This is a system that implements a simple form of temporal semantics within Prolog. By accessing the system upon Prolog, ten prepared sentences with temporal expression are processed and expected outcomes are presented. With instruction of the file ‘clsem-ex3-sose2019-final’, both types of direct references and indirect references to time are employed in building this system. This report is attached to the system. It is divided into two subparts, among which description of detailed files is the first part, a summary of potential issues takes the residual part.

1. Description
The entire system is consisted of following four files.
1.1 exampleModels.pl
1.2 modelChecker2.pl
1.3 modelCheckerTestSuite.pl
1.4 dcg2final.pl
1.1 Firstly, the model is built under the file ‘exampleModels.pl’ as example7. Vocabularies are extracted from these ten sentences. The first argument of model/2 is a list of domain entities. Besides, the second argument is the interpretation function F. To be more specific, there are totally fifteen different entities in the domain, namely, from d1 to d15, each one of them is further interpreted by the function. Three constants ‘vincent’, ‘mia’, and ‘butch’ are listed with arity of zero. Five entities are nouns entitled with one arity. Then, as core part of this model, temporal expression is initially constructed as direct references at this phrase. Function representation of time applies exact date to entities, they have arity of one individually.

f(0,'01,07,2019',d9), f(0,'02,07,2019',d10), f(0,'03,07,2019',d11), f(0,'04,07,2019',d12), f(0,’05,07,2019',d13), f(0,'06,07,2019',d14), f(0,’07,07,2019',d15),

Upon these representation of actual date, specific weekday can be built at ease.

f(1,monday,d9), f(1,sunday,d14), f(1,tuesday,d10),

As ’04,07,2019’ is labelled as the evaluation time, which can be viewed as the current event, a few more abstract concepts of time can be generated. *Today* is represented as *a day overlaps with evaluation time*, and *yesterday* is a two place predicate with sequential date as each entry of its argument. Inversely, *tomorrow* can be interpreted as two sequential date also, but in the opposite sequence with yesterday.

f(2,today,[(d16,d9),(d17,d10),(d18,d11),(d19,d12),(d20,d13),(d21,d14),(d22,d15)]), f(2,yesterday,[(d9,d10),(d10,d11),(d11,d12),(d12,d13),(d14,d15)]), f(2,tomorrow, [(d10,d9),(d11,d10),(d12,d11),(d13,d12),(d14,d13),(d15,d14)]),

The next step is to include more concepts such as *before*, *after* and *overlap*. By applying the same method, they can be listed as:

f(2,before,[(d9,d10),(d10,d11),(d11,d12),(d12,d13),(d13,d14),(d14,d15)]), f(2,after, [(d10,d9),(d11,d10),(d12,d11),(d13,d12),(d14,d13)]), f(2,overlap,[(d16,d9),(d17,d10), (d18,d11),(d19,d12),(d20,d13),(d21,d14),(d22,d15)]),

Temporal expression is treated as quantifiers. Temporal variables should be added into the interpretation of verbs, in order to be triggered at the same time when verbs take place. Then the past tense without specific date indicated can be expressed by adding temporal entity of abstract concepts. However, for those sentences with clear date indicated, only the entity of this date is added.

f(2,dance,[(d1,d14),(d1,d13),(d1,d12),(d1,d11),(d1,d10),(d1,d9)]), f(3,visit,[(d1,d2,d9)]), f(2,sleep,[(d1,d16),(d1,d17),(d1,d18),(d1,d19),(d1,d20),(d1,d21), (d1,d22)]), f(4,give,[(d1,d2,d4,d14)]), f(3,rob,[(d2,d5,d9),(d2,d5,d10),(d2,d5,d11), (d2,d5,d12),(d2,d5,d13),(d2,d5,d14)]), f(3,buy,[(d1,d6,d10),(d1,d6,d11),(d1,d6,d12), (d1,d6,d13),(d1,d6,d14),(d1,d6,d15)]), f(3,visit,[(d2,d3,d16),(d2,d3,d17),(d2,d3,d18), (d2,d3,d19),(d2,d3,d20),(d2,d3,d21),(d2,d3,d22)]), f(4,give,[(d3,d1,d7,d10)]), f(3,eat, [(d3,d8,d9),(d3,d8,d10),(d3,d8,d11),(d3,d8,d12),(d3,d8,d13),(d3,d8,d14)]), f(3,kill, [(d3,d1,d9),(d3,d1,d10),(d3,d1,d11),(d3,d1,d12)(d3,d1,d13),(d3,d1,d14)])])).

1.2 Model checker is built upon the existing ‘modelChecker2.pl’ file. The original model checker

only includes one place predicate and two place predicate, so that it is able to check predicate which

has maximum of two arguments. So, another two chunks of three place predicate and four place predicate should be added to improve the functionality. For three place predicate, there are specifically three parts to consider. The first part is a situation when a formula matches model, the second part deals with a negative situation that a formula does not match model. In both of these situations above, compose/3 is used at the beginning to decompose a formula. Then, a prolog interpretation function i/4 is used to assign different treatments to the formula, treatments depend on either the formula is a variable or a constant. However, last step is

reverse for these two situations. memberList/2 is added to check if the interpretation is a possible value for the predicate. For the positive one, value should be in the values list, while it should not in the list for negative one. The residual part tests if a formula is undefined in model.

```
satisfy(Formula,model(D,F),G,pos):- nonvar(Formula), compose(Formula,Symbol,
[Arg1,Arg2,Arg3]),
\+ memberList(Symbol,[eq,imp,or,and,some,all]), i(Arg1,model(D,F),G,Value1),
i(Arg2,model(D,F),G,Value2), i(Arg3,model(D,F),G,Value3),
memberList(f(3,Symbol,Values),F), memberList((Value1,Value2,Value3),Values).

satisfy(Formula,model(D,F),G,neg):- nonvar(Formula), compose(Formula,Symbol,
[Arg1,Arg2,Arg3]),
\+ memberList(Symbol,[eq,imp,or,and,some,all]), i(Arg1,model(D,F),G,Value1),
i(Arg2,model(D,F),G,Value2), i(Arg3,model(D,F),G,Value3),
memberList(f(3,Symbol,Values),F),

\+ memberList((Value1,Value2,Value3),Values). satisfy(Formula,model(D,F),G,undef):-

nonvar(Formula), compose(Formula,Symbol,[Arg1,Arg2,Arg3]),
\+ memberList(Symbol,[eq,imp,or,and,some,all]), (

\+ var(Arg1),

\+ atom(Arg1) ;

\+ var(Arg2),

\+ atom(Arg2) ;

\+ var(Arg3),

\+ atom(Arg3) ;

var(Arg1),

\+ i(Arg1,model(D,F),G,_) ;

var(Arg2),

\+ i(Arg2,model(D,F),G,_) ;

var(Arg3),

\+ i(Arg3,model(D,F),G,_) ;

atom(Arg1),

\+ i(Arg1,model(D,F),G,_) ;
```

```
atom(Arg2),

\+ i(Arg2,model(D,F),G,_) ;

atom(Arg3),

\+ i(Arg3,model(D,F),G,_) ;

\+ memberList(f(3,Symbol,_),F) ).

satisfy(Formula,model(D,F),G,pos):- nonvar(Formula), compose(Formula,Symbol,
[Arg1,Arg2,Arg3,Arg4]), \+ memberList(Symbol,[eq,imp,or,and,some,all]),
i(Arg1,model(D,F),G,Value1), i(Arg2,model(D,F),G,Value2),
i(Arg3,model(D,F),G,Value3), i(Arg4,model(D,F),G,Value4),
memberList(f(4,Symbol,Values),F), memberList((Value1,Value2,Value3,Value4),Values).

satisfy(Formula,model(D,F),G,neg):- nonvar(Formula), compose(Formula,Symbol,
[Arg1,Arg2,Arg3,Arg4]), \+ memberList(Symbol,[eq,imp,or,and,some,all]),
i(Arg1,model(D,F),G,Value1), i(Arg2,model(D,F),G,Value2),
i(Arg3,model(D,F),G,Value3), i(Arg4,model(D,F),G,Value4),
memberList(f(4,Symbol,Values),F),

\+ memberList((Value1,Value2,Value3,Value4),Values).
satisfy(Formula,model(D,F),G,undef):-

nonvar(Formula), compose(Formula,Symbol,[Arg1,Arg2,Arg3,Arg4]), \+
memberList(Symbol,[eq,imp,or,and,some,all]),
(

\+ var(Arg1),

\+ atom(Arg1) ;

\+ var(Arg2),

\+ atom(Arg2) ;

\+ var(Arg3),

\+ atom(Arg3) ;

\+ var(Arg4),

\+ atom(Arg4) ;

var(Arg1),

\+ i(Arg1,model(D,F),G,_) ;
```

var(Arg2),

\+ i(Arg2,model(D,F),G,_) ;

var(Arg3),

\+ i(Arg3,model(D,F),G,_) ;

var(Arg4),

\+ i(Arg4,model(D,F),G,_) ;

atom(Arg1),

\+ i(Arg1,model(D,F),G,_) ;

atom(Arg2),

\+ i(Arg2,model(D,F),G,_) ;

atom(Arg3),

\+ i(Arg3,model(D,F),G,_) ;

atom(Arg4),

\+ i(Arg4,model(D,F),G,_) ;

\+ memberList(f(4,Symbol,_),F)).

1.3 Before evaluation, the model has to be tested. Testing is created upon ‘modelCheckerTestSuite.pl’, it contains representation of semantic analysis of ten sentences.

test(some(X,and(before(X,'04,07,2019'),dance(vincent,X))),7,[],pos).
test(some(X,and(before(X,'04,07,2019'),monday(X),visit(vincent,mia,X))),7,[],pos).
test(some(X,and(overlap(X,'04,07,2019'),today(X),sleep(vincent,X))),7,[],pos).
test(some(X,and(after(X,'04,07,2019'),and(sunday(X),some(Y,and(footmassage(Y),give(vincent,mia,Y,X)))))),7,[],pos).
test(some(X,and(before(X,'04,07,2019'),and(yesterday(X),some(Y,and(bank(Y),rob(mia,Y,X)))))),7,[],pos).
test(some(X,and(after(X,'04,07,2019'),and(tomorrow(X),some(Y,and(newcar(Y),buy(vincent,Y,X)))))),7,[],pos). test(some(X,and(overlap(X,'04,07,2019'),visit(mia,butch,X))),7,[],pos).
test(some(X,and(before(X,'04,07,2019'),and(tuesday(X),some(Y,and(warning(Y),give(butch,vincent,Y,X)))))),7,[],pos).

test(some(X,and(before(X,04,07,2019),all(Y,imp(burger(Y),eat(butch,Y,X))))),7,[],pos).
test(some(X,and(after(X,'04,07,2019'),kill(butch,vincent,X))),7,[],pos).

1.4 As a grammar parsing mechanism, DCG is introduced to analyze these sentences and present their semantic relationship. 'dcgfinal.pl' contains only grammatical parsing information, and 'dcg2final.pl' comprises semantic information. The later one is divided into two parts, first part covers all the sentence combining rules, and the second part is about semantic representation of lexical. In the first part, syntactic rules are basically the same as those for parsing, however, an extra app/2 is added to merge subparts together. For sentence combination, because some sentences carry temporal expression at the front, while some other carry that at the end, more than one type of branching rules are listed to deal with different structures. Moreover, merging semantic representation of sentence into temporal representation is the last step, among which tense is temporal representation that inserted in each verb. In addition, sentences are composed of *NP* and *VP*, then *NP* is further achieved through three different paths. The first one is built by combining *TV* and *NP*, second is built solely by *IV*. For the third path, *VP* contains either auxiliary or dative verb is treated with extra sub-branch named *vprime*. *vprime* merges auxiliary with the verb, combines dative verb and direct object before *VPs* process. Apart from that, there is one circumstance that needs further branching, which is sentence four, it includes both auxiliary and dative verb. To deal with it, *vprime* splits into *g_fut* and *np*, within which *g_fut* further breaks up into *vaux_fut* and *give*.

s(app(PP,S),Tense) --> pp(PP),s(S,Tense). s(app(NP,VP),Tense) --> np(NP),vp(VP,Tense).
s(app(PP,S),Tense) --> s(S,Tense),pp(PP). tp(app(Tense,Sem)) --> s(Sem,Tense).

% for subject and object NP np(app(D,N))-->det(D),n(N). np(app(QUAT,N))--
>quat(QUAT),n(N). np(PN) --> pn(PN).

% for PP
pp(T) -->t(T).
pp(app(P,T)) --> p(P),t(T).
% for VP
vp(app(TV,NP),Tense) --> tv(TV,Tense),np(NP).
vp(IV,Tense) --> iv(IV,Tense).
% for VP with AUX or DTV
vp(app(TV,NP),Tense) --> vprime(TV,Tense),np(NP).
vp(IV,Tense) --> vprime(IV,Tense).

% for VP with DTV
vprime(app(DTV,NP),Tense) --> dtv(DTV,Tense),np(NP). vprime(app(DTV,NP),Tense) --> g_fut(DTV,Tense),np(NP). g_fut(app(VAUX,GIVE),Tense) --> vaux_fut(VAUX),give(GIVE,Tense). % for VP with AUX
vprime(app(VAUX,V),Tense) --> vaux_pre(VAUX),v_pre(V,Tense).
vprime(app(VAUX,V),Tense) --> vaux_fut(VAUX),v_fut(V,Tense).

The second part handles semantic representation of lexical. Different from pure description of verbs, tense information is included as an argument of verbs. To process temporal information appropriately, a variable T is added into lambda expression.

/
**==
====== Transitive Verbs*

*==
=====*/*
tv(lam(X,lam(Y,lam(T,app(X,lam(Z,visit(Y,Z,T)))))),lam(P,some(T,and(before(T,'04,07,2019'),app(P,T))))) --> [visited].
tv(lam(X,lam(Y,lam(T,app(X,lam(Z,rob(Y,Z,T)))))),lam(P,some(T,and(before(T,'04,07,2019'),app(P,T))))) --> [robbed].
tv(lam(X,lam(Y,lam(T,app(X,lam(Z,eat(Y,Z,T)))))),lam(P,some(T,and(before(T,'04,07,2019'),app(P,T))))) --> [ate]. /
**==
======*

Intransitive Verbs
*==
=====*/*
iv(lam(X,lam(T,dance(X,T))),lam(P,some(T,and(before(T,'04,07,2019'),app(P,T))))) --> [danced]. /
**==
======*

Vaux
*==
=====*/ vaux_pre(lam(P,P)) --> [is].*
vaux_fut(lam(P,P))-->[will]. /
**==
======*

Verbs for is/will
*==
=====*/*
v_pre(lam(X,lam(T,sleep(X,T))),lam(P,some(T,and(overlap(T,'04,07,2019'),app(P,T))))) --

> [sleeping].
v_pre(lam(X,lam(Y,lam(T,app(X,lam(Z,visit(Y,Z,T)))))),lam(P,some(T,and(overlap(T,
'04,07,2019'),app(P,T))))) --> [visiting].
v_fut(lam(X,lam(Y,lam(T,app(X,lam(Z,buy(Y,Z,T)))))),lam(P,some(T,and(after(T,'04,07,20
19'),app(P,T))))) --> [buy].
v_fut(lam(X,lam(Y,lam(T,app(X,lam(Z,kill(Y,Z,T)))))),lam(P,some(T,and(after(T,'04,07,201
9'),app(P,T))))) --> [kill]. /
**==========*
======

Ditransitive Verbs
==========
=====/*
dtv(lam(W,lam(S,lam(X,lam(T,app(S,lam(Z,app(W,lam(Y,gave(X,Y,Z,T)))))))),lam(P,some
(T,and(before(T,'04,07,2019'), app(P,T))))) -->[gave].
give(lam(W,lam(S,lam(X,lam(T,app(S,lam(Z,app(W,lam(Y,give(X,Y,Z,T)))))))),lam(P,some
(T,and(after(T,'04,07,2019'),a pp(P,T))))) --> [give].

The tricky part for me is the representation of time, there are two different approaches to express time, as mentioned at the beginning, namely, the direct and indirect way. As a consequence, there will be two distinguished patterns to convey direct expression such as *monday*, and indirect

expression such as *yesterday*. For direct expression, there is a preposition word *on* ahead, while

indirect expression stands alone. In order to get an accurate and paralleled outcome, representation of preposition is added to that of indirect temporal expression.

t(lam(X,monday(X))) --> [monday].
t(lam(X,tuesday(X))) --> [tuesday].
t(lam(X,sunday(X))) --> [sunday]. t(lam(Q,lam(X,and(today(X),app(Q,X))))) --> [today].
t(lam(Q,lam(X,and(yesterday(X),app(Q,X))))) --> [yesterday].
t(lam(Q,lam(X,and(tomorrow(X),app(Q,X))))) --> [tomorrow].

2. Potential issues

There are still remaining issues when implementing quantificational tense semantics. By applying anonymous variables to denote time, simple concepts such as past, present, and future are handled well. However, for complex temporal concepts such as past continuous tense, past perfect tense and future perfect tense, it is not the same thing. If adding the tense information into verbs like the what has been processed above, how to represent these temporal concepts which involve a structural overlap? And,

of course, in natural language, some clauses apply certain tense, but they are attached to main sentences with another tense. To clearly represent temporal semantics of such sentences, further reading and research is needed for me. Not much of the discussion is unfolded here, however, it is indeed an interesting and worthy topic. Thanks for such an explicit and progressive instruction involved in this exercise.