CM30174 + CM50206
Introduction to Intelligent Agents
Semester 1, 2008-09

Marina De Vos, Julian Padget

Modelling Institutions / 20081205 / version 0.4

December 5, 2008

Authors/Credits for this lecture

- Primary author: Marina De Vos.
- Material sourced from Owen Cliffe, Marina De Vos and Julian Padget: “Answer Set Programming for Representing and Reasoning about Virtual Institutions” and “Specifying and Reasoning about Multiple Institutions” [1, 2].

Characteristics of virtual institutions

Who: actors
- internal
- external

+

What:
- actors may say
- actors may do

When:
- a communication may occur
- an action may take place

Where:
- an actor may go
- an action may take place

+

State:
- records
- obligations

Observable actions of agents change the institution’s state

A Norm-driven approach

- A top-down approach to institutional modelling views an institution as:
  - A set of *institutional states* that evolve
  - in response to *institutional events*.
  - where an institutional state is a set of *institutional facts*
- These are the *observables* identified earlier

A Norm-driven approach

- How are institutional facts created?
  - Searle [3] identifies two kinds of facts
    - Brute facts that are observable in the physical world
    - and institutional facts that are neither observable, nor have any meaning outside their institution
  - Institutional facts are created by an action in the physical world that counts as taking that action in the institutional world.
  - Thus the observation of an agent action can lead to the creation of an institutional fact within the institution in which the agent is participating.

Social States

Several types of institutional facts are considered:

- **Permission:** An agent's Ability to carry out some action without sanction.
- **Obligation:** Facts stating that an agent is obliged to have done some action before some deadline.
- **Institutional Power:** (after Jones & Sergot) institutional facts describe an agent's capacity to affect the social state by performing *meaningful* institutional actions.
- **Domain Facts:** Those relating internally to the institution in question. (i.e. marina Ows X)

Events

- Account for (possible) changes in state
- May be:
  - Domain Events (exogenous): observed from the environment.
  - Institutionally generated (internal): generated by the institution
- Events may generate other events: Conventional generation.

Conventional Generation

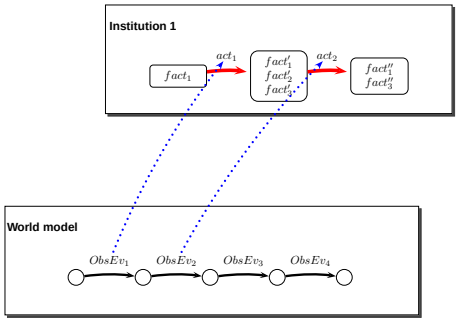
Origins in theory of action (Goldman, Searle, Jones & Sergot)

- "Doing X [in environment A] counts as doing Y [in environment B] iff Z"
- Allows us to abstract institutional actions from real world ones, i.e.:
  - "Saying 'aye' in an auction counts as an offer to buy some goods at the current price"
  - "Clicking 'buy it now' counts as an offer to buy some goods at a given price on amazon"
- Generation is assumed to be atomic (i.e. generated events occur concurrently with events which generate them)

Regulation

- Not all sequences of action are desirable
- We specify regulatory rules to identify "bad" paths of events
- Two regulatory mechanisms are considered:
  - **Obligation:** "You should do X before Y happens"
  - **Permission:** "You should not do X"
- **Violations:** When the above rules are broken *violation events* are generated for:
  - The failure to perform an action before a deadline.
  - Performing an action without permission.

Specification Model



Formal Specification

Definition

Institutions: $\mathcal{I} := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$

- $\mathcal{E} = \mathcal{E}_{obs} \cup \mathcal{E}_{inst}$ with $\mathcal{E}_{inst} = \mathcal{E}_{instact} \cup \mathcal{E}_{viol}$
- $\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$
- $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ with $C(X, e) = (\mathcal{C}^{\uparrow}(X, e), \mathcal{C}^{\downarrow}(X, e))$
- $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$
- Δ
- State Formula: $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$

where institutional facts ($\mathcal{F}$) are defined in terms of power ($\mathcal{W}$), permission ($\mathcal{P}$), obligation ($\mathcal{O}$) and domain facts ($\mathcal{D}$) and where $\mathcal{C}^{\uparrow}(X, e)$ and $\mathcal{C}^{\downarrow}(X, e)$ resp., contain those fluents which are *initiated/terminated* by the event e in any state matching X

Semantics

- Event Generation.
- Fluent Initiation.
- Fluent Termination
- State Transformation
- Traces

Event Generation

GR(S, E) generates

Intuition

1. Events that are generated remain generated
2. Empowered Events which are generated from conventional generation with conditions matching S
3. Violations generated from conventional generation matching the current state
4. Violations that result from events which were not permitted
5. Violations from obligations for which the deadline has expired

Event Generation

Definition

$$\begin{aligned}\mathrm{GR}(S,E) = \{e \in \mathcal{E} \mid\ & e \in E && \textbf{or}\\ & \exists\, e' \in E, \phi \in \mathcal{X}, e \in G(\phi, e') \cdot S \models \mathrm{pow}(e) \wedge S \models \phi && \textbf{or}\\ & \exists\, e' \in E, \phi \in \mathcal{X}, e \in G(\phi, e') \cdot e \in \mathcal{E}_{viol} \wedge S \models \phi && \textbf{or}\\ & \exists\, e' \in E \cdot e = viol(e'), S \models \neg \mathrm{perm}(e') && \textbf{or}\\ & \exists\, e' \in \mathcal{E}, d \in E \cdot S \models \mathrm{obl}(e', d, e)\}\end{aligned}$$

Initiation

Intuition

Fluents are initiated if: If a certain event in the current environment triggers the consequence relation to initiate this fluent

Definition

$$\mathrm{INIT}(S, e_{obs}) = \{p \in \mathcal{F} \mid \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), X \in \mathcal{X} \cdot p \in \mathcal{C}^{\uparrow}(X, e) \wedge S \models X\}$$

Termination

Intuition

Fluents are terminated if:

- A certain event in the current environment triggers the consequence relation to terminate this fluent, or
- The deadline or the event of an obligation occurred

Definition

$$\begin{aligned}\mathrm{TERM}(S, e_{obs}) = \{p \in \mathcal{F} \mid\ & \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}), X \in \mathcal{X} \cdot p \in \mathcal{C}^{\downarrow}(X, e), S \models X && \textbf{or}\\ & p = \mathrm{obl}(e, d, v) \wedge p \in S \wedge e \in \mathrm{GR}^{\omega}(S, \{e_{obs}\}) && \textbf{or}\\ & p = \mathrm{obl}(e, d, v) \wedge p \in S \wedge d \in \mathrm{GR}^{\omega}(S, \{e_{obs}\})\}\end{aligned}$$

State Transitions

Intuition

The new states consists of the fluents of the old state which were not terminated plus all the newly initiated fluents.

Definition

We define the transition function $\mathrm{TR} : \Sigma \times \mathcal{E}_{obs} \rightarrow \Sigma$ as:

$$\begin{aligned}\mathrm{TR}(S, e_{obs}) = \{p \in \mathcal{F} \mid\ & p \in S, p \notin \mathrm{TERM}(S, e_{obs}) && \textbf{or}\\ & p \in \mathrm{INIT}(S, e_{obs})\}\end{aligned}$$

Traces

- An ordered trace is defined as a sequence of observable events

$$\langle e_0, e_1, \ldots, e_n \rangle \quad e_i \in \mathcal{E}_{obs}, 0 \leq i \leq n$$

- The evaluation of an ordered trace for a given starting state S_0 is a sequence $\langle S_0, S_1, \ldots S_{n+1} \rangle$ such that $S_{i+1} = \mathrm{TR}(S_i, e_i)$
- Ordered traces and their evaluations allow us to monitor or investigate the evolution of an institution over time. They provide us with the data necessary to answer most queries one might have about a certain institution.

An example

Example

A country is constantly swinging between war and peace with its neighbour. The countries have agreed that when they are at peace, a citizen of the first shooting a citizen of the second counts as murder, when they are at war and a citizen has been conscripted into the army it is permitted to shoot. When one country is provoked, it is obliged to start war first before it is allowed to shoot.

An example

Example

$$\begin{aligned}
\mathcal{E}_{obs} &= \{\texttt{shoot}, \texttt{startwar}, \texttt{declaretruce}, \texttt{callup}, \texttt{provoke}\} && (1)\\
\mathcal{E}_{instact} &= \{\texttt{conscript}, \texttt{murder}\} && (2)\\
\mathcal{E}_{viol} &= \{\mathsf{viol}(\texttt{shoot}), \mathsf{viol}(\texttt{startwar}), \mathsf{viol}(\texttt{declaretruce}),\\
&\quad \mathsf{viol}(\texttt{callup}), \mathsf{viol}(\texttt{provoke}), \mathsf{viol}(\texttt{conscript}), \mathsf{viol}(\texttt{murder})\} && (3)\\
\mathcal{D} &= \{\texttt{atwar}\} && (4)\\
\mathcal{W} &= \{\mathsf{pow}(\texttt{conscript}), \mathsf{pow}(\texttt{murder})\} && (5)\\
\mathcal{P} &= \{\mathsf{perm}(\texttt{shoot}), \mathsf{perm}(\texttt{startwar}), \mathsf{perm}(\texttt{declaretruce}),\\
&\quad \mathsf{perm}(\texttt{callup}), \mathsf{perm}(\texttt{provoke}), \mathsf{perm}(\texttt{conscript}), \mathsf{perm}(\texttt{murder})\} && (6)\\
\mathcal{O} &= \{\mathsf{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} && (7)
\end{aligned}$$

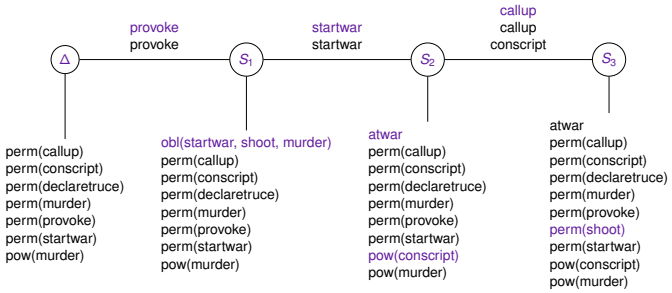
An example

Example

$$\begin{aligned}
\mathcal{C}^{\uparrow}(\mathcal{X},\mathcal{E}): \quad \langle\{\neg \mathit{atwar}\}, \texttt{startwar}\rangle &\rightarrow \{\mathit{atwar}\} && (8)\\
\langle\{\neg \mathit{atwar}\}, \texttt{provoke}\rangle &\rightarrow \{\mathsf{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} && (9)\\
\langle\emptyset, \texttt{conscript}\rangle &\rightarrow \{\mathsf{perm}(\mathit{shoot})\} && (10)\\
\langle\emptyset, \texttt{startwar}\rangle &\rightarrow \{\mathsf{pow}(\mathit{conscript})\} && (11)\\
\mathcal{C}^{\downarrow}(\mathcal{X},\mathcal{E}): \quad \langle\{\mathit{atwar}\}, \texttt{declaretruce}\rangle &\rightarrow \{\mathit{atwar}\} && (12)\\
\langle\emptyset, \texttt{declaretruce}\rangle &\rightarrow \{\mathsf{perm}(\mathit{shoot})\} && (13)\\
\langle\emptyset, \texttt{declaretruce}\rangle &\rightarrow \{\mathsf{pow}(\mathit{conscript})\} && (14)\\
\mathcal{G}(\mathcal{X},\mathcal{E}): \quad \langle\emptyset, \texttt{callup}\rangle &\rightarrow \{\mathit{conscript}\} && (15)\\
\langle\emptyset, \mathsf{viol}(\texttt{shoot})\rangle &\rightarrow \{\mathit{murder}\} && (16)
\end{aligned}$$

$$\begin{aligned}
S_0 &= \{\mathsf{perm}(\mathit{callup}), \mathsf{perm}(\mathit{startwar}), \mathsf{perm}(\mathit{conscript}), \mathsf{perm}(\mathit{provoke}),\\
&\quad \mathsf{pow}(\mathit{murder}), \mathsf{perm}(\mathit{murder})\} && (17)
\end{aligned}$$

Example



Reasoning in the formal model

- The formal spefications allows us to describe all the components of an institution in a very concise and precise way
- however, it comes with little functionality to validate or reason about the institution
- unless we want to do everything manually
- so we need a computational tools

Computational Tools

- Based on logic to assure verifiability
- Expressive
- Straightforward mapping
- Queries
- We use answer set programming for our modelling
  - Sound grounding in logic - verifiable
  - Specification equals implementation
  - Intuitive
  - Very expressive

Mapping Institutions to ASP

- Smodels syntax (as seen earlier)
- Time instances to indicate state transitions
- Atoms:
  - `evtype(E,T)` describes the type of an event
  - `instant(I)` denote time instances
  - `final(I)` denotes the last time instance in a trace
  - `before(I1,I2)` and `next(I1,I2)` denote time order
  - `occurred(E,I)` indicates E happened at time I
  - `observed(E,I)` indicates E was observed at time I
  - `holdsat(P,I)` indicates that P holds at time I
  - `initiated(P,I)` and `terminated(P,I)` indicate that P is initiated/terminated at time I

Parts of the Mapping

- Each mapping for institution $\mathcal{I}$ consists of two parts
  - P_{base}: institution independent
    - responsible for the occurrence of observed events
    - deals with obligations
    - assures inertia
  - $P^*_{\mathcal{I}}$ specific for the institution
    - event generation
    - state transition

The institution program P_{base} (I)

```
occurred(E,I)  ← observed(E,I).
holdsat(P,I2)  ← holdsat(P,I1), not terminated(P,I1),
                 next(I1,I2), instant(I1;I2).
holdsat(P,I2)  ← initiated(P,I1),
                 next(I1,I2), instant(I1;I2).
occurred(viol(E),I) ← occurred(E,I),
                 not holdsat(perm(E),I),
                 event(E), event(viol(E)), instant(I).
occurred(V,I)  ← holdsat(obl(E,D,V),I), occurred(D,I),
                 event(E;D;V), instant(I).
terminated(obl(E,D,V),I) ← occurred(E,I),
                 holdsat(obl(E,D,V),I),
                 event(E;D;V), instant(I).
terminated(obl(E,D,V),I) ← occurred(D,I),
                 holdsat(obl(E,D,V),I),
                 event(E;D;V), instant(I).
```

The institution program P_{base} (II)

To constrain the answer set to those containing observable traces we add the following rules to P_{base}:

$$
\begin{array}{rcl}
\{\texttt{observed(E, I)}\} & \leftarrow & \texttt{evtype(E, obs), event(E), instant(I),} \ \textbf{not} \ \texttt{final(I).} \\
\texttt{ev(I)} & \leftarrow & \texttt{observed(E, I), event(E), instant(I).} \\
 & \leftarrow & \textbf{not} \ \texttt{ev(I), instant(I),} \ \textbf{not} \ \texttt{final(I).} \\
 & \leftarrow & \texttt{observed(E1, I), observed(E2, I), E1 != E2, instant(I),} \\
 & & \texttt{event(E1), event(E2).}
\end{array}
$$

Thus an observable event occurs at each time instant, while the last constraint ensures that each answer set has only one observable event at any time instant.

A Shorthand

- $EX(\_, I)$ to denote the translation of expression $X \in \mathcal{X}$ into the body of an ASP rule referring to time I.
- $EX(x_1 \wedge x_2 \wedge \ldots x_n, I)$ is translated as $\texttt{EX(x}_1\texttt{, I)}, \texttt{EX(x}_2\texttt{, I)}, \ldots, \texttt{EX(x}_n\texttt{, I)}$.
- $EX(\neg p, I)$ becomes **not** $EX(p, I)$
- $EX(p, I)$ is translated as $\texttt{holdsat(p, I)}$.
- Thus $EX(perm(callup), \neg perm(murder), pow(murder), I)$ becomes $\texttt{holdsat(perm(callup)),}$ **not** $\texttt{holdsat(perm(murder)), holdsat(pow(murder))}$

The Institution Dependent Part $P^*_{\mathcal{I}}$

$$
\begin{array}{lcl}
p \in \mathcal{F} & \Leftrightarrow & \texttt{ifluent(p).} \\
e \in \mathcal{E} & \Leftrightarrow & \texttt{event(e).} \\
e \in \mathcal{E}_{obs} & \Leftrightarrow & \texttt{evtype(e, obs).} \\
e \in \mathcal{E}_{instact} & \Leftrightarrow & \texttt{evtype(e, act).} \\
e \in \mathcal{E}_{viol} & \Leftrightarrow & \texttt{evtype(e, viol).} \\
\mathcal{C}^{\uparrow}(X, e) = P & \Leftrightarrow & \forall p \in P \cdot \texttt{initiated(p, I)} \leftarrow \texttt{occurred(e, I)}, EX(X, I). \\
\mathcal{C}^{\downarrow}(X, e) = P & \Leftrightarrow & \forall p \in P \cdot \texttt{terminated(p, I)} \leftarrow \texttt{occurred(e, I)}, EX(X, I). \\
\mathcal{G}(X, e) = E & \Leftrightarrow & g \in E, \texttt{occurred(g, I)} \leftarrow \texttt{occurred(e, I)}, \\
 & & \quad \texttt{holdsat(pow(e), I)}, EX(X, I). \\
p \in S_0 & \Leftrightarrow & \texttt{holdsat(p, i}_0\texttt{).}
\end{array}
$$

Soundness and Completeness

- We still need to initialise the ASP with time
- We refer to this program as P^n:

$$0 < k < n \ : \ \begin{array}{l} \texttt{instant}(\texttt{i}_\texttt{k}). \\ \texttt{next}(\texttt{i}_\texttt{k}, \texttt{i}_{\texttt{k+1}}). \\ \texttt{final}(\texttt{i}_\texttt{n}). \end{array}$$

- Together P_{base}, $P^*_{\mathcal{I}}$ and P^n generate $P^n_{\mathcal{I}}$

Theorem

Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution with $P^n_{\mathcal{I}}$ its corresponding answer set program. Then, a one-to-one mapping exists between the ordered event traces of length n and the answer sets of $P^n_{\mathcal{I}}$.

War in ASP (1)

```
ifluent(atwar).
```

```
ifluent(obl(startwar,shoot,murder)).
```

```
event(shoot).
event(startwar).
event(declaretruce).
event(callup).
event(conscript).
event(murder).
event(provoke).
event(viol(shoot)).
event(viol(startwar)).
event(viol(declaretruce)).
event(viol(callup)).
event(viol(conscript)).
event(viol(provoke)).
```

```
evtype(shoot,obs).
evtype(startwar,obs).
evtype(declaretruce,obs).
evtype(callup,obs).
evtype(conscript,inst).
evtype(murder,inst).
evtype(provoke,obs).
evtype(viol(shoot),viol).
evtype(viol(startwar),viol).
evtype(viol(declaretruce),viol).
evtype(viol(callup),viol).
evtype(viol(conscript),viol).
evtype(viol(murder),viol).
evtype(viol(provoke),viol).
```


War in ASP (2)

```
initiated(obl(startwar,shoot,murder),I) ← occurred(provoke,I),instant(I),
                                          not holdsat(atwar,I).
initiated(atwar,I) ← occurred(startwar,I),instant(I),
                     not holdsat(atwar,I).
initiated(perm(shoot),I) ← occurred(conscript,I),instant(I).
initiated(pow(conscript),I) ← occurred(startwar,I),instant(I).
```

```
terminated(atwar,I) ← occurred(declaretruce,I),instant(I),
                      holdsat(atwar,I).
terminated(perm(shoot),I) ← occurred(declaretruce,I),instant(I).
terminated(pow(conscript),I) ← occurred(declaretruce,I),instant(I).
```

War in ASP (3)

```
occurred(conscript, I) ← occurred(callup, I), instant(I),
                         holdsat(pow(conscript), I).
occurred(murder, I)    ← occurred(viol(shoot), I), instant(I).
```

```
instant(i0; i1; i2; i3).
init(i0).
next(i0, i1).
next(i1, i2).
next(i2, i3).
final(i3).
```

```
holdsat(perm(callup), i0).
holdsat(perm(startwar), i0).
holdsat(perm(conscript), i0).
holdsat(perm(declaretruce), i0).
holdsat(perm(murder), i0).
holdsat(perm(provoke), i0).
holdsat(pow(murder)), i0.
```


Queries

- Given an institutional specification in ASP, queries are possible:
  - Given some known initial state, and a complete trace of events, what is the current social state?
  - Given partial information about the initial and/or current state what are the possible sequences of events which led us to this state.
- Two rules need to be added:
  - one to represent the query
  - one to indicate to the solver that we are only interested in those ordered traces that satisfies the condition

Example

```
condition←holdsat(obl(startwar, shoot, murder), I), instant(I).
compute all {condition}.
```


Exercise

Example

Ownership is a well know and simple institution. Objects can change from one person to the next when the former owns the object.
How would you formalise this institution?

- What are the facts?
- What are events?
- How does event generation look like?
- What are the consequences?

The Dutch Auction

- One agents acts as auctioneer
- One or more agents play the bidders
- The purpose of the protocol as a whole is either to determine a winning bidder and a valuation for a particular item on sale, or to establish that no bidders wish to purchase the item.

The Protocol (I)

Definition

- Round starts: Auctioneer selects a price for the item and informs each of the bidders present of the starting price. The auctioneer then waits for a given period of time for bidders to respond.
- Upon receipt of the starting price, each bidder has the choice as to whether to send a message indicating their desire to bid on the item at that price, or to send no message indicating that they do not wish to bid on the item.
- At the end of the prescribed period of time, if the auctioneer has received a single bid from a given agent, then the auctioneer is obliged to inform each of the participating agents that this agent has won the auction.

The Protocol (II)

Definition

- If no bids are received at the end of the prescribed period of time, the auctioneer must inform each of the participants that the item has not been sold.
- If more than one bid was received then the auctioneer must inform each agent that a conflict has occurred.
- In the case where the item is sold or unsold, the protocol is finished.
- In the case where a conflict occurs then the auctioneer must re-open the bidding and start the round again in order to resolve the conflict.

DAR InstAL(I)

Example

```
institution dutch;
type Bidder;
type Auct;

create event createdar;

exogenous event priceto;
exogenous event bidto;
exogenous event desto;

exogenous event annprice(Auct,Bidder);
exogenous event annbid(Bidder,Auct);
exogenous event annconf(Auct,Bidder);
exogenous event annsold(Auct,Bidder);
exogenous event annunsold(Auct,Bidder);

inst event pricedl;
inst event biddl;
inst event desdl;
inst event desdl;
```


DAR InstAL(II)

Example

```
inst event price(Auct,Bidder);
inst event bid(Bidder,Auct);
inst event conf(Auct,Bidder);
inst event sold(Auct,Bidder);
inst event unsold(Auct,Bidder);

dest event badgov;
dest event finished;

inst event alerted(Bidder);

fluent onlybidder(Bidder);
fluent havebid;
fluent conflict;

initially pow(price(A,B)), perm(price(A,B)),
    perm(annprice(A,B)),perm(badgov),pow(badgov),
    perm(pricedl),pow(pricedl), perm(priceto),
    perm(biddl),perm(bidto),perm(desto);
```


DAR InstAL(III)

Example

```
-(Phase 1:  pricing) -
initially obl(price(A,B),pricedl,badgov);
annprice(A,B) generates price(A,B);
price(A,B) terminates pow(price(A,B));
price(A,B) initiates pow(bid(B,A)),perm(bid(B,A)),perm(annbid(B,A));

- (Phase 2:  bidding) -
annbid(A,B) generates bid(A,B);
bid(B,A) terminates pow(bid(B,A)),perm(bid(B,A)),perm(annbid(B,A));
bid(B,A) initiates havebid,onlybidder(B) if not havebid;
bid(B,A) terminates onlybidder(_) if havebid;
bid(B,A) initiates conflict if havebid;

s - (Phase 3:  Resolution) -
annsold(A,B) generates sold(A,B);
annunsold(A,B) generates unsold(A,B);
annconf(A,B) generates conf(A,B);
biddl terminates pow(bid(B,A));
```

DAR InstAL(IV)

Example

```
- (Phase 3:  Resolution cont.)  -
biddl initiates pow(sold(A,B)),pow(unsold(A,B)),
    pow(conf(A,B)), pow(alerted(B)),perm(alerted(B));

biddl initiates perm(annunsold(A,B)),perm(unsold(A,B)),
    obl(unsold(A,B),desdl,badgov) if not havebid;
biddl initiates perm(annsold(A,B)),perm(sold(A,B)),
     obl(sold(A,B), desdl, badgov) if havebid, not conflict;
biddl initiates perm(annconf(A,B)),perm(conf(A,B)),
     obl(conf(A,B), desdl, badgov) if havebid, conflict;
unsold(A,B) generates alerted(B);
sold(A,B) generates alerted(B);
conf(A,B) generates alerted(B);
alerted(B) terminates pow(unsold(A,B)), perm(unsold(A,B)),
    pow(sold(A,B)), pow(conf(A,B)), pow(alerted(B)),
    perm(sold(A,B)), perm(conf(A,B)), perm(alerted(B)),
    perm(annconf(A,B)),perm(annsold(A,B)),perm(annunsold(A,B));

desdl generates finished if not conflict;
```


DAR InstAL(V)

Example

```
- (Phase 3:  Resolution cont.)  -
desdl terminates havebid,conflict,perm(annconf(A,B));
desdl initiates pow(price(A,B)), perm(price(A,B)),
     perm(annprice(A,B)), perm(pricedl),pow(pricedl),
     obl(price(A,B),pricedl,badgov) if conflict;

priceto generates pricedl;
pricedl terminates pow(pricedl);
pricedl initiates pow(biddl);

bidto generates biddl;
biddl terminates pow(biddl);
biddl initiates pow(desdl);

desto generates desdl;
desdl terminates pow(desdl);
```

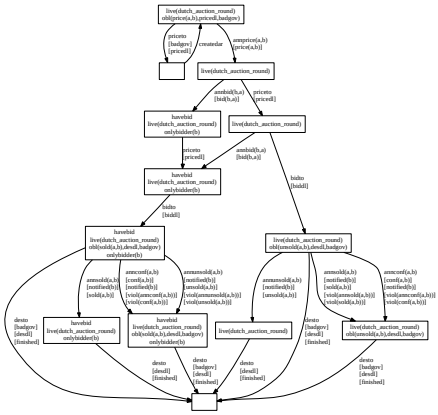

The DAR Scenes

- The DAR protocol is governed by the deadlines InstALpricedl, biddl and InstALdesdl.
- When the deadline event occurs, auctioneer and bidder are given different powers and permissions that allow the protocol to proceed to a different scene
- For example, InstALpricedl announces the end pricing scenes and provides the bidders with the power and permission to start bidding
- InstALdesdl is special in the sense that it either marks the end of the protocol (bidding was successfull) or has to start all over again.

Verification

- Just add the time frame
- and if requested a query program
- Run the solver and obtain the answer sets
- By varying the time frame you obtain all the states of the the protocol

The DAR Protocol as Finite State Machine



Summary

- Agent Societies: case for the norm-regulated agent and for an agent being governed by multiple interacting institutions
- Answer Set Programming: a logic programming paradigm that supports the definition of domain-oriented models, checking and querying
- Single Institutions: a trace-based formalization of a single institution in an executable framework
- Modelling Behaviour: a demonstration of the application of the formalization to some familiar scenarios

O. Cliffe, M. De Vos, and J. Padget.
Answer set programming for representing and reasoning about virtual institutions.
In *Computational Logic for Multi-Agents (CLIMA VII)*, Hakodate, Japan, May 2006.

O. Cliffe, M. De Vos, and J. Padget.
Specifying and reasoning about multiple institutions.
In *Coordination, Organization, Institutions and Norms in Agent Systems (COIN'06)*, Hakodate, Japan, May 2006.

John R. Searle.
The Construction of Social Reality.
Allen Lane, The Penguin Press, 1995.