INTRODUCING EQUIBEL

An Implementation of Consistency-Based Belief Change

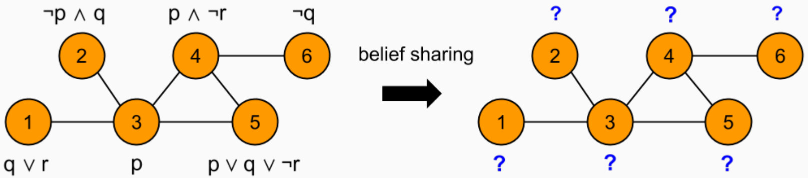
Paul Vicol[1] James Delgrande[1] Torsten Schaub[2]

LPNMR - September 28, 2015

[1]Simon Fraser University

[2]University of Potsdam

- We have a network of agents
- Each agent has some initial beliefs about the state of the world
- Agents communicate and share information
- **Goal:** Determine what each agent believes after learning as much as possible from other agents
- **How do we do this?**

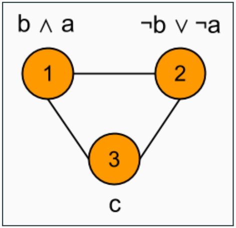


Original Graph G

Completion of G

Example: Drones looking for people in a disaster site

- Each drone has an initial belief:
  - Drone 1 believes that there is a person in the bookstore, and one in the atrium: $b \wedge a$
  - Drone 2 believes that there *cannot* be missing people in *both* the atrium *and* the bookstore: $\neg b \vee \neg a$
  - Drone 3 just believes that there is a person in the cafeteria: c



- The drones communicate, and learn from one another
  - Each drone is willing to incorporate new information that *does not conflict with its initial beliefs*

THE CONSISTENCY-BASED FRAMEWORK FOR BELIEF CHANGE

- An agent starts out with initial beliefs that it does not want to give up, and then includes as much information as consistently possible from other agents
- We want to determine what *pieces of information* an agent can incorporate from others
- **How is this done?**
  - Agent i expresses its beliefs in a language $\mathcal{L}^i$ over superscripted atoms $\mathcal{P}^i = \{p^i, q^i, r^i, \ldots\}$ (i.e. agent 1 believes $p^1 \wedge \neg q^1$)
  - We "force" the *languages* used by adjacent agents to agree on the truth values of corresponding atoms *as much as consistently possible*
  - This yields one or more maximal sets of equivalences, EQ, between atoms in the languages of adjacent agents
  - These equivalences provide a means to *consistently translate* information from one agent to another

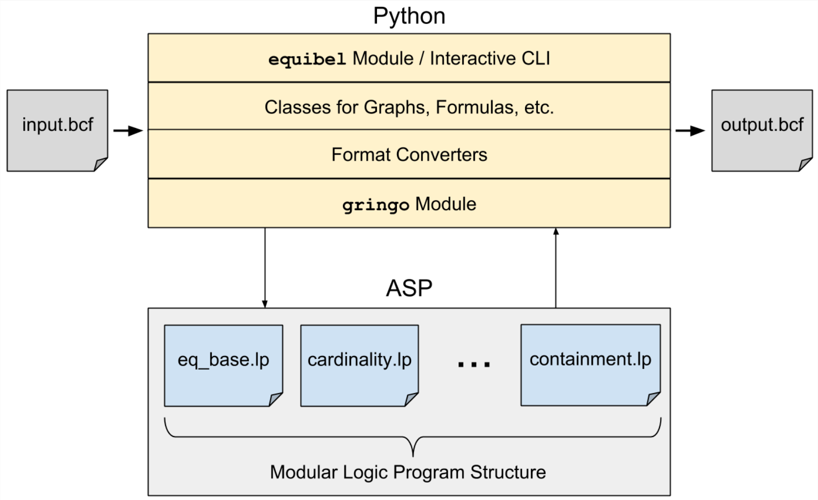
EQUIBEL OVERVIEW

- **Purpose:** To make it easy for students and researchers to experiment with belief change
- Equibel is an implementation of the consistency-based framework, in ASP and Python
- Allows users to *simulate* belief sharing in arbitrary networks of agents
  - Users create a graph and assign formulas to nodes
- Supports standard belief change operations like revision and merging by automatically constructing *implicit graph topologies*
  - Users specify a set of formulas and an operation to be performed
  - Behind the scenes, Equibel constructs a graph, finds the completion, and returns only the relevant formulas

EQUIBEL ARCHITECTURE

- The main operation performed by Equibel is finding the completion of a *G*-scenario
- The steps to find the completion are:
    1. Find maximal sets of equivalences between atoms of adjacent agents
    2. Translate beliefs between the languages of adjacent agents
    3. Combine beliefs resulting from different maximal equivalence sets
- Two architectural layers:
    - The ASP layer performs the core maximization procedure
    - The Python layer post-processes answer sets and provides programmatic and interactive interfaces

EQUIBEL SYSTEM DESIGN



ENCODING A BELIEF CHANGE SCENARIO IN ASP

- The graph structure is encoded using `node/1` and `edge/2`, and formulas are associated with nodes using `formula/2`
- Formulas are created using `neg/1`, `and/2`, `or/2`, `implies/2`, and `iff/2`

Example

```
node(1). node(2). node(3). node(4).
edge(1,2). edge(1,3). edge(2,3). edge(2,4).
formula(1, and(p,q)).
formula(2, or(q,neg(r))).
formula(3, implies(and(p,neg(q)),neg(r))).
formula(4, p).
```

- First, we break down formulas into subformulas and extract atoms
- We generate candidate equivalences $p^x \equiv p^y$ with:

```
{ eq(P,X,Y) : atom(P), edge(X,Y), X < Y }.
```

- Then we attempt to assign truth values to the atoms at each node:

```
1 { tv(N,P,1) ; tv(N,P,0) } 1 :- atom(P), node(N).
```

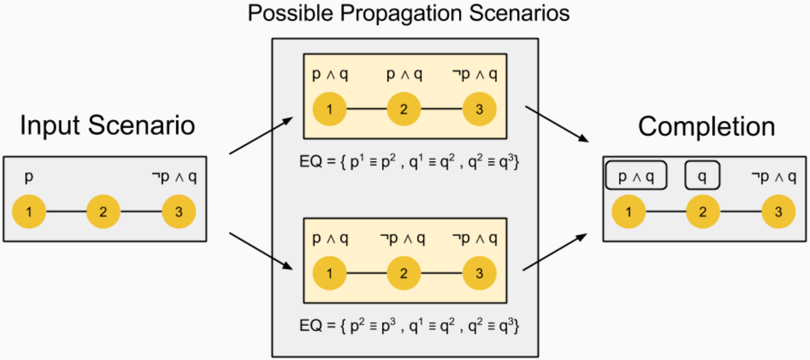
- Such that atoms p^x and p^y that participate in an equivalence $p^x \equiv p^y$ have the same truth value:

```
:- eq(P,X,Y), edge(X,Y), tv(X,P,V), tv(Y,P,W), V != W.
```

- We build up the original formulas from the bottom-up, checking satisfiability; all agents' original formulas must be satisfied:

```
:- formula(N,F), not sat(N,F).
```

- ASP gives us a collection of maximal equivalence sets
- In Python, we translate formulas between the languages of connected agents based on the EQ sets
- An agent may obtain different information from different EQ sets
  - Each EQ set represents an equally plausible way to share information
  - So we take the *disjunction* of beliefs obtained from different EQ sets



EQUIBEL IS EASY

- Equibel can be used interactively, by invoking the `equibel` prompt:

```
equibel (g) > add_nodes [1..4]
    nodes: [1, 2, 3, 4]
equibel (g) > add_edges [(1,2), (2,3), (3,4)]
    edges: 1 <-> 2   2 <-> 3   3 <-> 4
equibel (g) > add_formula 1 p & q
    node 1: q & p
equibel (g) > add_formula 4 ~p & r
    node 4: ~p & r
equibel (g) > completion
    node 1: q & p & r
    node 2: q & r
    node 3: q & r
    node 4: q & ~p & r
```

- The following script simulates belief sharing in the drone scenario:

```
import equibel
G = equibel.complete_graph(3)
G.add_formula(0, 'a & b')
G.add_formula(1, '~a | ~b')
G.add_formula(2, 'c')
R = equibel.completion(G)
print(R.formulas())
```

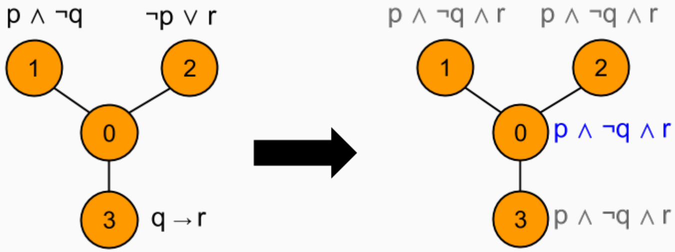
- `python drones.py`

```
{0: a & c & b, 1: c & ((a & ~b) | (~a & b)), 2: (a | b) & c}
```

- **Belief revision** = Incorporating a new belief α into a belief set K
- `equibel.revise(['p', 'q | ~r'], 'r')` constructs the graph:

- Agent 2 will incorporate as much information as possible from agent 1, while not giving up its initial belief
- The *revision* of $K = \{p, q \vee \neg r\}$ by $\alpha = r$ is the belief of agent 2 in the completion

- Two types of merging: projection-based and consensus-based
- `equibel.merge(['p&q', '~p|r', 'q->r'], type=equibel.PROJECTION)` constructs a star graph:



- The input formulas are *projected* onto the central node
- The result is the formula at the central node in the completion

Equibel

- Is a software system for working with equivalence-based belief change
- Simulates belief sharing in multi-agent scenarios
- Supports standard belief change operations (revision and merging) by constructing implicit graphs
- Provides a Python package, as well as an interactive prompt
- Is open source, hosted at `www.github.com/asteroidhouse/equibel`
- Is available on `PyPI`, so it can be installed using `pip`:

```
pip install equibel
```