ArgQL: A Declarative Language for Querying Argumentative Dialogues

RULEML+RR: INTERNATIONAL JOINT CONFERENCE ON RULES AND REASONING

DIMITRA ZOGRAFISTOU, GIORGOS FLOURIS, DIMITRIS PLEXOUSAKIS

FOUNDATION FOR RESEARCH AND TECHNOLOGY (FORTH)

JULY 2017

The big picture

- The recent growth of the **social** side of the web offers new incentive for users to:
  - Easily upload digital content
  - Communicate with each other
  - Express themselves
- ***Outcome***: online communities populated with tons of unstructured and unexploited data about:
  - Opinions and beliefs about political or social topics
  - Criticisms and consultations and
  - Reviews on products, services etc.
- ***Objective***: Development of methods to make data, created from users' participation in online communities, machine interpretable, retrievable and computable.

Role of Computational Argumentation

- Branch of Artificial Intelligence which studies the process of human reasoning while arguing and debating.

- Theoretical and computational models addressing miscellaneous issues like:
  - The process of resolving controversial positions through reasoning
  - Mental factors that affect that process like preferences, intents, beliefs, desires , trust, etc.
  - Evaluation of debates
  - Persuasiveness and impact of arguments

  Basic component is the *argument*.

- User created dialogues as an application field for argumentation models

Problem description

Need for mechanisms for navigation and information identification and extraction from a graph of interconnected arguments

ArgQL: a declarative query language designed on a data model for argumentation

Why is it important?

- Focus on understanding the information requirements when extracting information from dialogues.
- Familiar terminology to the community of argumentation

Why not using traditional query languages like SPARQL?

- We can use traditional languages for our purpose but …
- Argumentation defines distinct and solid concepts different than the triples of RDF e.g.
- Generates much simpler and descriptive queries for its purpose.

Similar approaches

No similar approach to a query language for arguments

Significant efforts towards the realization of a web of opinions

A number of tools that facilitate the participation in online debates.

- Argument mining from text (Natural Language Processing)
- Visualization (*e.g. Debate Graph, Argunet, Opinion Space etc*.)
- Reasoning (*e.g. Parmenides, Avicenna, etc.*)
- User engagement (*CreateDebate, Debate.org, etc.*)
- Storage and Searching engines (*AIFdb, ArgDF, DourcourseDB etc.*)

***AIF (Argument Interchange Format)*.** Interlingua that bridges arguments among the tools.

None of those tools defines a pure language for argumentation

- Application fields for traditional languages like SQL, SPARQL.

Argumentation data model (1/3)

$$L = \langle P, -, \simeq, \rightarrow \rangle$$

P : infinite set of propositions

$- \subseteq P \times P$: ***contrariness*** relation

- $(p_1, p_2) \in -$: p_1 in conflict with p_2

$\simeq \subseteq P \times P$: ***equivalence*** relation

- $(p_1, p_2) \in \simeq$: p_1 equivalent to p_2

$\rightarrow$: ***Inference*** as commonly used

- a, b, c $\rightarrow$ d : propositions a, b, c imply proposition d

Argumentation data model (2/3)

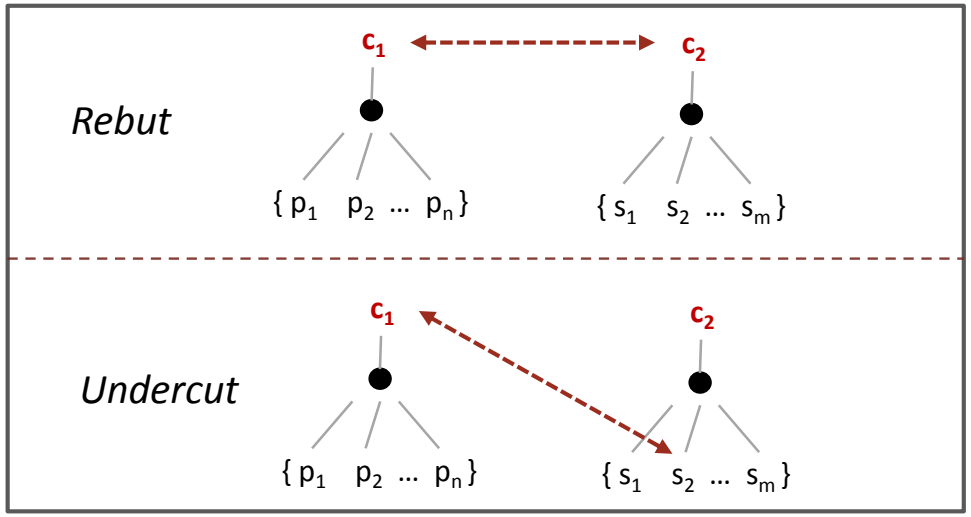
Arguments:

$$A = \langle pr, c \rangle$$

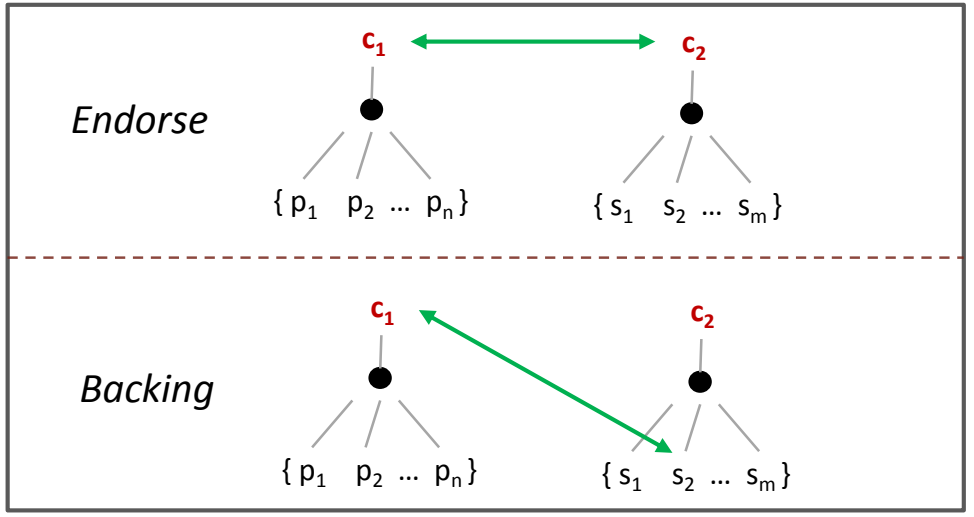
where $pr \subset P$, $c \in P$, $pr \rightarrow c$

Relations:

Attack



Support



: equivalence
: conflict

Argumentation data model (3/3)

Debate graph : **D = (A, R)**

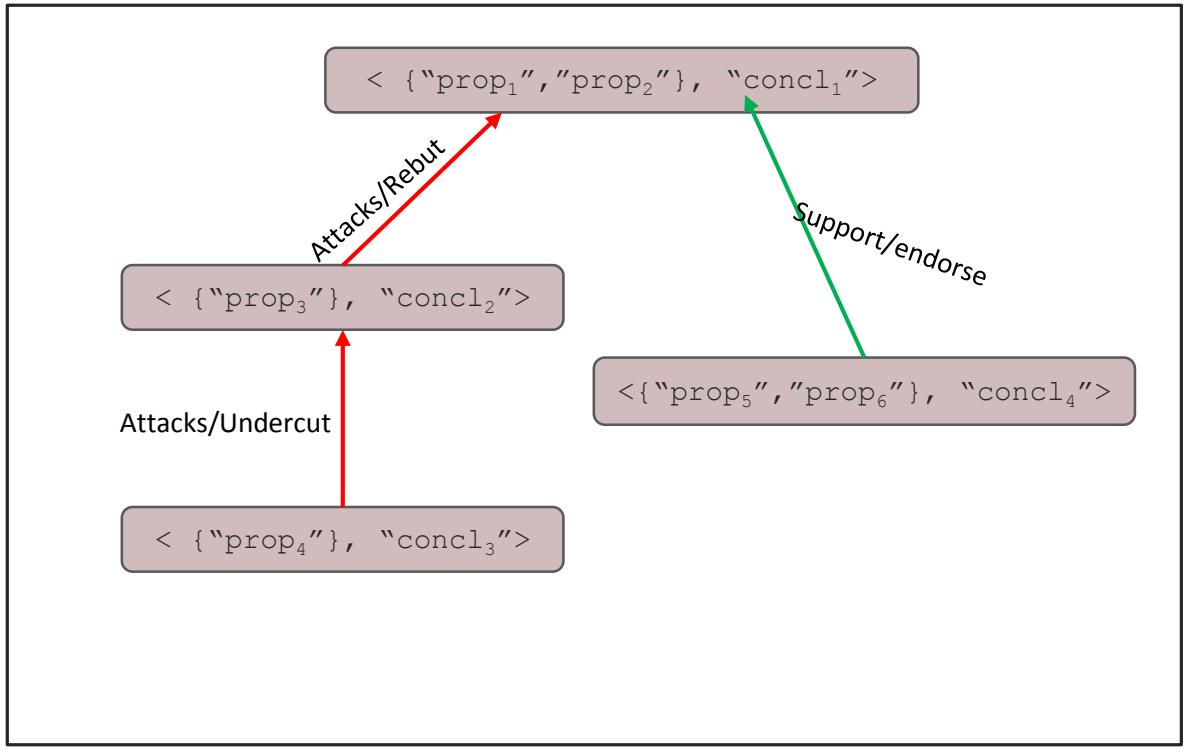
A : set of arguments

R⊆ $A \times A$: set of relations

Data

```
< {"prop1","prop2"}, "concl1">
<{"prop3"}, "concl2">
<{"prop4"} , "concl3">
<{"prop5", "prop6"} , "concl4">

"concl1" in_conflict "concl2"
"concl3" in_conflict "prop3"
"concl4" equivalent "concl1"
```



Query examples

1. Find arguments which "defend" (attack their attackers) all arguments with conclusion "$concl_1$".

```
match ?arg (attack/attack)+ <?pr, "concl1">
return ?arg
```

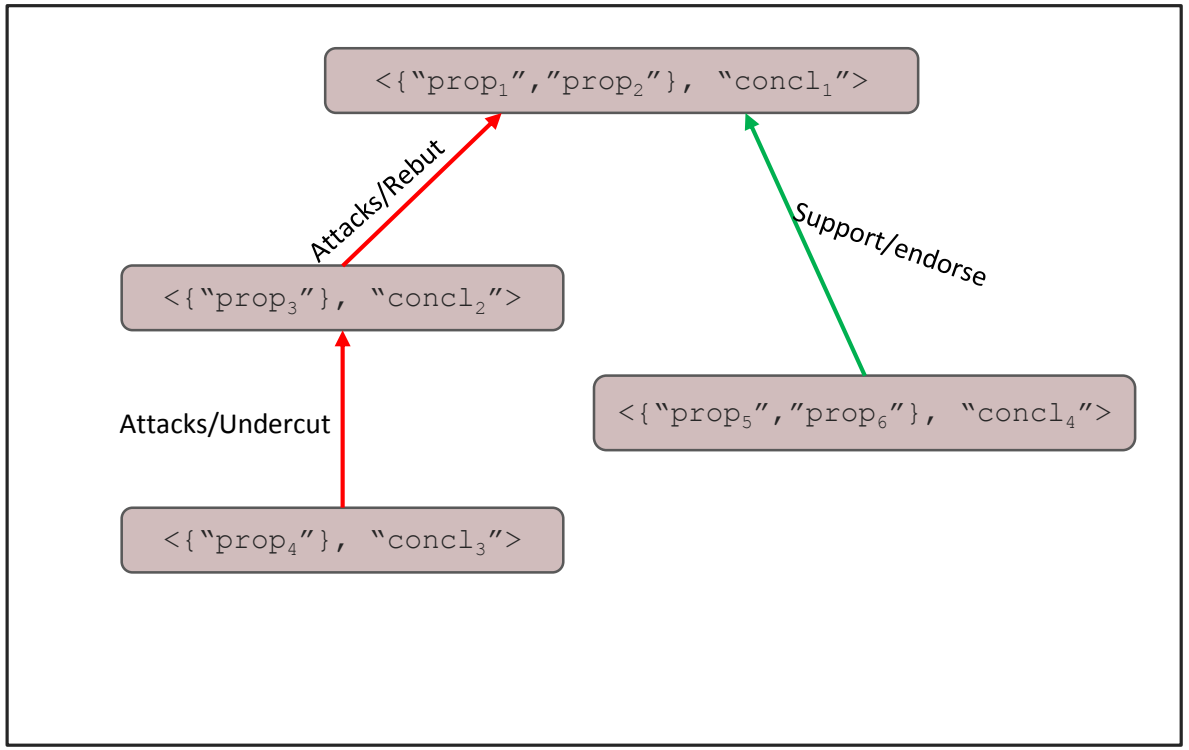
Results: <{"$prop_4$"}, "$concl_3$">

2. Find arguments which support an argument with the proposition "$prop_2$" in its premise set.

```
match ?arg support <?pr[/{"prop2"}], ?c>
return ?arg
```

Results: <{"$prop_5$", "$prop_6$"}, "$concl_4$">

Data



ArgQL Syntax

V: an infinite set of variables

- Start with the character '?' (e.g. ?x)

Values for propositions are quoted (e.g. *"prop$_i$"*)

Language expressions combined by the two basic constructs:

- *Argument patterns*: expressions used to match *arguments* in the debate graph
- *Path patterns:* expressions used to match *sequences of arguments* in the debate graph

ArgQL Syntax

Argument patterns

A single variable, or be of the form:

```
< pr [filters], c >
```

- $pr \in V$ a variable that matches the premise set
- $c \in P \cup V$ a variable or proposition value that matches with the conclusion.
- the part *[filters]* is **optional** and adds constraints on the premise set

Given that $p_i \in P$, the supported filters are:

- *Inclusion:* [/{p_1, …, p_n}] propositions p_i must be included in *pr*.
- *Join*: [.?x] or [.{p_1, …, p_n}] pr must be joint with the given set.
- *Disjoin*: [!.?x] or [!.{p_1, …, p_n}] pr must be disjoint with the given set
- *Equality*: [= ?x] or [= {p_1, …, p_n}] pr must have exactly the same elements with the given set

ArgQL Syntax

Path patterns

Keywords for relations:

- '**rebut**' , '**undercut**' , '**attack**' , '**endorse**' , '**back**' , '**support**'

Symbol for relation sequence: '/'

- E.g. *rebut/endorse/attack*

Numerical indicators

- **'+'** : one or more occurrences of the path *e.g. (attack/support)+*
- **'*n** ' :exactly *n* occurrences of the path *e.g. (endorse/undercut)*3*

Arbitrary combinations to support many complex types of path patterns

- *E.g. ((attack/support)*2)/rebut, ((rebut/undercut)*2/(support/attack)*3)+*

Issues to be addressed at performance evaluation stage:

- Complexity of '+'
- Cycles on the paths

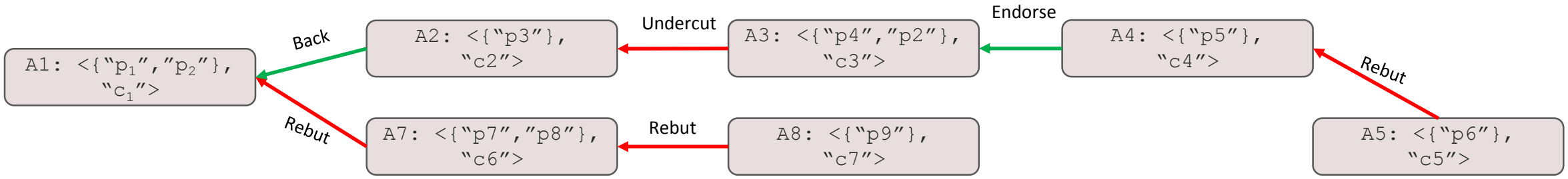
ArgQL Syntax

BNF grammar for the syntax of ArgQL

```
query  ::=  'MATCH'  pattern(',' pattern) *
pattern ::= argpattern (pathpattern argpattern)*
argpattern ::= variable | '<' premisepattern ',' proposition '>'
premisepattern ::=  variable ( premisefilter ) ?
premisefilter ::= '[' ( '/' | ':' | '!' | '=' | '!=') propositionset ']'
propositionset ::=  '{'  proposition  (' , '  proposition)*  '}'
pathpattern ::= path ('/' path) *
path ::= relsequence | '(' relsequence ')' length
relsequence ::= relation ( '/' relation )*
relation ::= ('ATTACK'|'SUPPORT'|'REBUT'|'UNDERCUT'|'ENDORSE'|'BACK' )
length ::= '+' | '*' num
proposition ::= variable | string
variable ::=  '?' ('a'..'z'|'A'..'Z')('a'..'z'|'A'..'Z'|'_'|'0'..'9')*
string ::=  ""'.*?'"";
num ::= '0' .. '9' +;
```

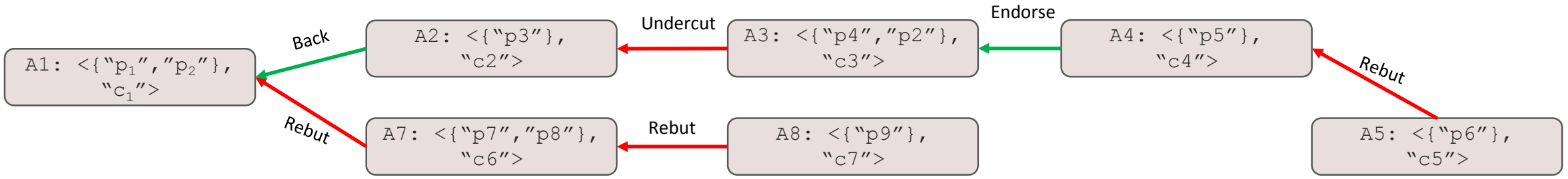
Semantics description

Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



Semantics description

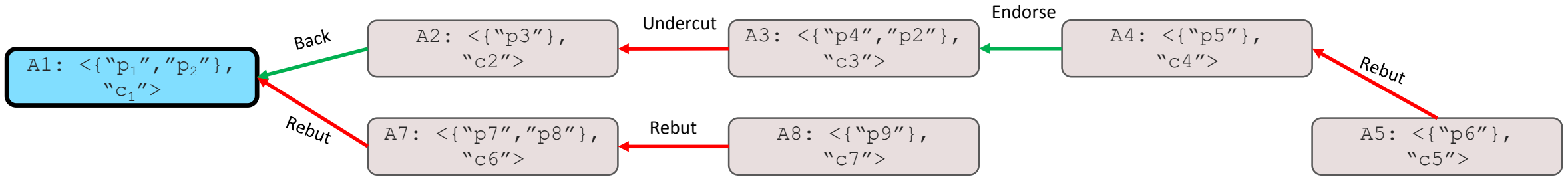
Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$

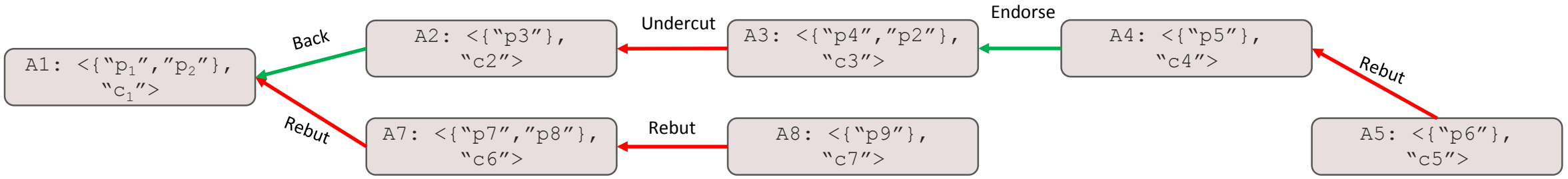


<?pr, "c1"> $\xrightarrow{\mu}$ <μ(?pr), μ("c1")> = A1: <{"p1", "p2"}, c1>

Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$

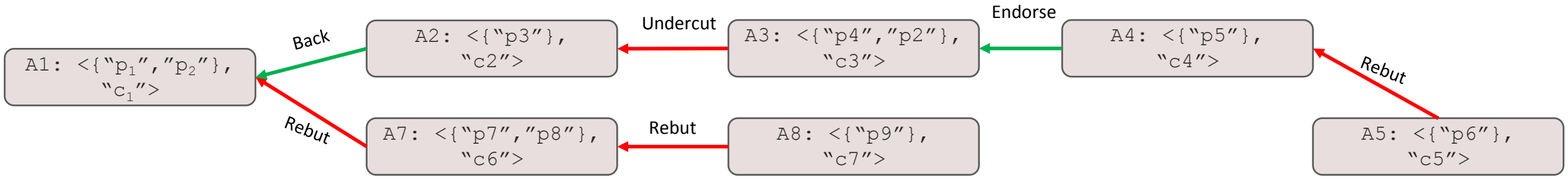


(attack/support)+ = {attack/support , attack/support/attack/support ,
attack/support/attack/support/attack/support ….}

Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

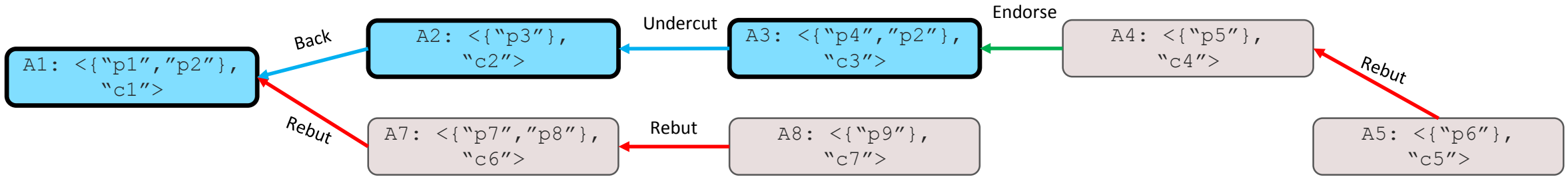
Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

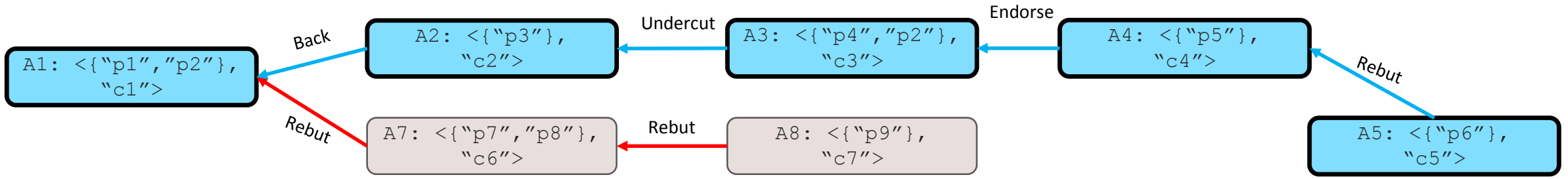
Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



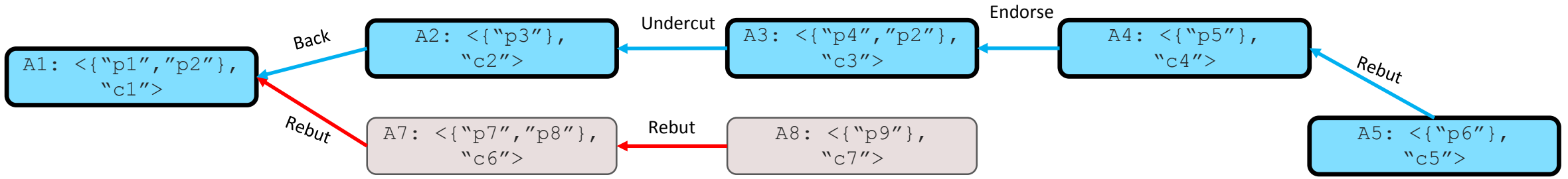
Evaluation of expression in D = {**A3** *attack/support* **A1**,

A5 *attack/support/attack/support* **A1**}

Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Semantics description

Semantics are defined above a function μ: $S \cup V \rightarrow S$, where $S = P \cup 2^P \cup A$



Evaluation of expression in D = {**A3** *attack/support* **A1**,

A5 *attack/support/attack/support* **A1**}

Query Answer: ***?arg = {A3, A5}***

Query: **Match** ?arg (attack/support)+ <?pr, "c1"> **return** ?arg

Query execution

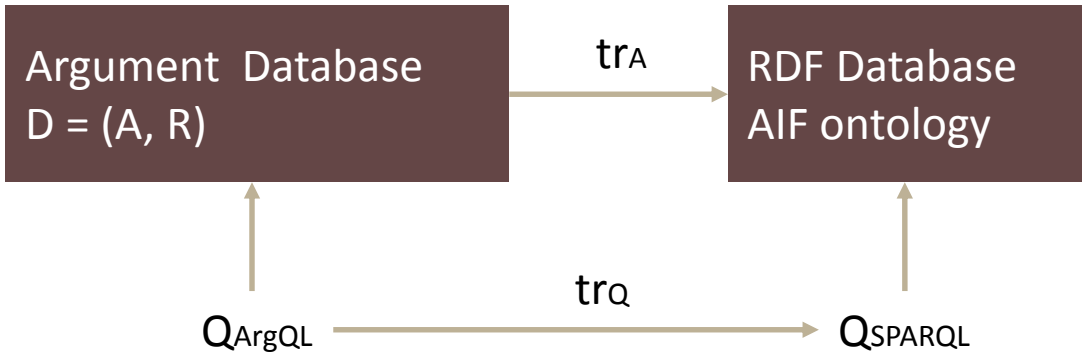
Translation into standard and well-optimized storage schemes

- Independence from the storage scheme.
- RDF and AIF scheme (target field is the Web, exploit SPARQL1.1 property paths)

Definition of two translation functions:

- tr_A: translates Argumentation data model into RDF instances of AIF ontology
- tr_Q: translates ArgQL queries into SPARQL

Formally prove that translation is semantics preserving



Conclusions

- Initial results from the definition of ArgQL, a declarative query language for querying dialogues constructed of arguments.
  - Syntax and semantics
- Future directions
  - Complete the proof of correctness for translation
  - Experimentation on real datasets and development of an endpoint to run queries
  - Data model enrichment with more complicated concepts (e.g. topics) and expand the syntax with the new concepts.
  - Integration with reasoning mechanisms for dynamic computations (e.g. Dung's acceptability)
  - Smart searching facilities (like keyword searching, imprecise textual mappings etc.)
  - Expressive power

Questions?