ASP Formalism for Ground Instances

1 Background

A schema is a tuple $\langle \phi, \psi, d, Sub, Obj, \rangle$ where ϕ is the antecedent, ψ the consequent, d the deadline condition, Sub the subject, and Obj the object of the schema. The antecedent and consequent is some set of conditions represented by predicates with different kinds of variables as arguments. For simplicity, we regard the deadline condition as a discreet condition that only holds at a single time instant and that does not prolong to future instants.

A ground instance instantiates from a schema when all of the variables are substituted by constants. A ground instance is represented by the constant returned by the function $si(S, Sub, Obj, T)$ where S is the schema it instantiates from, Sub and Obj are the substituted subject and object respectively, and T the first time its antecedent holds. We use ASP to represent and and reason about the ground instances.

2 Commitment Schema

A commitment schema means the subject is committed to the object to bring about the consequent at or before the deadline condition holds, when the antecedent is true. We illustrate how its ground instances are represented using ASP and how to reason about the instances using the following example.

Example 1 (gCom): A guardian is committed to bring his/her child to the hospital within 3 time units after the child gets ill.

The formalization of gCom's ground instances are in Listing 1. Each time of sickness is represented by one ground instance. To distinguish between different times of sickness, we require there is some interval of time during which the child is not sick. The antecedent holds, i.e. $ant()$ is true, when the child is sick. For simplicity, we assume that gCom already exists from the beginning of the timeline. The consequent holds when the guardian brings the child to the hospital after the child first starts to get sick. The deadline condition is true when the child has been sick for three time units. The subject and object are specified by extracting the arguments of an instance. The last statement says we only care about the instances whose antecedent has been true.

The utility predicates are defined as following:

- *becomeSick*(C, T): T is the first time that C is sick during this sickness.
- *remainSick*$(C, T1, T2)$: C is sick at all time instants in $[T1, T2]$. Note: C may be sick at T.

```
1  ant(si(gCom, G, C, T1), T2) :-
     becomeSick(C, T1),
     remainSick(C, T1, T2),
     guardianR(C, G, T2).
5  con(si(gCom, G, C, T'), T) :-
     bring(G, C, hospital, T),
     ant(si(gCom, G, C, T'), T'),
     T >= T'.
   deadCon(si(gCom, G, C, T'), T) :-
10   ant(si(gCom, G, C, T'), T'),
     remainSick(C, T', T),
     T = T'+3.
   sub(si(S, P1, P2, T), P1) :-
     isSiOf(si(S, P1, P2, T), S).
15 obj(si(S, P1, P2, T), P2) :-
     isSiOf(si(S, P1, P2, T), S).
   isSiOf(si(S, P1, P2, T), S) :-
     ant(si(S, P1, P2, T), T').
```

Listing 1: Definition of gCom

The life cycle of an instance consists of three parts: detachment, satisfaction, and violation. For each of the three parts, we distinguish between two states: the state at which the instance becomes detached, satisfied, or violated, which is a single time instant, and the state at which the instance stay detached, satisfied, or violated, which is an interval of time instants. How an instance comes into these states only depends on the type of the schema from

which it is instantiated, regardless of the specific definition of the instance.

A commitment instance becomes detached (Listing 2) when its antecedent first starts to be true. And it remains detached until its deadline condition holds. Notice that an instance is still detached at the time instant when the deadline condition holds, but not the time instant after it holds. The utility predicate *antTrueBefore*(Si, T) means the Si's antecedent is true at all time $T' < T$.

```
becomesDetached(Si, T) :-
  ant(Si, T),
  not antTrueBefore(Si, T).
detached(Si, T) :-
  becomesDetached(Si, T).
detached(Si, T+1) :-
  detached(Si, T),
  not deadCon(Si, T),
  not time(T).
```

Listing 2: Detachment of A Commitment Instance

A detached commitment instance becomes satisfied (Listing 3) at time T if its consequent holds at T for the first time after the instance becomes detached and at or before the deadline condition holds. And it stays satisfied from then on. The predicate *conTrueBetween*$(Si, T1, T2)$ means Si's consequent is true at some time in $[T1, T2]$.

```
becomesSatisfied(Si, T) :-
  becomesDetached(Si, T1),
  deadCon(Si, T2),
  con(Si, T),
  detached(Si, T),
  T1 <= T, T <= T2,
  not conTrueBetween(Si, T1, T-1).
satisfied(Si, T) :-
  becomesSatisfied(Si, T).
satisfied(Si, T+1) :-
  satisfied(Si, T),
  not time(T).
```

Listing 3: Satisfaction of A Commitment Instance

A detached commitment instance becomes violated (Listing 4) at T if its consequent is not true at any time from it becomes detached till the deadline condition holds.

```
becomesViolated(Si, T) :-
  becomesDetached(Si, T1),
  deadCon(Si, T),
  not conTrueBetween(Si, T1, T).
violated(Si, T) :-
  becomesViolated(Si, T).
violated(Si, T+1) :-
  violated(Si, T),
  not time(T).
```

Listing 4: Violation of A Commitment Instance

To illustrate an example instance of *gCom*, we set up the scenario as in Listing 5. Based on the definition of gCom's antecedent, there are three instances we will be considering: 1). si(gCom, bob, alice, 1), 2). si(gCom, bob, alice, 6), and 3). si(gCom, bob, alice, 12). Each instance becomes detached at 1, 6, and 12, respectively. The first one is not satisfied nor violated because Alice stops being sick before reaching the deadline condition. The second one is violated because her guardian, Bob, has not brought her to the hospital before reaching the deadline. The last one becomes satisfied at 14 when Bob brings Alice to the hospital.

```
child(alice).
guardianR(alice, bob, 1..15).
time(15).
sick(alice, 1..3).
sick(alice, 6..9).
sick(alice, 12..15).
bring(bob, alice, hospital, 14).
```

Listing 5: A Scenario Setup of gCom

3 Authorization Schema

An authorization schema means when the antecedent holds, the subject is authorized to bring about the consequent.

Example 2 (shareAut): A physician is authorized to share his/her patients' records with a colleague or the patients' family.

The formalization of shareAut's ground instances is in Listing 6. Similar with the antecedent of gCom instances, the antecedent of a shareAut instance holds at time T if Pat is Phy's patient. The detachment time for the instance is the first time when Pat becomes Phy's patient during this hospital visit. However, the consequent gets complicated compared with gCom. The basic idea of shareAut's consequent is that the physician Phy shares Pat's PHI

with *Phy*'s colleagues or *Pat*'s family. In gCom, we do not care if an instance's consequent holds when its antecedent does not, i.e. whether the guardian brings the child to the hospital when the child is not sick. However, in the case of shareAut, when the antecedent is not true but the consequent is, i.e. the physician shares the PHI of someone who is not the physician's patient, the instance of shareAut is violated. We implement the consequent by dividing it into two cases: the first one is when the antecedent does hold (Lines 4-10), and the second is the antecedent does not hold (Lines 11-13). The notation of $\{famMemberR(); colleagueR()\} >= 1$ is the same as a disjunction saying at least one of the predicates between the curly braces is true. Finally, the deadline condition holds at time T when *Pat* stops being *Phy*'s patient for the first time during the current hospital visit.

```
1  ant(si(shareAut,Phy,Pat,T1),T2) :-
     becomePatient(Phy, Pat, T1),
     remainPatient(Phy, Pat, T1, T2).
   con(si(shareAut,Phy,Pat,T1),T2) :-
5    share_PHI_with(Phy, Pat, P, T2),
     { famMemberR(Pat, P, T2);
       colleagueR(Phy, P, T2) } >= 1,
     ant(si(shareAut,Phy,Pat,T1),T1),
     patientR(Phy, Pat, T2),
10   remainPatient(Phy, Pat, T1, T2).
   con(si(shareAut,Phy,Pat,T),T) :-
     share_PHI_with(Phy, Pat, P, T),
     not patientR(Phy, Pat, T).
   deadCon(si(shareAut,Phy,Pat,T1),
     T2) :-
15   ant(si(shareAut,Phy,Pat,T1),T1),
     remainPatient(Phy, Pat, T1, T2),
     not patientR(Phy, Pat, T2+1).
```

Listing 6: Definition of shareAut

The life cycle of an authorization is defined in Listing 7. The detachment is the same as that of a commitment instance so I did not include here for simplicity. The satisfaction and violation are simple. Whether the consequent holds or not does not matter for a detached authorization instance. Hence, a detached authorization instance will become satisfied when the deadline condition holds. On the other hand, a non-detached authorization instance becomes violated at the time instant when the consequent holds.

```
1  becomesSatisfied(Si, T) :-
     detached(Si, T),
     deadCon(Si, T).
   becomesViolated(si(S,Sub,Obj,Td),
     T) :-
5    con(si(S, Sub, Obj, Td), T),
     not detached(si(S,Sub,Obj,Td),
       T).
```

Listing 7: Life Cycle of Authorization

The following scenario (Listing 8) is used to test the definition of shareAut and an authorization's life cycle. The result is that 1) the instance si(shareAut,bob,alice,5) becomes satisfied at 10; 2) si(shareAut,bob,alice,12) becomes satisfied at 14; 3) si(shareAut,bob,alice,11) becomes violated at 11.

```
1  principal(bob, 1..15).
   principal(alice, 1..15).
   principal(dorthy, 1..15).
   principal(carol, 1..15).
5
   physician(bob, 1..15).
   child(alice, 1..15).

   patientR(bob, alice, 5..10).
10 patientR(bob, alice, 12..14).
   colleagueR(bob, dorthy, 1..15).
   colleagueR(dorthy, bob, 1..15).
   famMemberR(alice, carol, 1..15).
   famMemberR(carol, alice, 1..15).
15
   share_PHI_with(bob, alice, dorthy,
     9).
   share_PHI_with(bob, alice, carol,
     10).
   share_PHI_with(bob, alice, carol,
     11).
   share_PHI_with(bob, alice, carol,
     13).
20
   time(15).
```

Listing 8: Life Cycle of Authorization

4 Prohibition Schema

A prohibition schema means that when the antecedent holds, the subject is prohibited to bring about the consequent.

Example 3 (sharePro): A physician is prohibited to share any of his current or previous patients' PHI.

The difference between sharePro and shareAut is that the prohibition of sharing PHI is still in effect even after a principal stops being the physician's patient, which is reflected by the definition of sharePro's antecedent (Listing 9). $notPatientBefore(Phy, Pat, T)$ means Pat is not Phy's patient at any time $T' < T$. $beenPatientBefore(Phy, Pat, T)$ means Pat is Phy's patient at some time $T' < T$. Hence, the antecedent predicate says once Pat becomes Phy's patient for the first time, the antecedent starts to hold. As for the consequent, we only care about the instances whose antecedent is true, i.e. whose object has been the patient of the subject at some point. And the consequent holds when the physician publish a patient's PHI online. Finally, the deadline condition holds till the end of the current timeline.

```
ant(si(sharePro,Phy,Pat,T1),T2) :-
  notPatientBefore(Phy, Pat, T1),
  patientR(Phy, Pat, T1),
  beenPatientBefore(Phy, Pat, T2).
con(si(sharePro,Phy,Pat,T1),T2) :-
  publish_PHI_online(Phy,Pat,T2),
  T2 >= T1,
  ant(si(sharePro,Phy,Pat,T1),T1).
deadCon(si(sharePro, Phy, Pat,T1),
    T2) :-
  ant(si(sharePro,Phy,Pat,T1),T2),
  time(T2).
```

Listing 9: Definition of sharePro

A detached prohibition instance becomes satisfied at the time T when its deadline condition holds at T and its consequent has never been true between the time at which it becomes detached and T (Listing 10). Further, it becomes violated at T which is between its detachment time and deadline time, if its consequent holds at T.

```
becomesSatisfied(Si, T2) :-
  becomesDetached(Si, T1),
  deadCon(Si, T2),
  not conTrueBetween(Si, T1, T2).
becomesViolated(Si, T) :-
  becomesDetached(Si, T1),
  deadCon(Si, T2),
  con(Si, T), detached(Si, T),
  T1 <= T, T <= T2,
  not conTrueBetween(Si, T1, T-1).
```

Listing 10: Life Cycle of Authorization