" alcune strutture anche complesse che corrispondono ad una posizione del corpus e poi di usare direttamente tale etichetta (*label*) all'interno di un'unica query.

La sintassi di base di questa tecnica è data in Tav. 277 (a), in cui la annotazione del token cui si fa riferimento viene attuata come in (b):

(a) `label:[`*`pattern`*`] :: `*`global constraint`*,
(b) `label.`*`attributo`*, ecc.

Tav. 277: Etichette di variabili: (a) sintassi di base e (b) annotazione del token.

L'applicazione di questa tecnica, più caratteristicamente utile in locale, anche nella versione web del CT consiste nell'usare le *labels* come nomi di variabili non definite (*a*, *b*, ecc.): ciò, infatti, spesso torna utile per formulare query non altrimenti possibili con semplici RegExp (come 2104a, derivata da Evert 2005), o comodo per formularne altre in modo più semplice (come in 2104b, tratta da Heid 2000):

[2104a] `a:[] "e" b:[] :: a.word = b.word` (13) *query CQP*,
[2104b] `a:[] :: a.msform != a.philform` (414) *query CQP*,
[2104c] `« Messer , **tanto e tanto** » .` *Novellino*, xxiiij.4, p. 186,
[2104d] `ebe ÷ne madona Konttessa [**Kottessa/Ko[n]ttessa**] trentta [**tretta/tre[n]tta**] livre` *Libro Castra*, j.19, p. 209.

La query 2104a fa match con tutte le fraseologie costituite da due parole identiche collegate da una *e* (letteralmente richiede [j] che la prima posizione sia *a*, che la seconda posizione sia occupata dal token "e", e che la terza posizione sia *b*; e [ij] che *a* sia il medesimo token di *b*), cfr. es. 2104c; la query 2104b, invece, cerca tutti i token in cui la forma editoriale scelta dal filologo (`philform`) differisce da quella (`msform`) del manoscritto (letteralmente richiede [iij] che *a* sia ogni token del corpus; (b) che la `msform` di ciascun *a* sia diversa dalla sua `philform`), cfr. es. 2104d.

Si noti anche che le *labels* possono essere usate, con poca variazione sintattica, anche all'interno di una struttura (purché non sia la medesima struttura cui si riferiscono), come si vede dalla query seguente (sempre derivata da Evert 2005),

[2105a] `a:[] [pos = a.pos]{3}` (78) *query CQP*,
[2105b] `, **sì come qui dinanzi** fu ÷e detto .` Brunetto, *Rettorica*, xxxxvij.2, p. 102.

che cattura tutte le occorrenze di 4 token di séguito con POS uguale (tutti *avverbi* o `foreign`).

21.2.14 SPECIALI COMANDI CQP. CQP è dotato di un discreto numero di comandi; molti di questi però non sono raggiungibili dalla versione web (cfr. § 20.3 e sottoparagrafi,

[29] Abbastanza avanzata da essere descritta in Evert 2005 (§ 4.1, cui si rimanda per una esposizione meno limitata della presente) ma non in Christ et alii 1999.

in particolare 20.3.5), e non ne faremo pertanto cenno. Altri, però, possono essere utili anche in questa versione, in particolare il `within` e lo `show`, e ne diremo pertanto qualcosa nei sottoparagrafi seguenti (risp. §§ 21.2.14.1 e 21.2.14.2).

Oltre a questi va menzionato, in quanto in sé fondamentale, il segno che rende una stringa di caratteri una legale query di CQP:

`;`	*semicolon*	carattere di chiusura di una query di CQP

Tav. 278: Il marcatore di chiusura query in CQP

Il suo uso, indispensabile in locale, nelle query via web è già aggiunto di default dall'interfaccia, rendendone facoltativa (e ridondante) l'introduzione manuale. Conformemente, in tutti gli esempi di query presentati in questo capitolo (ed in questo volume tutto, fuorché nel capitolo 19), ci siamo sempre astenuti dall'impiegarlo, anche se è bene che l'utente sia avvertito della sua implicita presenza.

21.2.14.1 WITHIN. Il comando palesemente più utile anche nella versione web è il `within` che consente di precisare l'àmbito di un attributo strutturale al cui interno deve avvenire la query precedente

`[query] within` *`attributo`*	

Tav. 279a: Sintassi generale di `within`.

Gli attributi strutturali, ricordiamo (cfr. § 19.1.2.3, 20.1.3, 21.1.1, ecc.), non sono direttamente interrogabili; ma possiamo accedervi indirettamente col comando `within`: se consideriamo, ad es. la Tav. 257 (§ 21.2.1), il testo (cioè la successione delle posizioni direttamente interrogabili) è racchiuso dall'attributo strutturale *XML-like* `<line> .... </line>`; con il comando `within` noi possiamo limitare la nostra query all'interno di quell'attributo.

E se per la riga ciò non pare rivestire particolare interesse linguistico, non è così per altri attributi. Si badi, tuttavia, che anche l'attributo `line` potrebbe avere qualche uso per ricerche specifiche sulla lingua poetica, stante la coincidenza nelle opere versificate con il verso, combinando in modo accorto un `within line` con l'attributo posizionale `typ`: la query 2106a seguente[30], ad esempio, cerca tutti gli aggettivi collocati del sostantivo *cuore* che non siano su *enjambement*:

[2106a] `[lemma = "cuore"] [pos=".*adj.*" & typ="V"] within line` (18) *query CQP*,
[2106b] `[...] ,| i quali eran venuti per difesa | de ÷l` **`cor dolente`** `che gli avea chiamati .` Cavalcanti, *Rime*, x.3, son. *Vedete ch' i' son un*, v. 10, pp. 503.

La combinazione più importante è comunque senz'altro quella `within s`[31]

`[query] within s`	query solo all'interno di frase

Tav. 279b: Il comando `within s`.

che stabilisce che il dominio di una query non esca dai confini di una frase: evidente ne è infatti la potenzialità di ottenere risultati più puliti in query su posizioni multiple (come 2107a, mirata ad individuare le collocazione verbo+nome ad ampio spettro), ma anche in query relativamente semplici in cui possa cadere un segno di interpunzione all'interno di un *match any* od analoga espressione, come in 2107b, che cerca di trovare i collocati preferenziali del verbo *rispondere*:

[30] La query complementare, però, non è possibile perché la negazione non si può applicare ad un `within`.
[31] Per modalità ed efficienza della marcatura dei confini di frase cfr. Barbera - Colombo - Corino - Onesti *i.s.*

[2107a] `[pos=".v.[a].*"]{0,2} [pos=".v.[m|md|a].*"] [pos=".*adv.*"]{1,2} [pos=".*art.*"]{0,2} [pos=".*n.c.*"] within s` *query CQP*,
[2107b] `[lemma="rispondere.*"] []{0,3} [pos=".*n.c.*"] within s` (74) *query CQP*,
[2107c] `[lemma="rispondere.*"] []{0,3} [pos=".*n.c.*"]` (82) *query CQP*,
[2107d] « Io non **risponderò di questo peccato** » Brunetto, *Rettorica*, xxxviij.7, p. 91,
[2107e] Tutte avemo inteso ciò c' ànno detto e qui apresso vi ne **rissponderemo . Sopra le saie** di Luia che volemo per quest' anno v' avemo isscritto Consiglio de' Cerchi, *Lettera*, j.1, pp. 593-594.

Così la query generica 2107c sarà sporcata anche da esempi come 2107e, che saranno invece evitati da 2107b, che farà match prevalentemente con gli esempi voluti, tipo es. 2107d.

Precipuo interesse linguistico potrebbero essere anche query limitate per `typ` alla prosa od al verso o per `genre`, ma purtroppo qui andiamo incontro ad uno scoglio (che possiamo però sempre aggirare!): con il comando `within`, infatti, si può richiamare solo l'attributo posizionale, non i suoi valori, per cui una *query* `within did` abortirebbe, mentre una *query* `within genr` funzionerebbe, ma di fatto cercherebbe le stringhe corrispondenti alla query *in tutto il corpus*: cosa che non riveste ovviamente utilità alcuna. Per questa ragione sono stati creati (cfr. § 20.1.3) degli attributi posizionali gemelli, `genre` e `typ`, per permetterne l'interrogazione diretta. Ad esempio, le query seguenti, che riescono ad accertare la pertinenza ad un dominio specialistico specifico (cfr. query 2108a ed es. 2108c vs. query 2108b ed es. 2108d) della seconda accezione del termine *ragione*, anche se la query 2108e col `within` è sintatticamente erronea:

[2108a] `[pos=".*v.+.ind.*"] []{0,3} [lemma="ragione" & genre = "Doc"]` (52) *query CQP*,
[2108b] `[pos=".*v.+.ind.*"] []{0,3} [lemma="ragione" & genre =! "Doc"]` (158) *query CQP*,[32]
[2108c] levamo ove dovea dare a sua **ragione** qui di sopra *Libro Riccomanni*, xxvij.2, p. 536,
[2108d] L' uomo cúpido e tenace è una sustanzia sanza **ragione** : Bono, *Libro Vizi*, v.16, p. 13,
[2108e] `[pos=".*v.+.ind.*"] []{0,3} [lemma="ragione"] within Doc` *query CQP erronea*.

Per le medesime ragioni anche gli attributi `author` e `title` non si prestano all'uso con `within`; ma in questo caso non sono stati predisposti doppioni posizionali per consentirne l'interrogazione, parendo che, al bilancio di costi (ulteriore appesantimento del corpus) e ricavi (interrogabilità per titolo ed autore), il gioco non ne valesse la candela: in fin dei conti, *nemo tenetur ad impossibilia*, soprattutto se non ne vale la pena.

Utilizzabile e, di fatto, utile per le ricerche con multiword è invece l'attributo `mwl`. Riprendiamo, per esemplificare, il campo semantico e terminologico precedente di *ragione*: la prima query, esplorativa, 2109a è di fatto equivalente a 2109b, e trova strutture come quella in es. 2109c; ben più fini però sono i risultati (ad es. 2109e) della query 2109d, che mi consente di trovare le MW preposizionali a pronome variabile con *ragione* proprie del genere documentario; e posso, addirittura, con 2109f ulteriormente restringere la ricerca alle sole MW che non solo soddisfino le condizioni precedenti, ma che siano anche frutto di un intervento editoriale, come l'esempio 2109g:

[2109a] `[lemma="ragione"] within mwl` (52) *query CQP*,

[32] Si noti anche che, in questo modo, torna (cfr. invece n. 30 § 21.2.14.1) ad essere possibile la query negativa.

[2109b] `[lemma="ragione" & mwlword=.*ragione.*]` (52) *query CQP*,
[2109c] `non hai male onde` **`per ragione`** `debbi morire` Bono, *Libro Vizi*, iij.12, p. 7,
[2109d] `[pos=".*adp.*"] [] [lemma="ragione" & genre = "Doc"] within mwl` (22) *query CQP*,
[2109e] **`levamo da loro ragone`** `ove doveano dare` *Libro Riccomanni*, j.2, p. 517,
[2109f] `[pos=".*adp.*"] [] [lemma="ragione" & genre = "Doc" & corr="y"] within mwl` *query CQP*,
[2109g] `ponemo` (msform `po` / philform `po(nemo)`) **`a loro ragone`** (msform `ragon` / philform `ragon¦e¦`) *Libro Riccomanni*, j.2, p. 517.

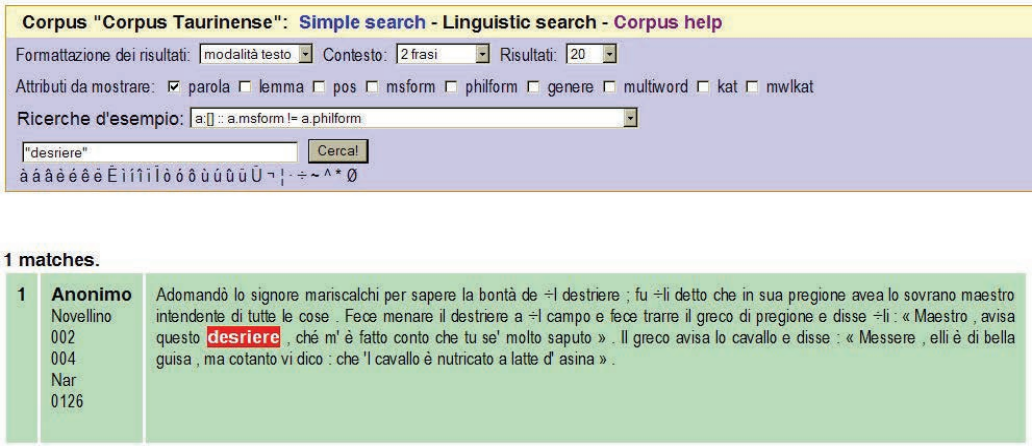
21.2.14.2 SHOW. Il comando `show` è propriamente uno *switch* (cioè un interruttore) che attiva o disattiva la visualizzazione di un attributo, strutturale o posizionale che sia:

`show +attribute`	attiva la visualizzazione di un attributo
`show -attribute`	disattiva la visualizzazione di un attributo

Tav. 280: Sintassi del comando `show`.

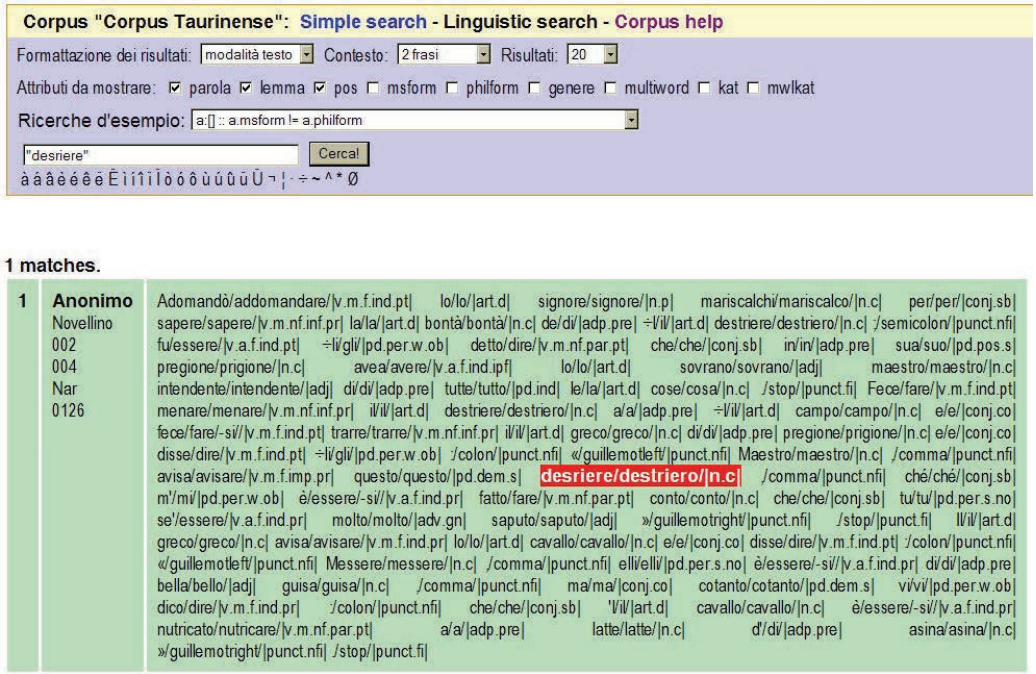
Nella versione web del CT (dove il comando non è utilizzabile direttamente: cfr. §§ 19.1.4.2 e 20.3.3) alcune possibilità di `show` sono già predisposte dall'interfaccia. In particolare sulla colonna sinistra sono visualizzati di default alcuni attributi strutturali (nell'ordine: `author`, `title`, `chapter`, `paragraph`, `genr` e `page`), mentre nella testata della pagina sono selezionabili gli attributi posizionali da mostrare (oltre a `word`, che è il default, sono disponibili `lemma`, `pos`, `msform`, `philform`, `genre`, `mwlword`, `kat` e `mwlkat`).

Ad esempio la semplice query `"desriere"` (che ha nel corpus un solo match) nella visualizzazione di default risulta come in Tav. 281,

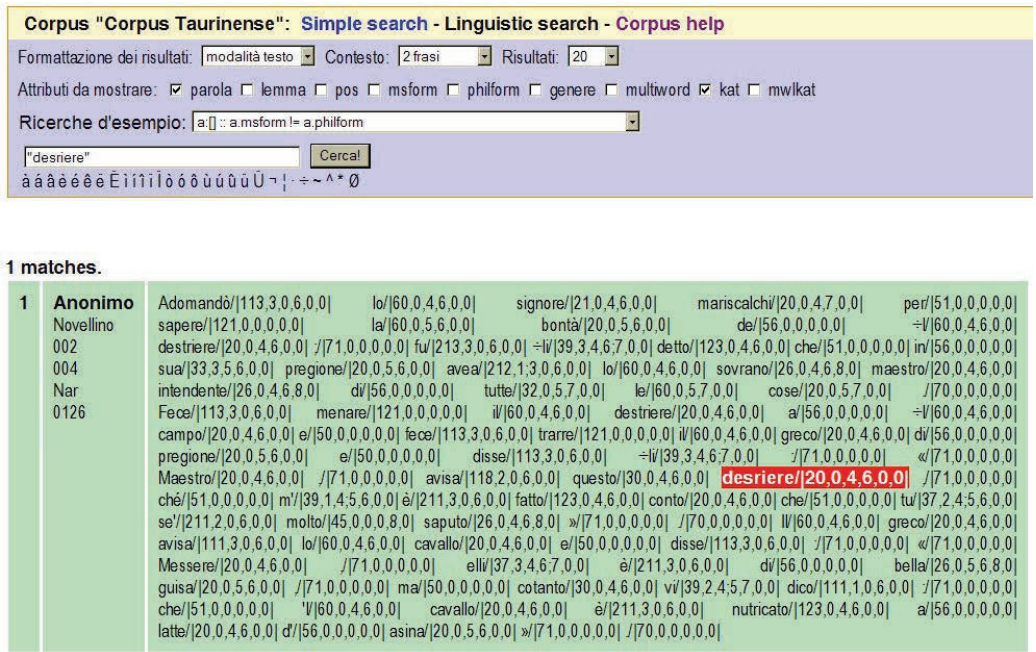


Tav. 281: Query `"desriere"` nella visualizzazione di default (`word`).

scegliendo invece di visualizzare anche `lemma` e `pos` riuscirebbe invece come in Tav. 282; e scegliendo infine di visualizzare `kat` sarebbe come in Tav. 283,



Tav. 282: Query `"desriere"` con visualizzazione di `word`, `lemma` e `pos`.



Tav. 283: Query `"desriere"` con visualizzazione di `word` e `kat`.

21.2.15 IDIOSINCRASIE DEL CT VERSIONE-CQP: "TIPS AND TRICKS". Come forse il lettore si sarà ormai avveduto, per interrogare il CT si impongono a volte alcune strategie specifiche, diverse dallo *standard average CQP*.

La prima e più cospicua è che ai valori degli attributi `pos` e `kat` non si accede direttamente (cfr. 2110a) ma contornandoli con *wildcharacters* (cfr. 2110b) senza i quali la query (cfr. 2110b) sortirebbe «no matches».

[2110a] `[lemma="casa"]` (110) *query CQP*,
[2110b] `[lemma="casa" & kat=".*20.*"]` (110) *query CQP*,
[2110c] `[lemma="casa" & kat="20"]` (*no matches*) *query CQP erronea*.

Ciò è dovuto alla particolare forma di annotazione dei due attributi (secondo già osservato nel § 19.1.4.1), che materialmente si presentano come

[2111] `|n.c|` `|20,0,5,6,0,0|` `pos` *e* `kat` *in forma-CQP*,

per cui, ovviamente, la stringa, ad es., *20* è presa da `.*20.*` ma non da `20`.

Il sistema non è senza controindicazioni, perché

[2112] `[kat=".*21.*"]` (1000+) *query CQP*

non coglierà solo i token taggati come nomi propri (`21`) ma molti altri, tra cui, soprattutto quelli verbali etichettati `121`, ecc. (ad es. `fare/121,0,0,0,0,0`). Il rimedio, però, se si tiene sempre ben in mente la struttura del bastone di annotazione, non è difficile da trovare: la query 2113

[2113] `[kat!=".*[0-9]{3}.*" & kat=".*21.*"]` (1000+) *query CQP*,

con l'aggiunta della condizione negativa che impedisce le sequenze di tre cifre (verbi), risolve elegantemente il problema limitando i risultati ai soli nomi propri (`21`); e, per converso, una condizione negativa che vieti le sequenze di *due* cifre consentirà di prendere i `121` senza i `21`.

Il sistema è esportabile anche alle multiword, per la cui annotazione non si dispone di POS ma solo di numeriche `mwlkat`; infatti

[2114] `[mwlkat=".*21.*"]` (1000+) *query CQP*,

non coglierà solo le MW il cui lemma sia stato taggato come nome proprio (`21`) ma molte altre, tra cui, soprattutto quelle verbali etichettate `121`, ecc. (ad es. `fare/ben°fare°/121,0,0,0,0,121`). La query 2115 seguente, costruita al modo di 2113, però,

[2115] `[mwlkat!=".*[0-9]{3}.*" & mwlkat=".*21.*"]` (1000+) *query CQP*,

coglie sì *solo* le MW `21` (ad esclusione delle `121`, ecc.), ma ripetute tante volte quante sono i loro costituenti-MW su cui il tag è spalmato; se quello che si cerca è *il totale* dei token MW con quella `mwlkat`, allora bisognerà seguire un'altra strategia[33], aggiungendo un controllo sul token successivo:

[2116] `[mwlkat!=".*[0-9]{3}.*" & mwlkat=".*21.*"] [(mwlkat="--")|(mwlkat=".*,1")]` (712) *query CQP*.

Un'analoga avvertenza va fatta anche con i lemmi verbali: essendo marcati anche i pronominali, spesso vi sono due lemmi distinti, ad es. *destare/-si/* e *destare*: pertanto la query 2117a coglierà solo una parte delle forme di *destare* (quelle lemmatizzate `destare`, cfr. es. 2117d), la query 2117b prenderà l'altra parte (quelle lemmatizzate `destare/-si/`, cfr. es.

[33] Come, infatti, si è qui fatto nel § 18.7.1, dove la query 2116 compare come 2030a.

2117e), mentre 2117c, la struttura più disponibile, le prenderà tutte (incluse quelle lemmatizzate `destare/-si/` e quelle lemmatizzate `destare`, ossia tanto gli esempi tipo 2117d come quelli tipo 2117e):

[2117a] `[lemma="destare"]` (2) *query CQP*,
[2117b] `[lemma="destare/-si/"]` (3) *query CQP*,
[2117c] `[lemma="destare.*"]` (5) *query CQP*,
[2117d] `Pe ÷gli occhi fere un spirito sottile , che fa 'n la mente spirito` **`destare`** , Cavalcanti, *Rime*, xxviij.1, son. *Pegli occhi fere*, v. 2, p. 530,
[2117e] `Quelli` **`si destaro`** `e fecero gran corrotto`
Novellino, lxxxxviiij.19, p. 351.

Sempre a proposito di lemmi, è da ricordare che esistono dei lemmi numerati per distinguere gli omografi (cfr. § 16.2 e sottoparagrafi), che non sarebbero prendibili da semplici query per lemma come 2110a o 2117a. Se la categoria è in blocco tracciabile con la query 2118a [=1210 § 16.2], è bene ricordare che per assicurarsi un risultato basta aggiungere un punto all'indicazione di lemma: 2118b, infatti, restituisce tutti i suoi 41 risultati (come l'es. 2118c), laddove a 2118d il CQP reagirebbe con un laconico «no matches»:

[2118a] `[lemma=".*[A-Za-z].*[0-9]"]` *query CQP*;
[2118b] `[lemma="prigione."]` (41) *query CQP*;
[2118c] `e XV+C~ di` **`prigioni`** `[lemma="prigione1"] Pisani ne fuoro menati presi in Firenze .` *Cronica fiorentina*, mccxx, p. 121,
[2118d] `[lemma="prigione"]` (0) *query CQP*.

Va naturalmente da sé che nella fattispecie una query con *prigione* formulata come 2117c avrebbe sortito gli stessi risultati della 2118b, che tuttavia è molto meno permissiva (il lemma deve essere seguito da un carattere, laddove in 2117c può o no essere seguito da qualsiasi stringa di caratteri).

Per concludere, è forse qui la sede di ricordare ancora due piccole idiosincrasie del CT, ossia la persistenza di 8 ineliminabili transcategorizzazioni esterne (cfr. § 9.0) e di una sola (ma di buona frequenza) transcategorizzazione lemmatica (§ 16.3). Idiosincrasie che sono prendibili in blocco con le query, rispettivamente, 2119a (=57 nel § 9.0) e 2119 (=1227b nel § 16.3)

[2119a] `[lemma=".*\|.*"]` *query CQP*;
[2119b] `[lemma=".+\;.+"]` *query CQP*.

intorno alle quali molte altre ricerche più complesse si possono agevolmente impostare: al lettore, ormai esperto, la facoltà di sbizzarrirvicisi.