ID	**Templates, part of Learning Algorithm**	**Predictors, automatically generated**	**Effect**	**Description**
`Once up on a time there lived a poor widow and her son Jack.`				
1	*person isA thing;* *wife isA person;* *widow isA wife;*	None	No immediate impact yet, but any instances including *p0001* in Row #3 with the type *widow* can be predicted as a *wife, person*, and *thing* through the lifecycle of the NLP process.	Before the three templates entered, the database was empty except for *thing* the only user defined data representing the sole root type.
2	*adverB (once up on a time) := there_is $t: [$t isA time] where $t start < $t end and $t end < '1/1/1000';*	*t0001 isA time;* (where an internal structure is created for: *tp0001 start < tp0001 end < '1/1/1000'*).	The L mode process actually started an inquiry first by executing the expression *there_is ....* It created the data because it didn't find any relevant data in the new context that was established for the Jack and Bean Stalk forktale.	The node *t0001* contains partial information because its *start* and *end* are not assigned with an exact date. A time always has a start time and an end time for a period. When the start and end times are the same, a time becomes a point of time. The value '1/1/1000' is randomly chosen for a demonstration purpose to reference a time in the past.
3	*family isA thing;* *a widow := there_is $f: [$f isA family], $p: [$p isA person] where $f $p isA widow;*	*p0001 isA person; f0001 isA family; f0001 p0001 isA widow;*	*p0001, f0001, and f0001 p0001* reflect the basic information from the sample text, an interpretation from human experience.	The word `widow` must be involved with a group of people, *family*. The *f0001* family is created to have *p0001* and later *jack* and c0001(*a cow*) as members.
4	*adJ poor $p: [$p isA person] := there_is $f:[$f isA family] where ($f $p != null and $f (bE be) (ajD poor));*	*f0001 (bE be) (adJ poor);*	The inquiry `who is poor`? would have an answer by matching expressions like *f0001 (bE be) (adJ poor).*	Adjectives like *poor* is semantically more complex, involving statistics. No additional adjectives are discussed in this table. Additionally, we manage to say Jack's family is poor, instead of saying Jack's mom is poor.
5	*her son := (coreF her) S son;*	No data is created, but a interim syntax transformation from *her son* to *p0001 S son*.	No effect, only a syntax transformation	*coreF* abbreviates coreference. *coreF her* retrieves the `widow` in the story. *S* abbreviates the `'s` symbol after a person's name. Therefore `her son` is transformed as `the widow's son`
6	*son isA person;* *mother isA person;* *person S son : = there_is $f: [$f isA family], $s: [$s isA person] where $f person isA mother and $f $s isA son;*	*p0002 isA person;* *f0001 p0002 isA son;* *f0001 p0001 isA mother;*	*p0002* is the new data added in database, referring to Jack. Also specify that the widow is a mother.	The phrases like `her son` have a fixed syntactical form. They are defined once and reused for later to parse other text. In the 3rd assignment of the "Template" column, *person* and *son* are global types and acting as variables.
7	*name isA thing;* *person name : = (person namE == name);*	*p0002 namE := jack;*	*jack* from now on is a coreference to *p*0002, i.e., *coreF jack := p0002.*	The text `son Jack` matches the 2nd template. `Jack` is categorized as a name while others are possible, e.g., a machine. All text from users are converted to small cases while capital cases are memorized separately.

8	*there (verB live) person := person (verB live);*	*p0001 (verB live) (preP in) t0001;* *p0002 (verB live) (preP in) t0001;*	The entire English sentence is now not mapped to two EP terms, each represents the state of being living without detailed semantics for now	The parser splits the text into two sentences because of the conjunction word `and`. Therefore, there are two corresponding EP terms.
9	*husband isA person;* *widow (dO do) (noT not) (verB live) (preP with) husband := there_is $f: [$f isA family] where widow {+ $f and ! (there_is $h : [$h isA husband] where $h {+ $f);*	no new data is created because there is no text in corresponding this template. This template is optional to give a constraint that a widow doesn't not have or live with a husband.	This constraint may not necessarily be enforced. But It can be invoked for validation in an I mode.	We could add more semantics by adding more templates like this one to enrich the understanding of this sentence.
`One day, Jack's mother told him to sell their only cow.`				
10	*adverB (one day) := there_is $t: [$t isA day] where (coreF (one day)) start < $t and $t < (coreF (one day)) end;*	*t0002* (where an internal structure is created for *t0001 end* $\leq$ *t0002 start* $\leq$ *t0001 end, and t0002 end - t0002 start = 24 hours)*	*coreF*(*one day*) is mapped to be "Once upon a time" that was recorded earlier. *t0002* is created for a period of time within once upon a time.	The phrase `One day` can be a future day or a past day but unsure exactly which day it would be when it serves as an adverb in a sentence. In the given sentence, it is a past tense and within the time period *t0001* set by `Once up on a time`. *coreF (one day)* finds out the tense of the sentence first, e.g., the past tense in this case, and then searches a previously defined time period constraining `one day`.
11	*coreF name := there_is $p: [$p isA person] where $p namE == name;*	No data is created	*coreF jack* returns *p0002*.	`Jack` may not only refers to a person but also others such as a tool to lift a car. Therefore `Jack` was tried to be parsed in different categories before confirming it is the name for p0002.
12	*person S mother : = there_is $f: [$f isA family], $p: [$p isA person] where ($f person isA son or $f person isA daughter) and $f $p isA mother;*	No data is created	*f0001 p0001* is returned as Jack's mother, where Jack is an instance of *person*.	Since the L mode process found the instance *f0001 p0001*, it doesn't create a new one but retrieve the existing one.
13	*animal isA thing;* *livestock isA animal;* *cow isA livestock;*	No data is created.	No immediate impact yet	However, any instances including *c0001* in Row #15 with the type *cow* can be predicted as a *livestock, animal*, and *thing* through the lifecycle of the NLP process.
14	*coreF their;*	No data is created	*f0001* is returned. Like other coreferences, *coreF their* is determined by a built-in process	*coreF they* returns Jack and his mother, but *coreF their* returns *f*0001, something shared by both Jack and his mother.
15	*family S cow := family cow isA livestock;*	*c0001 isA cow* *f0001 c0001 isA livestock*	*c0001, f0001 c0001* reflect the basic information from the original text `their cow`	In the L mode, the process tried to find a cow instance in *f*0001 and created *c*0001 because no one was found.

16	*person (verB tell) $p: [$p isA person] Infinitive;*
	person (verB sell) $t : [$t isA thing];
		p0001 tell jack (jack sell c0001 ((PreP at) Notime)); Note the action represented by *tell* is implicitly modified by a time within the given `One day`. Therefore the sentence is actually in past tense because the system can tell.	The constructed data is a predictor, which represents an action taken by Jack's mother. Subsequent queries can be answered such as `what did Jack's mother tell Jack?`, `Who told Jack to sell their cow?`, etc..	*Infinitive* is a built-in operator indicating the following text is an infinitive phrase, i.e., `to do ....` The built-in node *Notime* indicates that *jack sell c1000* is not a fact yet as it may or may not happen. This sentence with the verb *sell*, converted from the infinitive clause, will be further updated to make the *sell* a fact in Row #28.

17	*desirE 1 := desire; desirE 2 := like; desirE 2.5 := want, desirE 3 := hint; desirE 4 := encourage; desirE 5 := tell; desirE 6: = ask; desirE 7 := command; desirE 8 := enforce};*	No data was generated		Facing the phrase "Jack's mother told him to ...", we optionally construct a template *desirE* that places all verbs related to "desire" in a sequence to reflect the degree of desires. This template helps to correlate similar sentences together to find paraphrase sentences.
`Jack went to the market and on the way he met a man who wanted to buy his cow.`				
18	*coreF name := there_is $p: [$p isA person] where $p namE == name;*	No data is created	*coreF jack* returns *p0002*.	The same process as discussed in #11.
19	*location isA thing;* *market isA location*	No data was created	No immediate impact yet	However any instances including *m0001* in Row # 20 with the type *market* can be predicted as a *market, location*, and *thing* through the lifecycle of the NLP process.
20	*the market := there_is $m: [$m isA market];* Note `a market` would be given the same definition as our process doesn't rely on `a` or `the` vigorously	*m0001 isA market*	*m0001* reflects the basic information of the original text `the market`, an interpretation from human experience.	While this template is defined based on human experience, it can also be derived by the sentence `Jack went to the market`, where `the market` can be reasoned as a location, where there is a template like the one in #21.

21	*person (verB go) (preP from) $l1: [$l1 isA location] (preP to) $l2: [l2 isA location] := (update person geoLoc := $l2 geoLoc);*	*l0002 isA location;*
		*f0001 geoLoc := l0002*;
		p0002 geoLoc := m0001;
			l0002 refers to Jack's home location, a dump node without information	There should be a standard geographical (and time) data calculation to construct the first and second assignments on the Predictor column. The update command in the template enforces an update for both L and O modes

22	*adverB (on the way (preP from) $l1: [$l1 isA location] (preP to) $l2: [$l2 isA location]) := there_is $l: [$l isA location] where $l* `is between` *$l1* `and` *$l2;*	*l0003 isA location;* where *l0003* `is between` *l0002* `and` *m0001*;	the geographical distance should be implemented in a standard geographical calculation package	
23	*a man = there_is $p: [$p isA person];* Note a man can be defined with more attributes but we simply define it as a person just for demonstration purpose.	*p0003 isA person;*	A template for "the man" would be the same one for "a man", as our process doesn't differentiate "a" from "the" to tolerate human errors.	However, the parser would prefer to create a new person because of "a man" is given. Otherwise, considering "the man" being Jack himself would make the sentence "Jack met the man", where "the man" is Jack, not meaningful.

24	*person (verB meet) $p: [$p isA person] (preP at) location := ((person geoLoc == location) and ($p geoLoc == location));*	*p0002 geoLoc := l0003; p0003 geoLoc := l0003;*	The template enforces the geoLoc of the two persons to be changed	The template can be enriched with more attributes, but the same location the two persons met is the highlight of this template.
25	*person (verB want) infinitive; person (verB buy) $t: [$t isA thing];*	*p0003 (verB want) (p0003 (verB buy) c0001 ((preP at) Notime));*	`his cow` is mapped to *c0001* based on the similar process we discussed earlier	*person (verB want) infinitive* is very similar to *person (verB tell) $p: [$p isA person] infinitive* and they can be correlated using the *desirE* template in #17.

`Jack took the magic beans and gave the man the cow.` Note: in our discussion, we skipped the conversations between Jack and the man, where the man's 5 magic beans would be traded to Jack for Jack's cow. To simplify our discussion, we assume only one bean and the following data have been generated: *f0002 isA family; f0002 p0003; bean isA thing; b0001 isA bean; b0001 (bE be) (adJ magic); f0002 bean;* where we consistently set up an organization, such as a family, a person belongs to.

26	*person (verB take) thing from $p: [$p isA person] := Botran ($p (verB give) person thing);*	No data is generated.	The text `Jack took the magic beans` is to be converted to *p0003 (verB give) p0002 b0001* in #27.	It defines that a person takes a thing from another is equivalent to that the second person gives the first person the thing. The original text doesn't have the phrase `from the man` but the template gives the parser a hind to find it.

27	*person (verB give) $p1 : [$p1 isA person] $t: [îsA thing] := (person geoLoc == $p1 geoLoc),*
	delete coreF (family person) $t,
	create coreF (family $p1) $t;
	
	coreF (family person) := select $f: [$f isA family] where $f person != null;
		f0001 b0001 and *f0002 c0001* were added into database, and *f*0002 *b*0001 and *f*0001 *c*0001 are removed from database. The results come from the intermediate expressions: *p0003 (verB give) p0002 b0001; p0002 (verB give) p0003 c0001;*	The template is aimed to first validate that the persons and the good to be exchanged are next to each other, e.g., the geographical coordinates are the same. Then it update the belongings of both Jack and the man in their family accounts.	When a sentence implies actions like "give", we use the Froglingo update commands in template explicitly to enforce the action for both L and O modes. Also the built-in term *coreF* can be user-defined this time.

28	*person (verB sell) $g: [$g isA thing] := there_is $buyer: [$buyer isA person] where person (verB give) $buyer $g;*	No data was generated as there is no text to trigger an execution on it	This template would help to validate `Did Jack sell his cow?`, which is related to the word "sell" in the sentence "Jack's mother told him to sell their only cow" at an I mode	An extra step to demonstrate that new information can be derived from the sample text by using the template.
29	*person (verB buy) $g: [$g isA thing] := there_is $seller: [$seller isA person] where $seller (verB give) person $g;*	No data was generated	Not used in this demonstration. This template would help to answer the question: `Did the man buy his cow?`	An extra step to demonstrate that new information can be derived from the sample text by using the template.

`What does the function fac take 5 to produce?` Note: though natural language itself is ambiguous, there are some text that can precisely express vigorous mathematical expressions.

30	*multiplication $n1: [$n1 isa number] $n2: [$n2 isa number] = ($n1 multi $n2);* *fac (verB take) $n:[$n isA number] (preP to) (verB produce) $m:[$m isA number] = Botran ("if" $n "is 0," $m "is 1 or" $m" is the multiplication of" $n "with what fac takes"* ($n − 1) *"to produce;"*);	No data is generated in database, but respond with an answer of 120.	The answer to the given text is 120. The template acts precisely as a factorial function	