Bridging the Specification-Protocol Gap in Argumentation

Ashwag Omar Maghraby,
Dave Robertson,
Adela Grando and
Michael Rovatsos

E-mail : A.O.Maghraby@sms.ed.ac.uk
Website: http://homepages.inf.ed.ac.uk/s0961321/index.html

Outline

- Introduction
- Argument Interchange Format
- Research objectives
  - Lightweight Coordination Calculus (LCC)
  - *Techniques editing*
  - *CPNs*
- Conclusion

Introduction

Standalone Computer system

Distributed, open and dynamic computer systems

Some degree of autonomy

Abilities to communicate and negotiate on behalf of human user

Agent Technology

Agent and Multi-Agent Systems

- An agent is a computer system that acts autonomously in a dynamic, unpredictable and open environment.

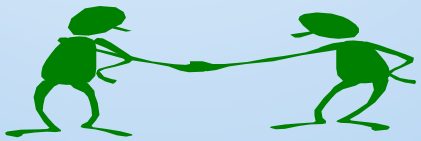
Agent and Multi-Agent Systems

- A multi-agent system is one that consists of a number of agents, which interact with one another in order to achieve their goals.

Agent and Multi-Agent Systems

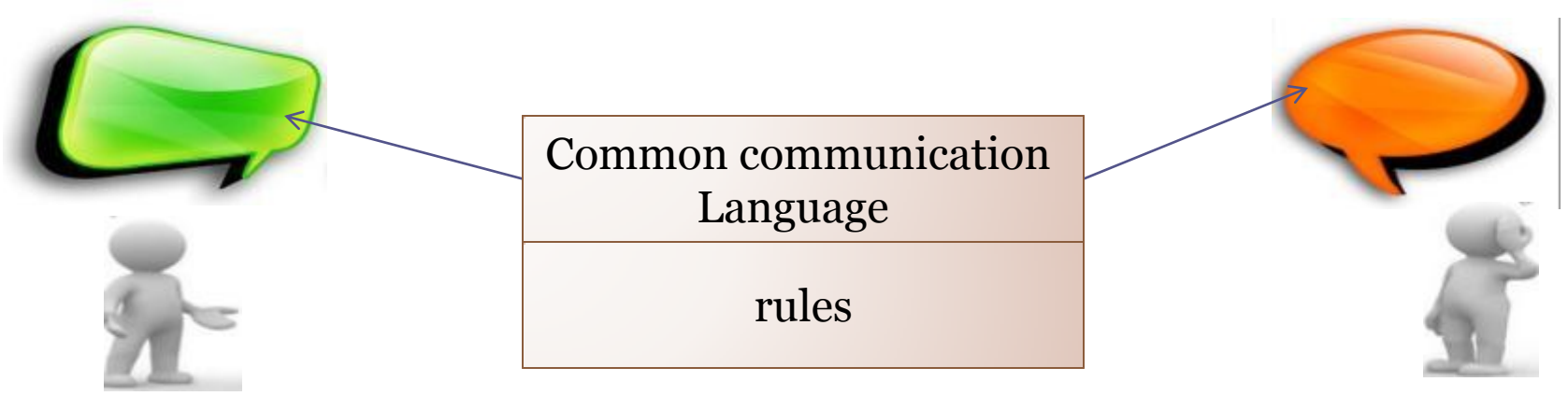
- To successfully interact, Agents need to:
  - Exchange information and explantations
  - Resolve conflicts of opinions and interests
  - Reach decisions

They need to engage in argument (dialogue)



Argumentation for agent communication

- An argumentation is a basic mechanism for interaction.
- When agents are collaborative, the argumentation process progresses through a dialogue.
- The rules of agent communication language are called Interaction Protocol.
- Interaction protocols are possible communication scenarios between individual agents in multi-agent systems.



Argumentation for agent communication

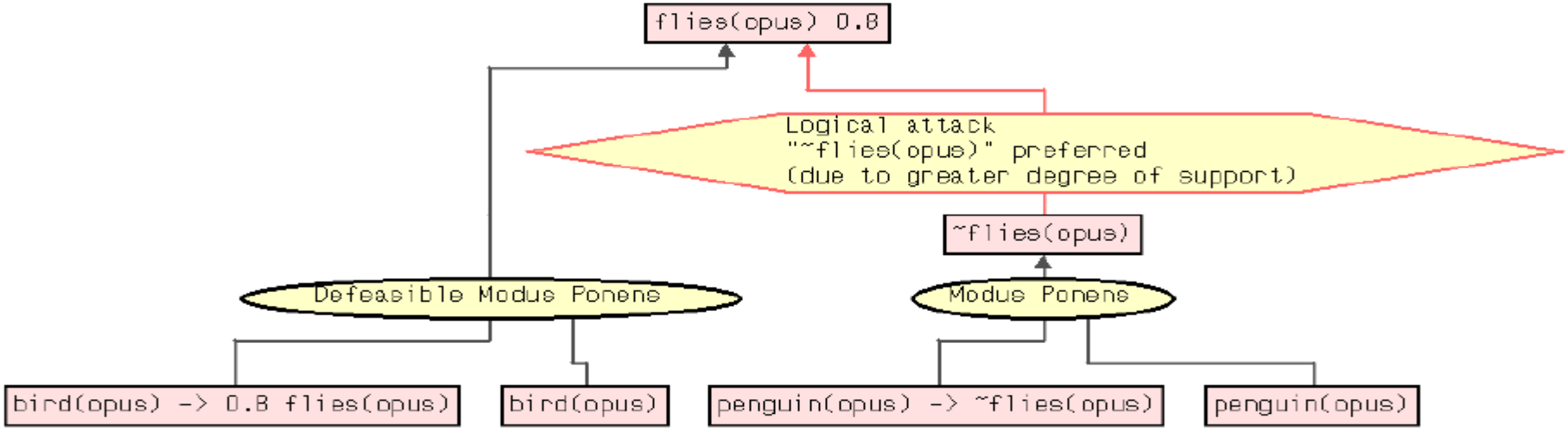
- In, recent years, significant progress has been made in the argumentation community for modelling agent communication.

- However, the argumentation community encounters various problems:
  - The lack of shared for an interchange format for arguments
  - The lack of ability to implement complex systems of argument form high level specifications.

Argument Interchange Format (AIF)

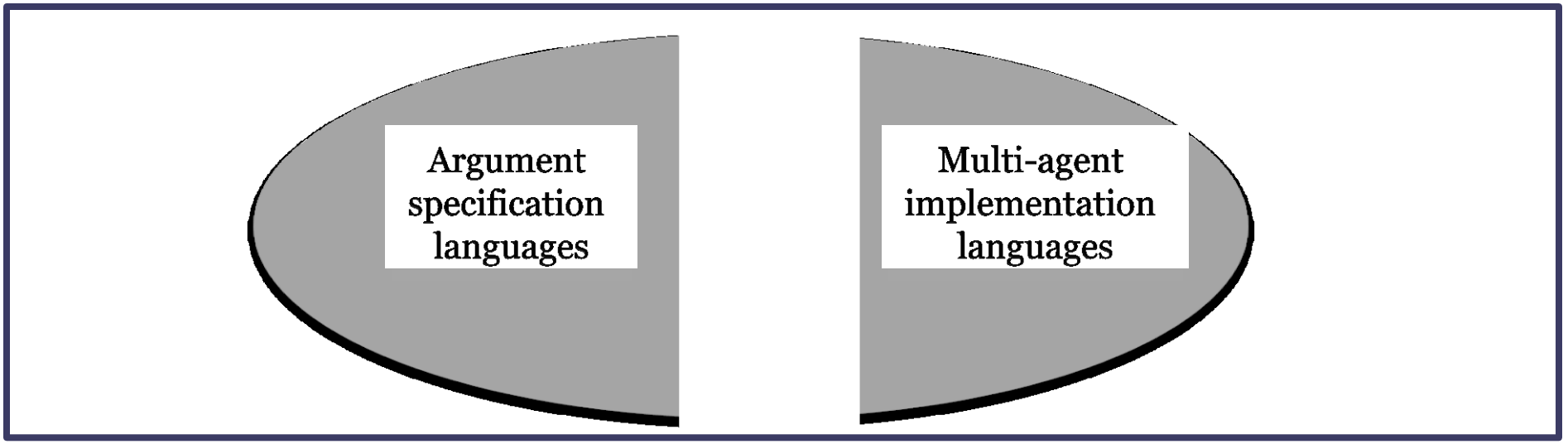
- The *AIF* is the result of an international effort which provides a common language to exchange argumentation concepts among agents in a MAS.

- It provides an ontology which represents an argument as a network of linked nodes.



Problems

- However, the AIF dose not solve the implementation problem.
- The AIF language is an abstract language.
- Concerned with only the structure of argument, while implemented multi-agent systems are concrete and need social constraints via protocols.
- This means that there is a gap between argument specification languages and multi-agent implementation languages.



Research objectives

Bridge the gap between AIF and agent protocol using a combination of :

1. Transformational synthesis
2. Model checking

First Objective:

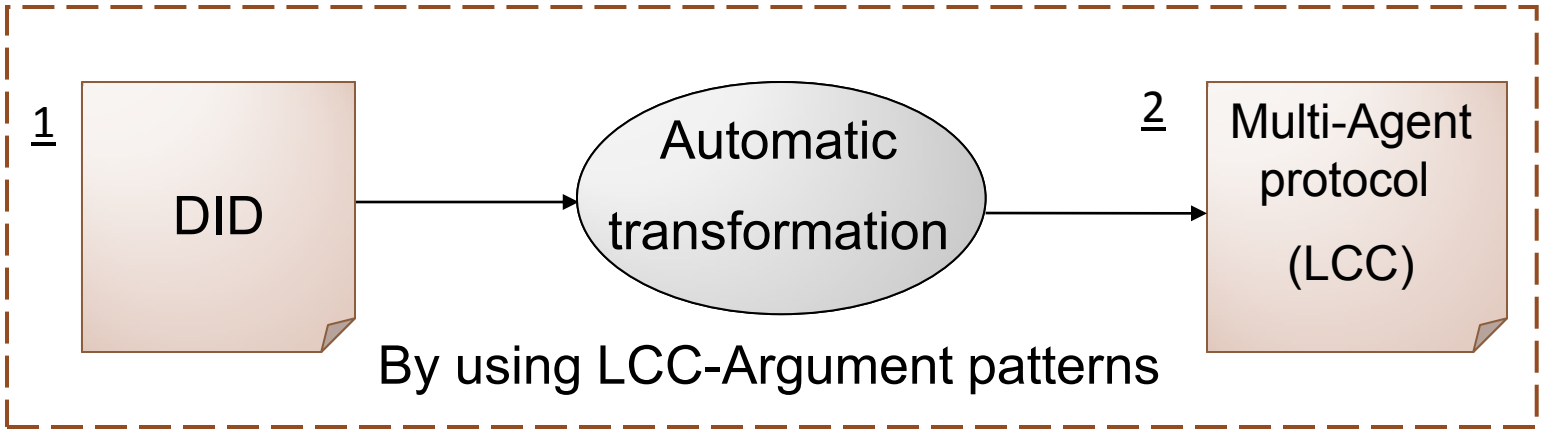
Transformational synthesis

Transformational synthesis

1. Propose a new high level control flow specification language called Dialogue Interaction Diagram (DID), which is an extension of AIF, used to specify the argument protocol in an abstract way.

2. Synthesis of concrete multi-agent protocols from DID by reusing reliable and common argument patterns

Part 1



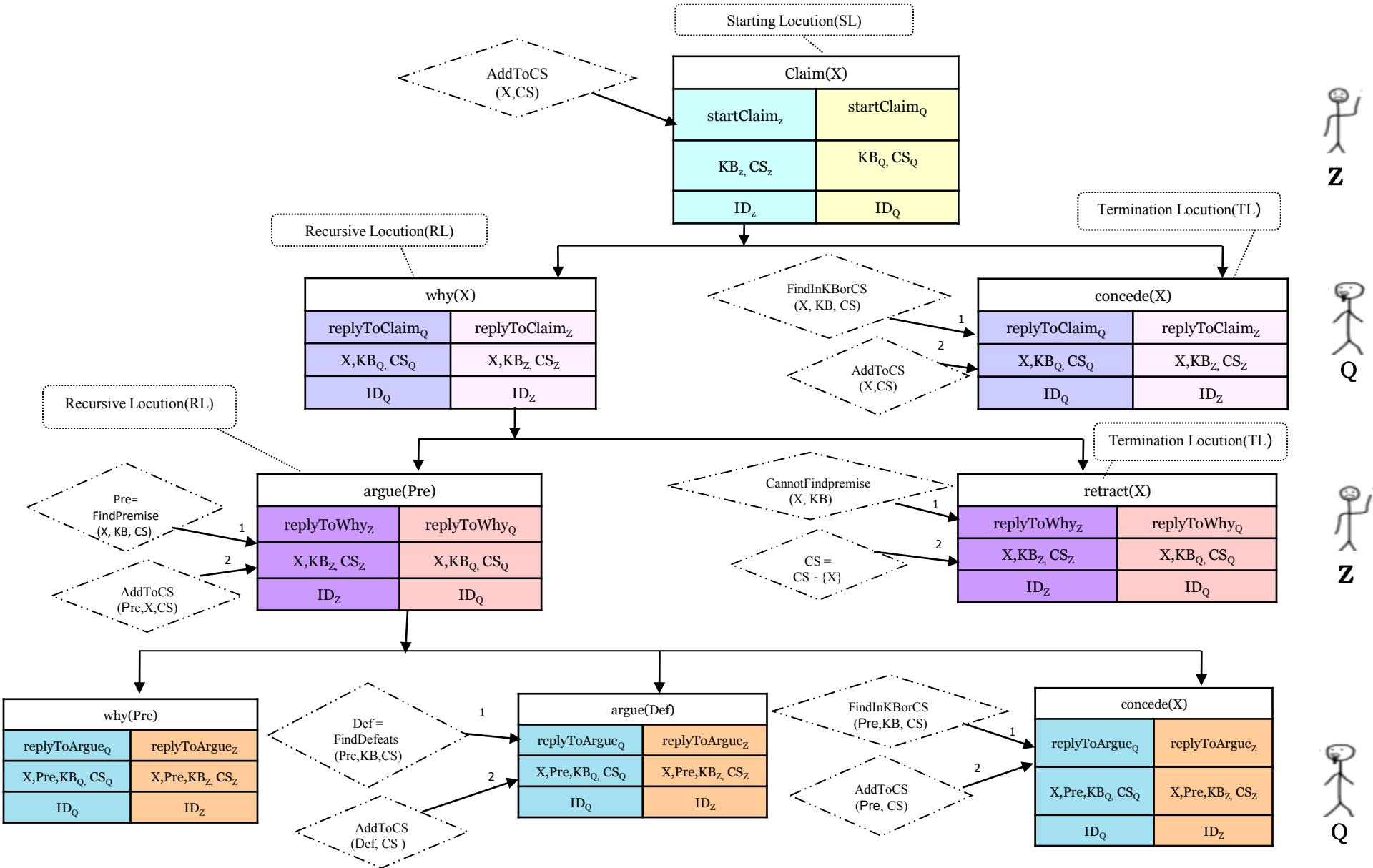
1.Dialogue Interaction Diagram

- Dialogue Interaction Diagram (DID)
- Graphical language
- DID can be used by designers to describe arguments interaction protocol:
  1. Dialogue Interaction Diagram for 2 agents (DID)
  2. Dialogue Interaction Diagram for N-agent (DIDN)

Dialogue Interaction Diagram

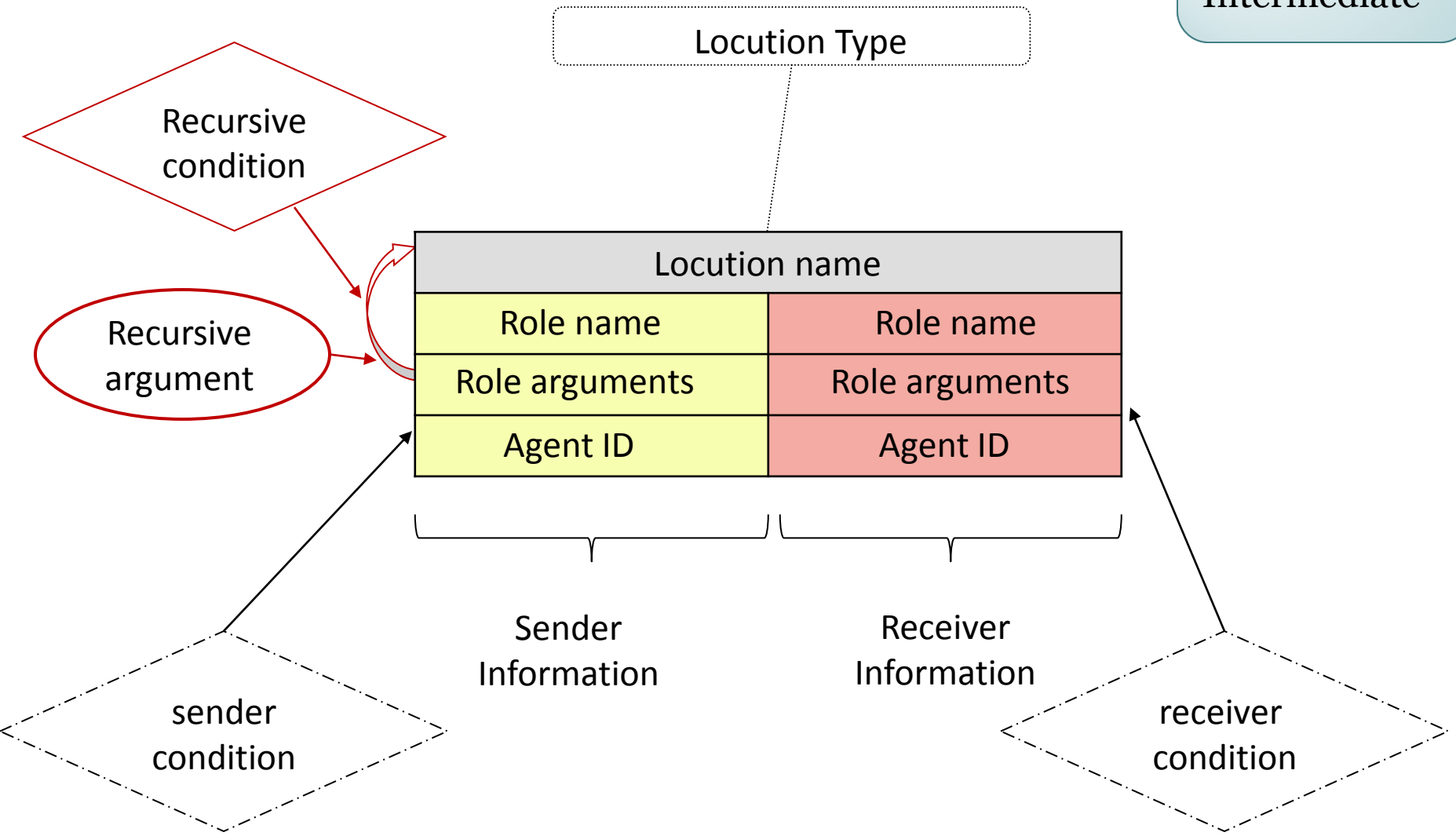
- DID defines four different rules:
  1. Locution rules (permitted moves)
  2. Commitment rules
  3. Structural rules (reply rules)
  4. Precondition rules

DID(2 Agents)

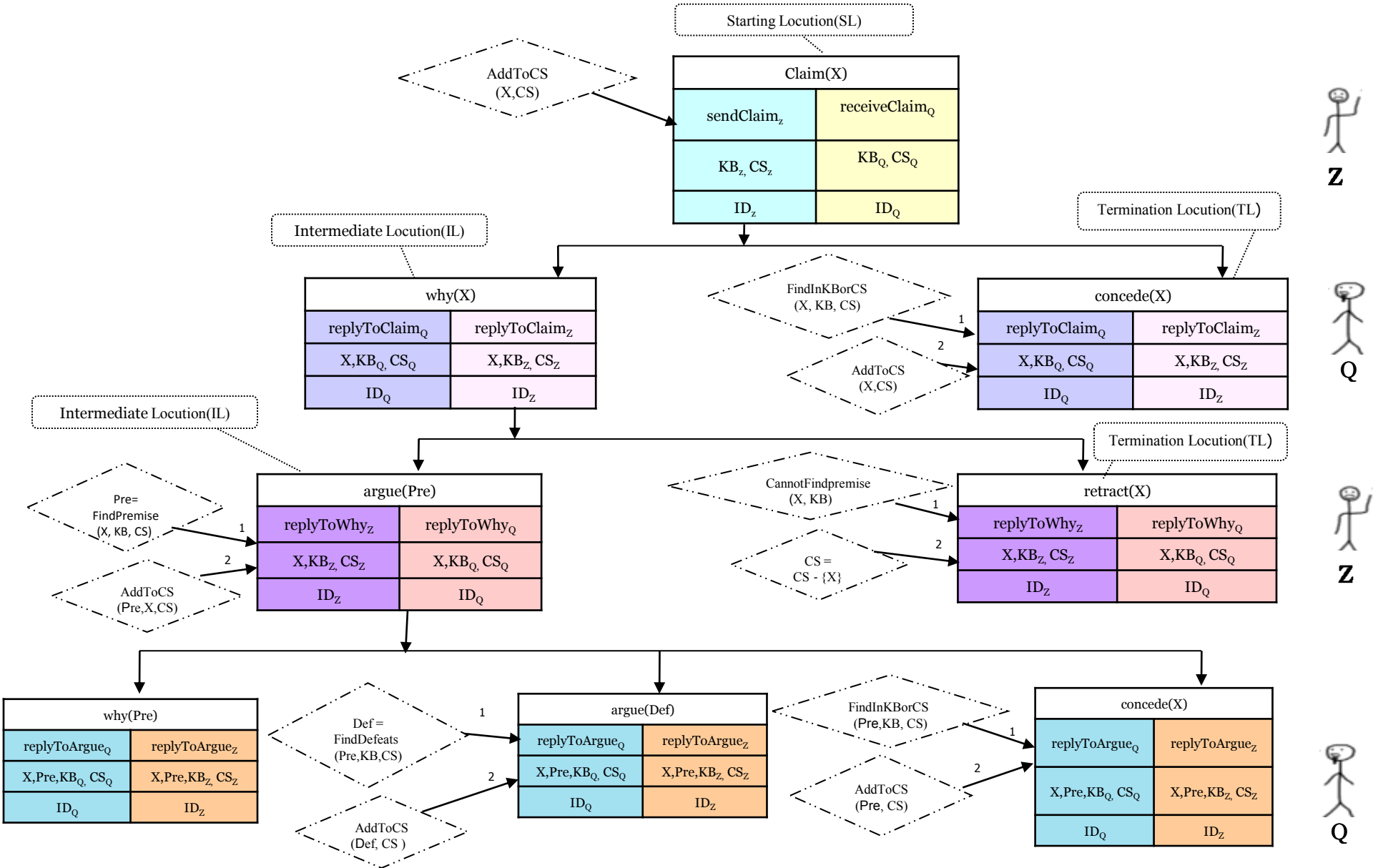

Starting Locution(SL)
AddToCS (X,CS)
Claim(X)
startClaim$_z$
startClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$
Recursive Locution(RL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_z$
X,KB$_Q$, CS$_Q$
X,KB$_z$, CS$_z$
ID$_Q$
ID$_z$
FindInKBorCS (X, KB, CS)
AddToCS (X,CS)
concede(X)
Recursive Locution(RL)
Termination Locution(TL)
Pre= FindPremise (X, KB, CS)
AddToCS (Pre,X,CS)
argue(Pre)
replyToWhy$_z$
replyToWhy$_Q$
CannotFindpremise (X, KB)
CS = CS - {X}
retract(X)
why(Pre)
Def = FindDefeats (Pre,KB,CS)
AddToCS (Def, CS)
argue(Def)
FindInKBorCS (Pre,KB, CS)
AddToCS (Pre, CS)
concede(X)
z
Q
z
Q

Locution Icon

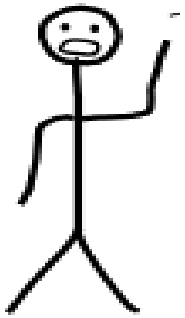
Starting
Termination
Intermediate



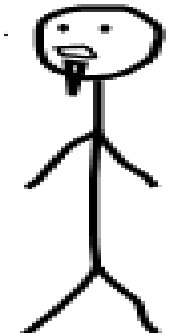
DID(2 Agents)


Starting Locution(SL)
AddToCS (X,CS)
Claim(X)
sendClaim$_z$
receiveClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$
Intermediate Locution(IL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_z$
X,KB$_Q$, CS$_Q$
X,KB$_z$, CS$_z$
ID$_Q$
ID$_z$
FindInKBorCS (X, KB, CS)
AddToCS (X,CS)
1
2
concede(X)
Intermediate Locution(IL)
Termination Locution(TL)
Pre= FindPremise (X, KB, CS)
AddToCS (Pre,X,CS)
argue(Pre)
replyToWhy$_z$
replyToWhy$_Q$
CannotFindpremise (X, KB)
CS = CS - {X}
retract(X)
why(Pre)
replyToArgue$_Q$
replyToArgue$_z$
X,Pre,KB$_Q$, CS$_Q$
X,Pre,KB$_z$, CS$_z$
Def = FindDefeats (Pre,KB,CS)
AddToCS (Def, CS)
argue(Def)
FindInKBorCS (Pre,KB, CS)
AddToCS (Pre, CS)
concede(X)
z
Q
z
Q

Car safety Example

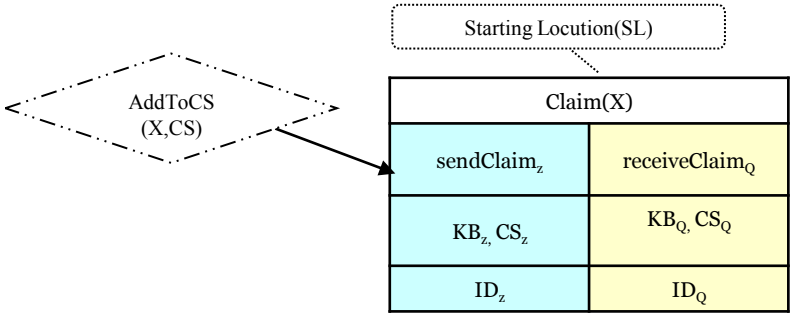


Z



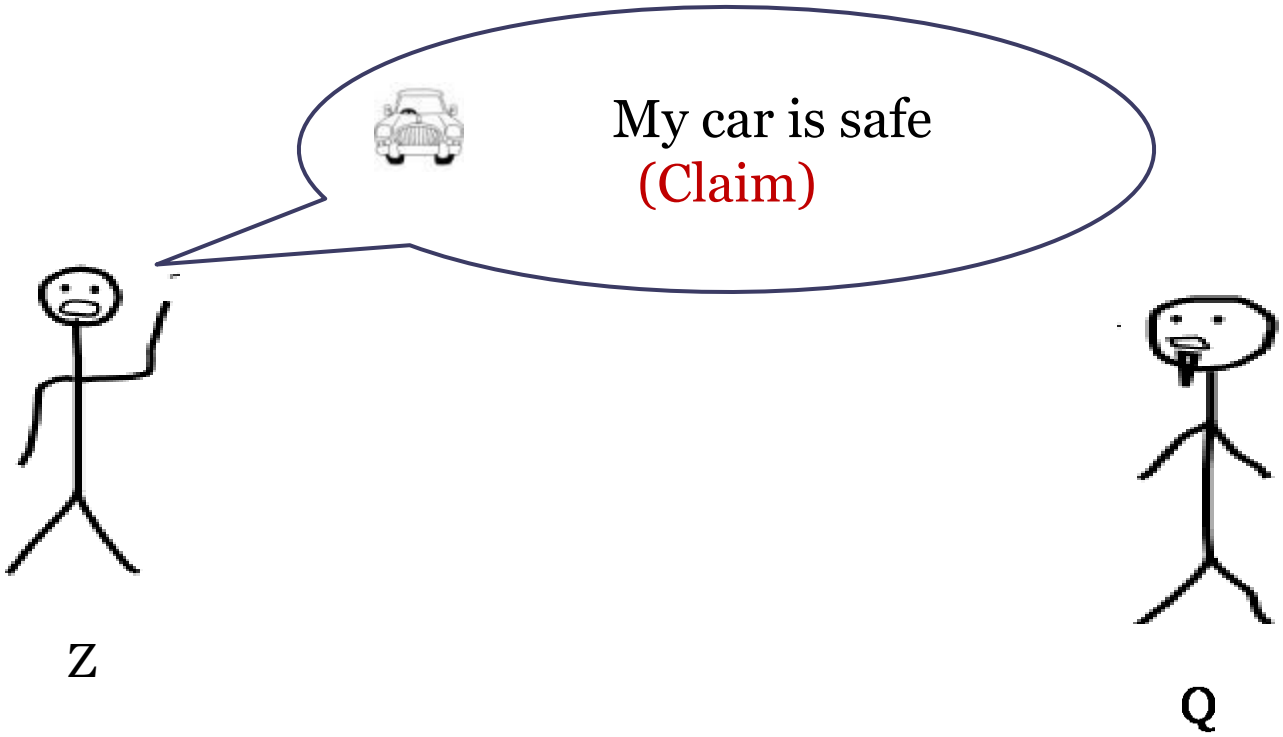
Q

DID(2 Agents)

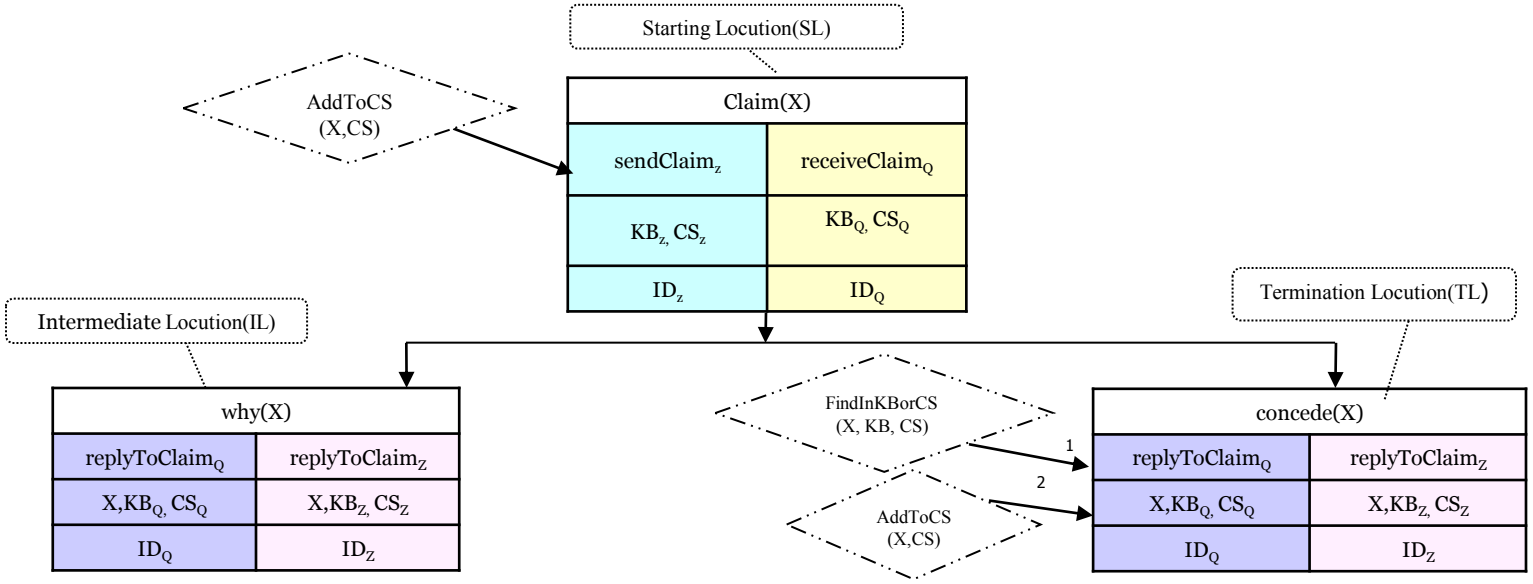

Starting Locution(SL)
AddToCS
(X,CS)
Claim(X)
sendClaim$_z$
receiveClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$


Z

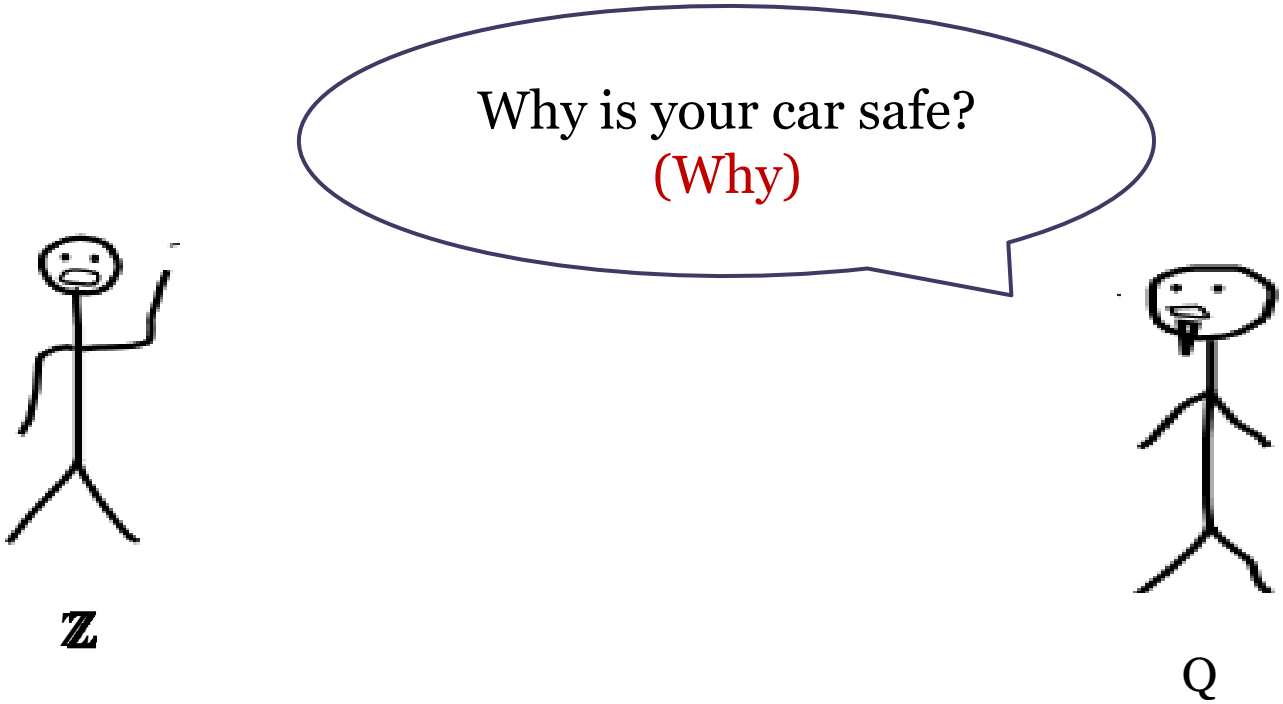
Car safety Example



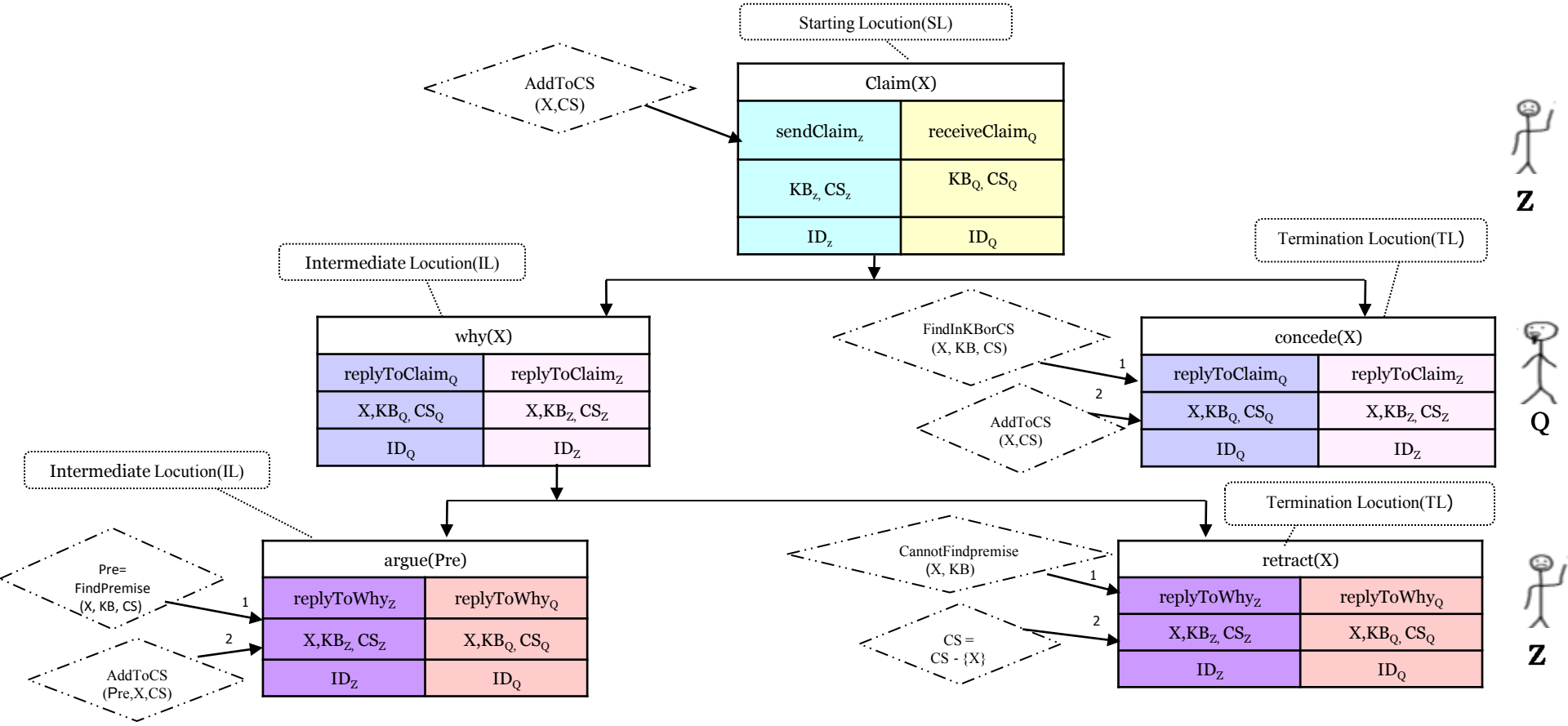
DID(2 Agents)


Starting Locution(SL)
AddToCS
(X,CS)
Claim(X)
sendClaim$_z$
receiveClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$
Intermediate Locution(IL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_Z$
X,KB$_Q$, CS$_Q$
X,KB$_Z$, CS$_Z$
ID$_Q$
ID$_Z$
FindInKBorCS
(X, KB, CS)
AddToCS
(X,CS)
1
2
concede(X)
replyToClaim$_Q$
replyToClaim$_Z$
X,KB$_Q$, CS$_Q$
X,KB$_Z$, CS$_Z$
ID$_Q$
ID$_Z$
Z
Q

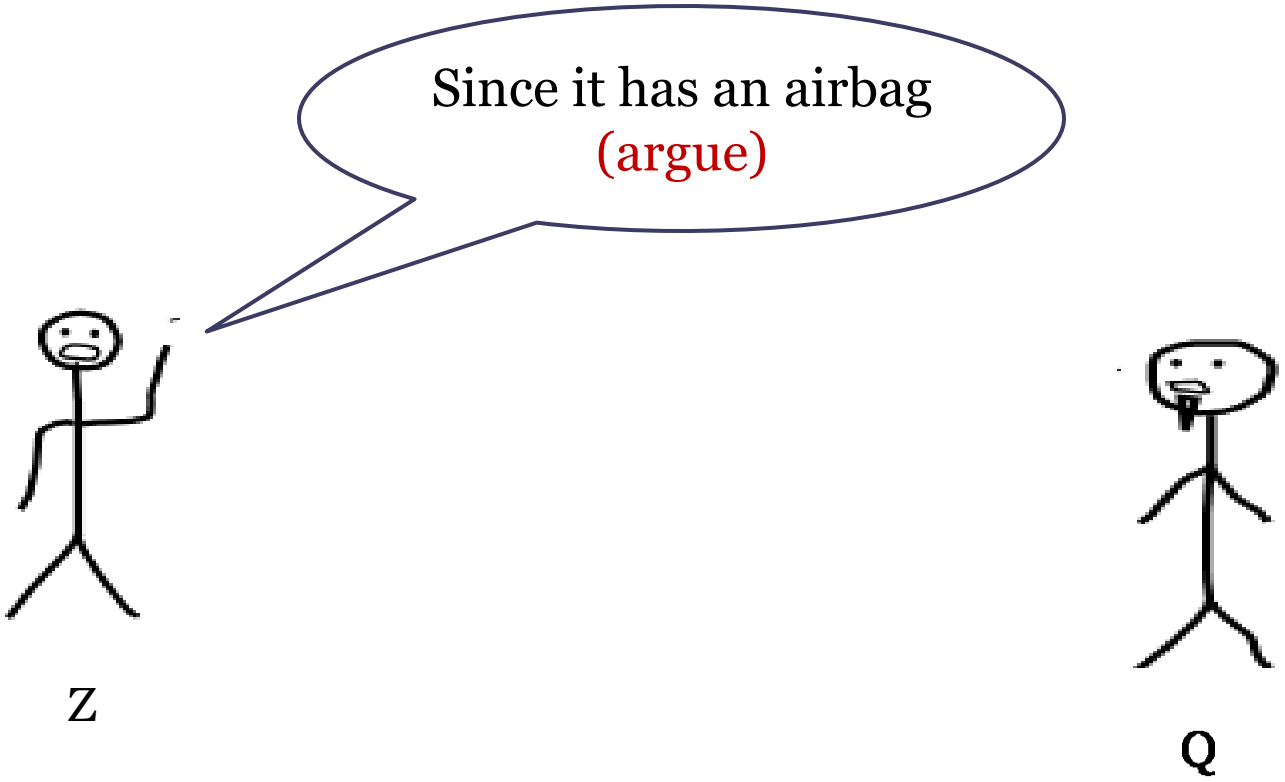
Car safety Example



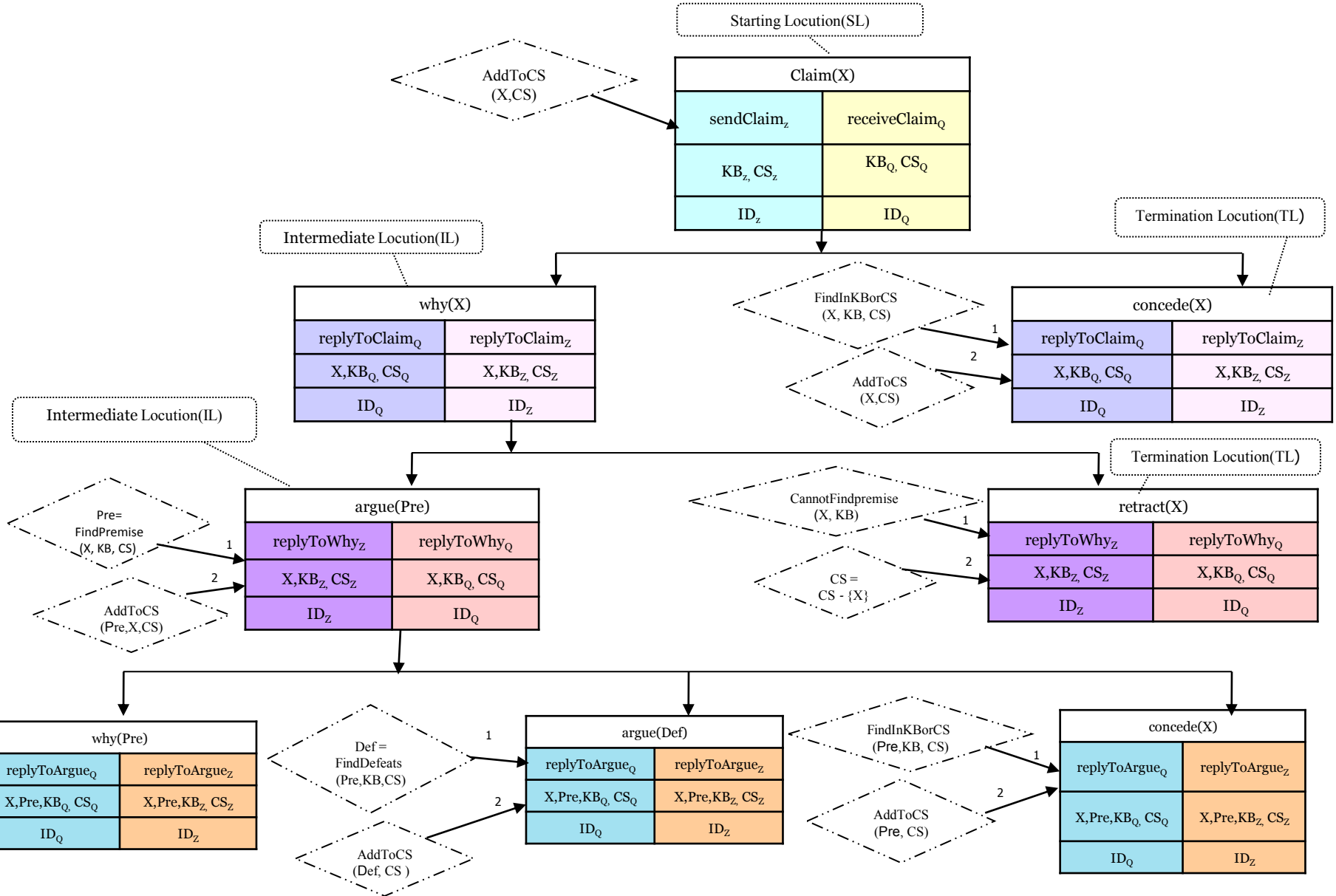
DID(2 Agents)


Starting Locution(SL)
AddToCS
(X,CS)
Claim(X)
sendClaim$_z$
receiveClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$
Intermediate Locution(IL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_Z$
X,KB$_Q$, CS$_Q$
X,KB$_Z$, CS$_Z$
ID$_Q$
ID$_Z$
FindInKBorCS
(X, KB, CS)
AddToCS
(X,CS)
1
2
concede(X)
Intermediate Locution(IL)
Termination Locution(TL)
Pre=
FindPremise
(X, KB, CS)
AddToCS
(Pre,X,CS)
argue(Pre)
replyToWhy$_Z$
replyToWhy$_Q$
CannotFindpremise
(X, KB)
CS =
CS - {X}
retract(X)
Z
Q
Z

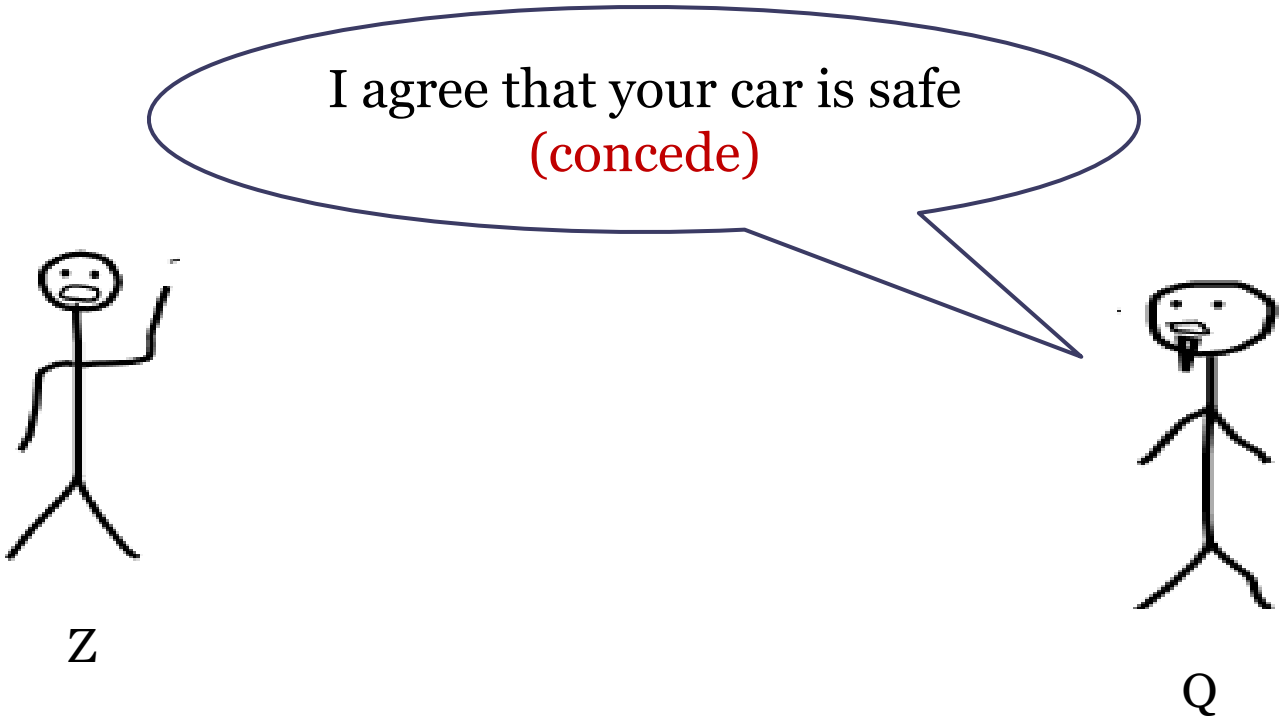
Car safety Example



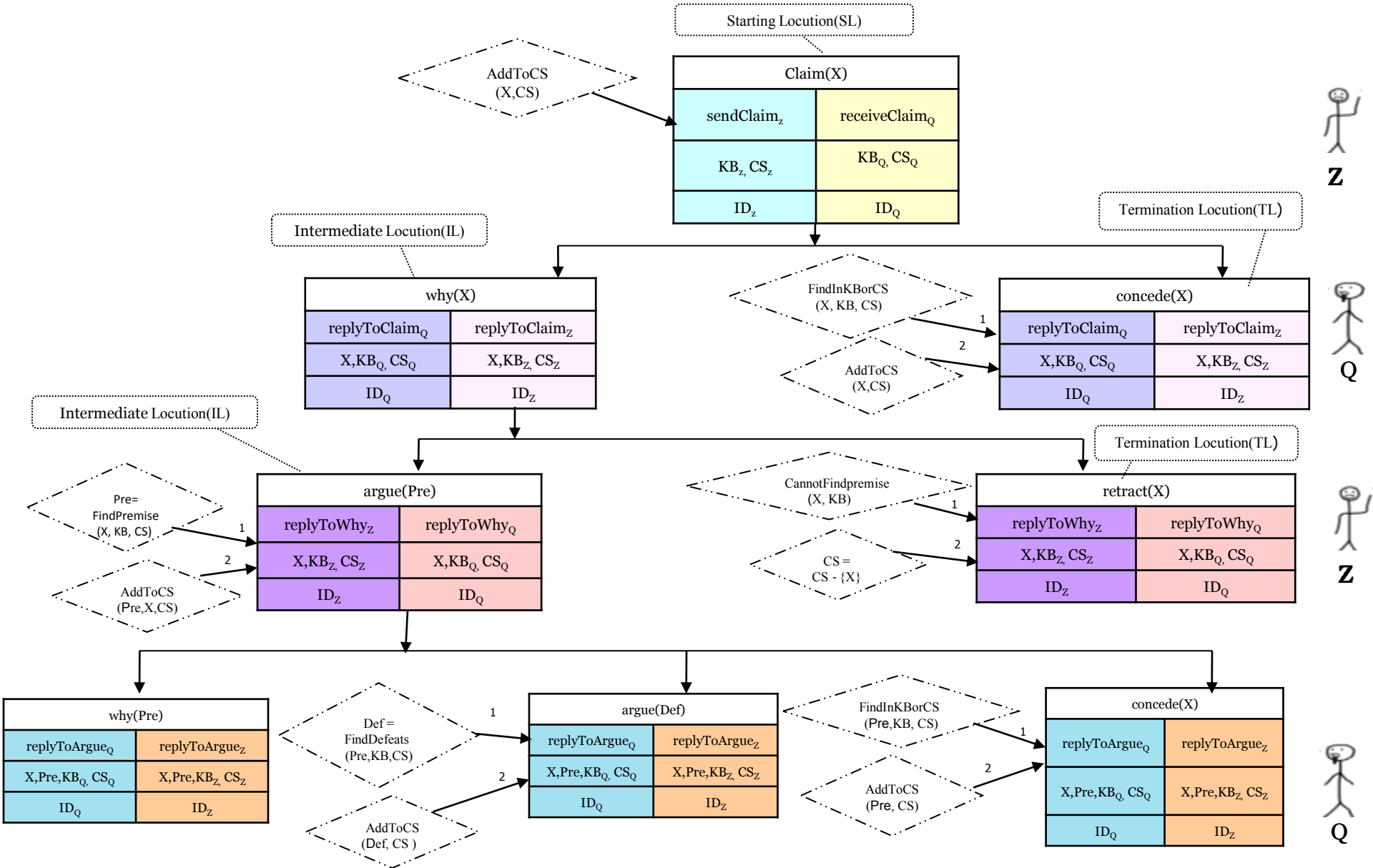
DID(2 Agents)


Starting Locution(SL)
AddToCS (X,CS)
Claim(X)
sendClaim$_z$
receiveClaim$_Q$
KB$_z$, CS$_z$
KB$_Q$, CS$_Q$
ID$_z$
ID$_Q$
Intermediate Locution(IL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_z$
X,KB$_Q$, CS$_Q$
X,KB$_z$, CS$_z$
ID$_Q$
ID$_z$
FindInKBorCS (X, KB, CS)
AddToCS (X,CS)
1
2
concede(X)
Intermediate Locution(IL)
Termination Locution(TL)
Pre= FindPremise (X, KB, CS)
AddToCS (Pre,X,CS)
argue(Pre)
replyToWhy$_z$
replyToWhy$_Q$
CannotFindpremise (X, KB)
CS = CS - {X}
retract(X)
why(Pre)
replyToArgue$_Q$
replyToArgue$_z$
X,Pre,KB$_Q$, CS$_Q$
X,Pre,KB$_z$, CS$_z$
Def = FindDefeats (Pre,KB,CS)
AddToCS (Def, CS)
argue(Def)
FindInKBorCS (Pre,KB, CS)
AddToCS (Pre, CS)
concede(X)
z
Q
z
Q

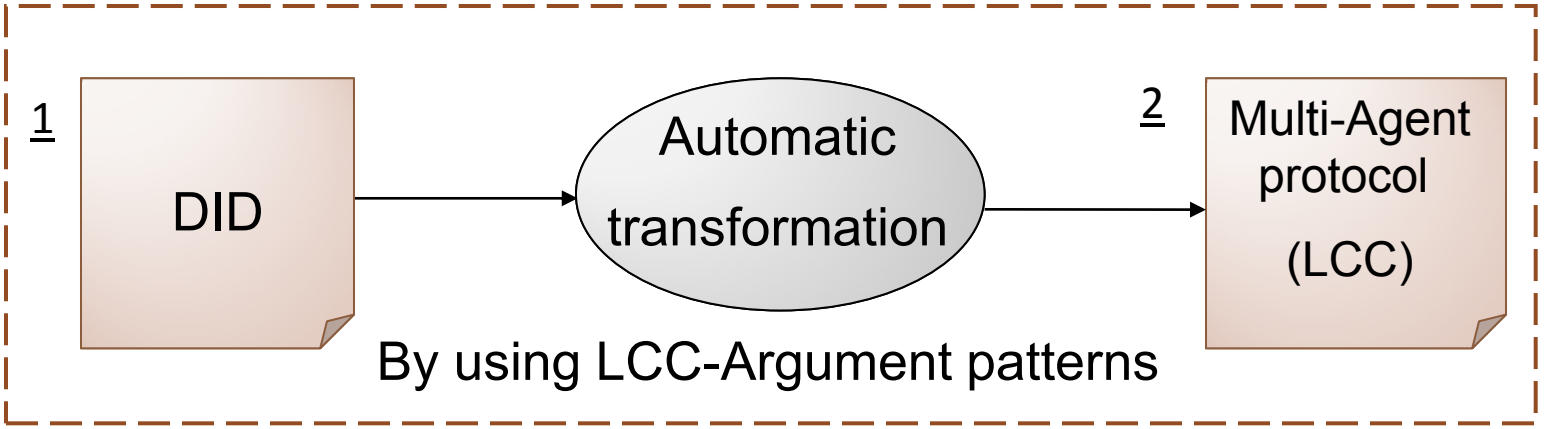
Car safety Example



DID(2 Agents)


Starting Locution(SL)
AddToCS (X,CS)
Claim(X)
sendClaim$_Z$
receiveClaim$_Q$
KB$_Z$, CS$_Z$
KB$_Q$, CS$_Q$
ID$_Z$
ID$_Q$
Z
Intermediate Locution(IL)
Termination Locution(TL)
why(X)
replyToClaim$_Q$
replyToClaim$_Z$
X,KB$_Q$, CS$_Q$
X,KB$_Z$, CS$_Z$
ID$_Q$
ID$_Z$
FindInKBorCS (X, KB, CS)
AddToCS (X,CS)
1
2
concede(X)
Q
Intermediate Locution(IL)
Termination Locution(TL)
Pre= FindPremise (X, KB, CS)
AddToCS (Pre,X,CS)
argue(Pre)
replyToWhy$_Z$
replyToWhy$_Q$
CannotFindpremise (X, KB)
CS = CS - {X}
retract(X)
Z
why(Pre)
replyToArgue$_Q$
replyToArgue$_Z$
X,Pre,KB$_Q$, CS$_Q$
X,Pre,KB$_Z$, CS$_Z$
Def = FindDefeats (Pre,KB,CS)
AddToCS (Def, CS)
argue(Def)
FindInKBorCS (Pre,KB, CS)
AddToCS (Pre, CS)
concede(X)
Q

2. Synthesis Concrete Protocols from DID



Build an efficient Interaction protocol in the easiest and quickest way by reusing common argument patterns

Tool

Dependent on a particular language

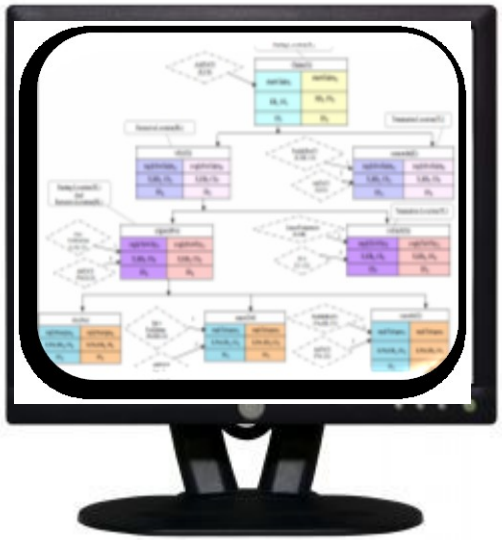
Independent of any particular algorithm or problem domain

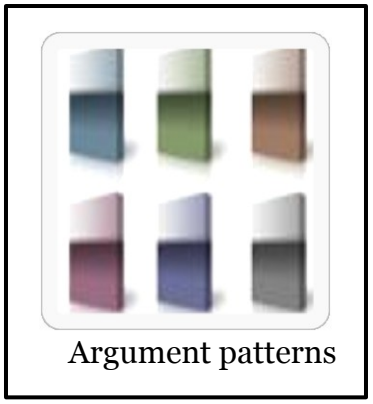
The patterns supported by Techniques editing method

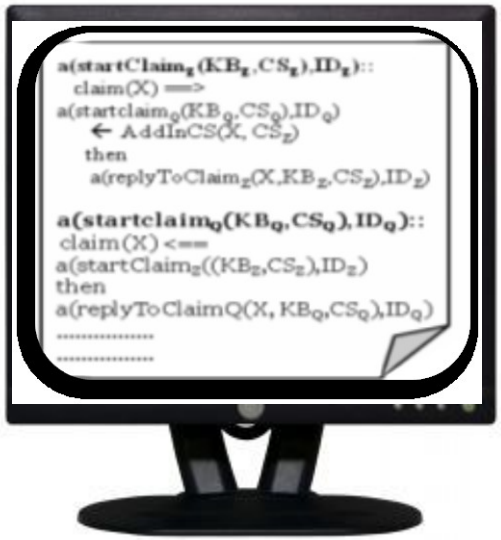
AIF

Simple

Lightweight Coordination Calculus (LCC)



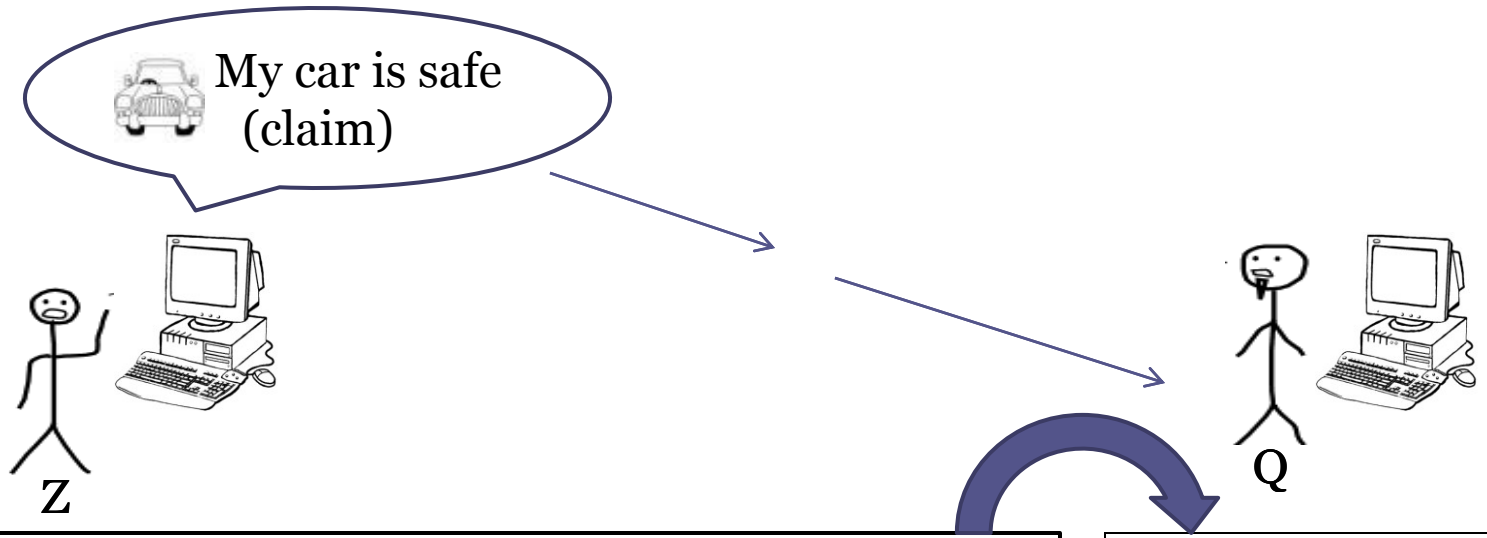




Dialogue Interaction Diagram

Generate Interaction Protocol Tool

Argument Interaction protocol



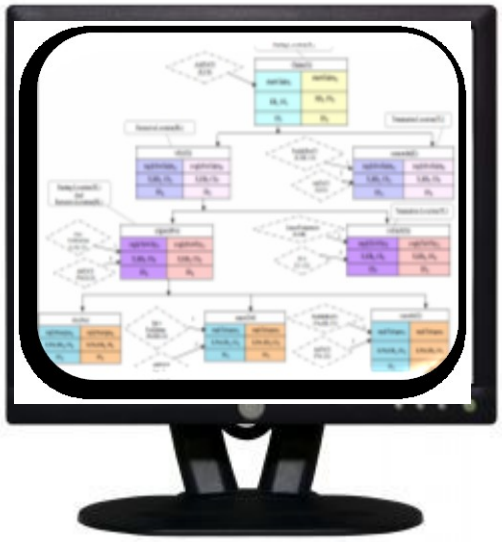
a(sender,z)::
claim("My car is safe") ==> a(receive, q)
then
a(reply,z)

a(receiver,q)::
claim("My car is safe") <== a(send, z)
then
a(askwhy,q)

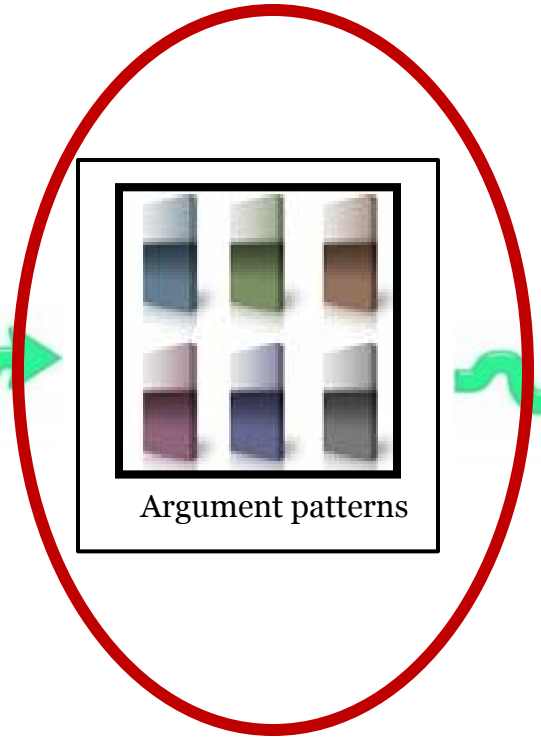
a(R1,ID1)::
Locution (X) ==> a(R1, ID2)
then
a(R3,ID1).

a(R2,ID2)::
Locution (X) <== a(R2, ID1)
then
a(R4,ID2).

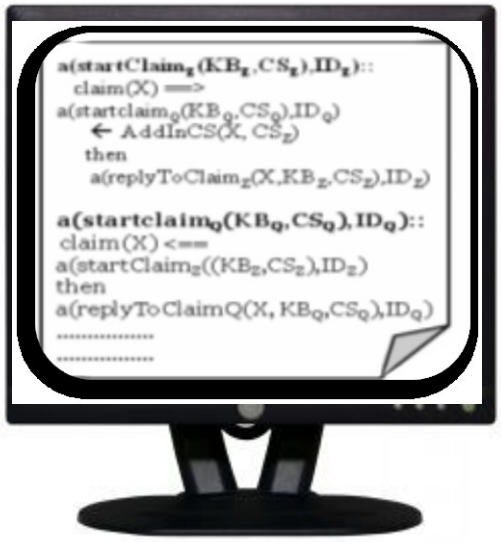
Tool



Dialogue Interaction Diagram



Generate Interaction Protocol Tool



Argument Interaction protocol

Techniques editing

•*Techniques editing* is a method used to synthesize Prolog clauses

Skeleton

- Pattern 1
- Pattern 2
- Pattern 3
- -
- -

Additions

- Addition 1
- Addition2
- Addition3
- -

Pattern 1

Addition2

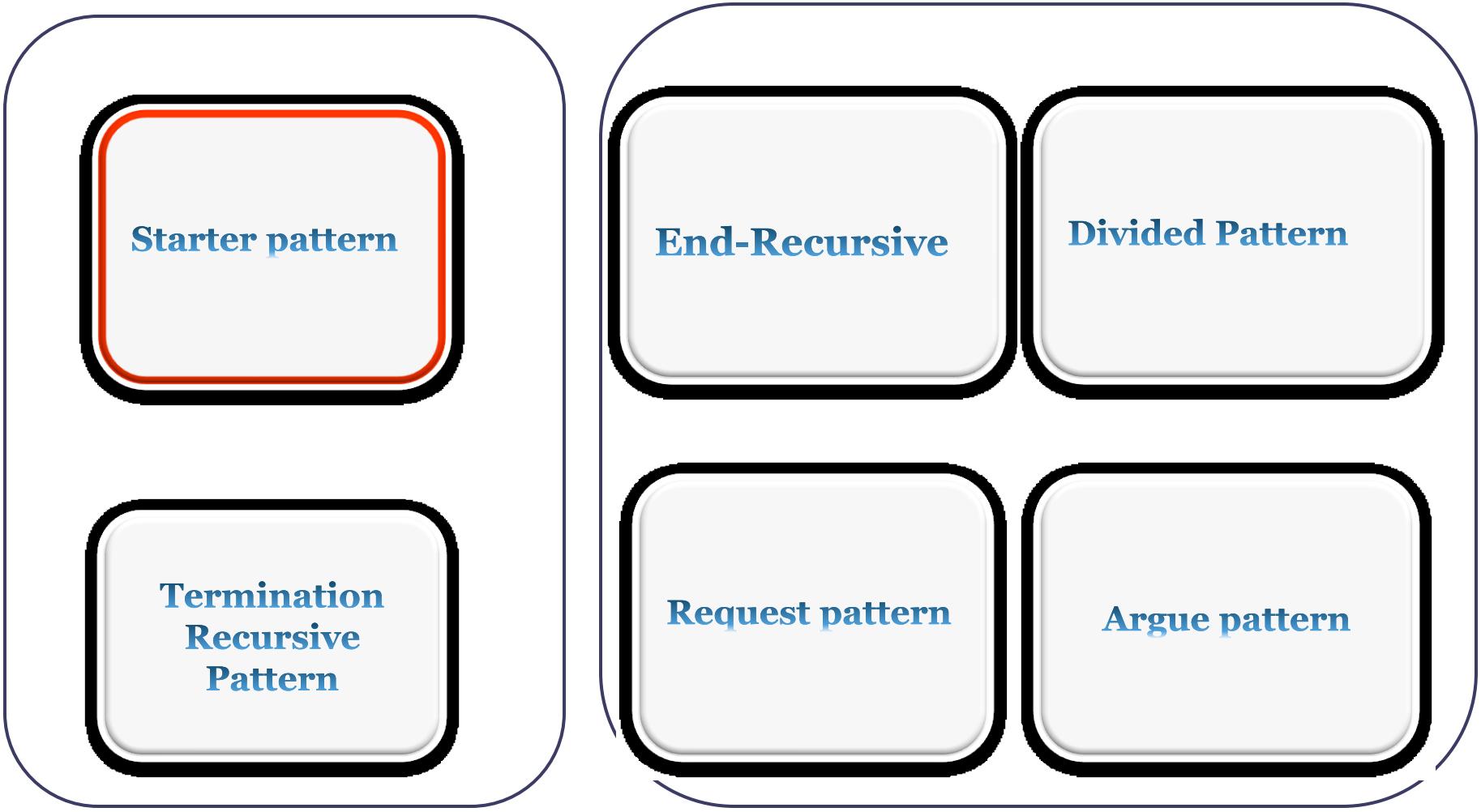
Addition3

Final program

LCC-Argument Pattern (Structure Synthesis)

Two for 2 agents

Four for N-agent



Starter pattern (S-P)

It is going to be used to start the argument between two agents.

$a(R_{Z1}(KB_Z,CS_Z),ID_Z)$::

$SL(X) \Longrightarrow a(R_{Q1}(KB_Q,CS_Q),ID_Q) \leftarrow C1(X, CS_Z)$
then
$a(R_{Z2}(KB_Z,CS_Z),ID_Z)$.

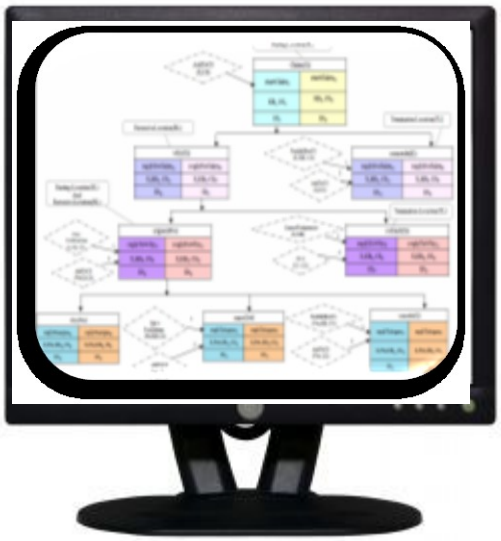
Sender

$a(R_{Q1}(KB_Q,CS_Q),ID_Q)$::

$SL(X) <== a(R_{Z1}(KB_Z,CS_Z),ID_Z)$
then
$a(R_{Q2}(KB_Q,CS_Q),ID_Q)$

Receiver

Transfer from DID to LCC by using Starter Pattern



Dialogue Interaction Diagram

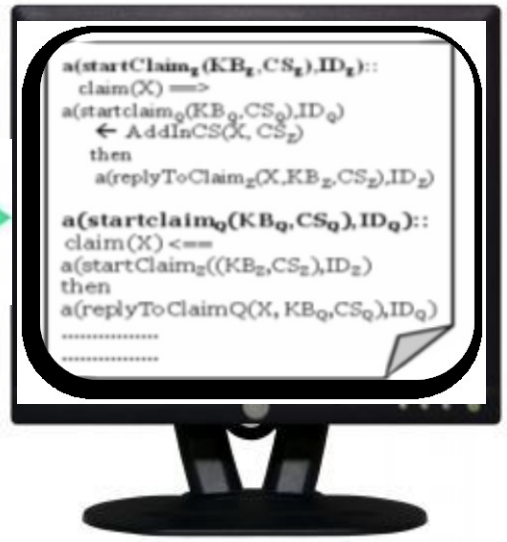
$a(R_{Z1}(KB_Z,CS_Z),ID_Z)::$

$SL(X) ==> a(R_{Q1}(KB_Q,CS_Q),ID_Q) \leftarrow C1(X,CS_Z)$
then
$a(R_{Z2}(KB_Z,CS_Z),ID_Z).$

$a(R_{Q1}(KB_Q,CS_Q),ID_Q)::$

$SL(X) <== a(R_{Z2}(KB_Z,CS_Z),ID_Z)$
then
$a(R_{Q2}(KB_Q,CS_Q),ID_Q)$

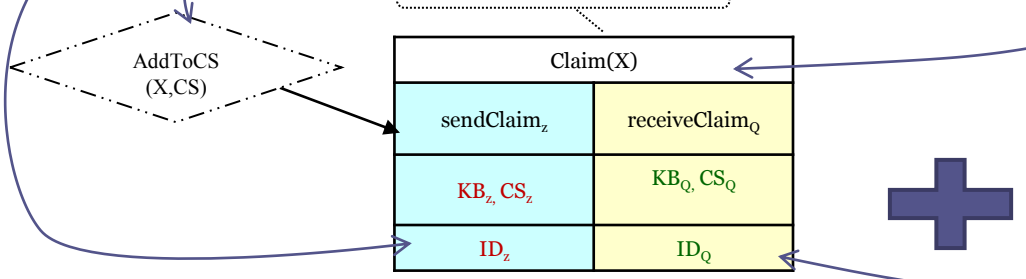
Starter Pattern



LCC Argument Interaction protocol

DID

Starter Pattern



$a(R_{Z1}(KB_Z,CS_Z),ID_Z)$::

$SL(X) \Longrightarrow a(R_{Q1}(KB_Q,CS_Q),ID_Q) \leftarrow C1(X, CS_Z)$
then
$a(R_{Z2}(KB_Z,CS_Z),ID_Z)$.

$a(R_{Q1}(KB_Q,CS_Q),ID_Q)$::

$SL(X) \Longleftarrow a(R_{Z1}(KB_Z,CS_Z),ID_Z)$
then
$a(R_{Q2}(KB_Q,CS_Q),ID_Q)$

$a(sendClaim_Z(KB_Z,CS_Z),ID_Z)$::

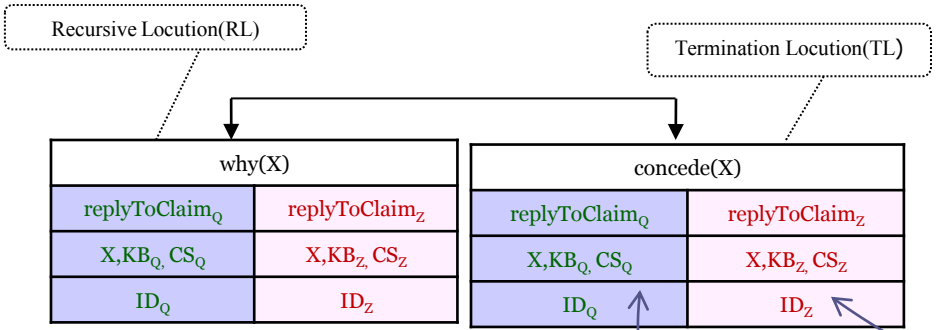
$claim(X) \Longrightarrow a(receiveclaim_Q(KB_Q,CS_Q),ID_Q) \leftarrow AddToCS(X, CS_Z)$

$a(receiveclaim_Q(KB_Q,CS_Q),ID_Q)$::

$claim(X) \Longleftarrow a(sendClaim_Z((KB_Z,CS_Z),ID_Z)$

LCC Argument Interaction protocol

DID



Starter Pattern

a(R$_{Z1}$(KB$_Z$,CS$_Z$),ID$_Z$)::

SL(X) ==> a(R$_{Q1}$(KB$_Q$,CS$_Q$),ID$_Q$) ← C1(X, CS$_Z$)
then
a(R$_{Z2}$ (KB$_Z$,CS$_Z$),ID$_Z$).

a(R$_{Q1}$(KB$_Q$,CS$_Q$),ID$_Q$)::

SL(X) <== a(R$_{Z1}$(KB$_Z$,CS$_Z$),ID$_Z$)
then
a(R$_{Q2}$(KB$_Q$,CS$_Q$),ID$_Q$)

a(startClaim$_Z$(KB$_Z$,CS$_Z$),ID$_Z$)::

claim(X) ==> a(receiveclaim$_Q$(KB$_Q$,CS$_Q$),ID$_Q$) ← AddToCS(X, CS$_Z$)
then
a(replyToClaim$_Z$(KB$_Z$,CS$_Z$),ID$_Z$).

a(startclaim$_Q$(KB$_Q$,CS$_Q$),ID$_Q$)::

claim(X) <== a(sendClaim$_Z$((KB$_Z$,CS$_Z$),ID$_Z$)
then
a(replyToClaim$_Q$(X,KB$_Q$,CS$_Q$),ID$_Q$).

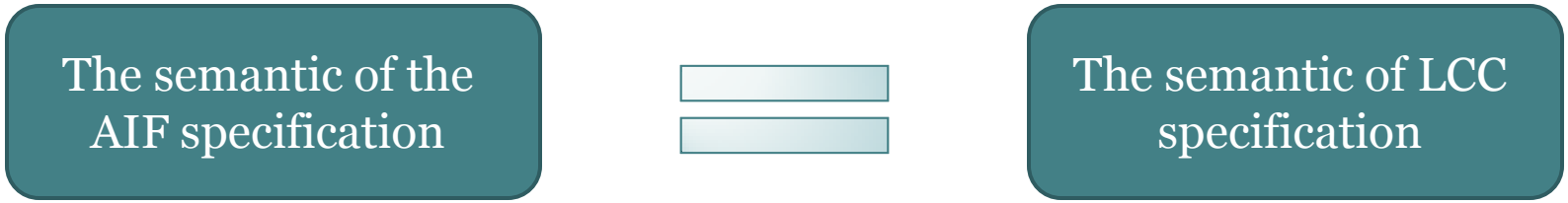
LCC Argument Interaction protocol

Second objective:

Model Checking

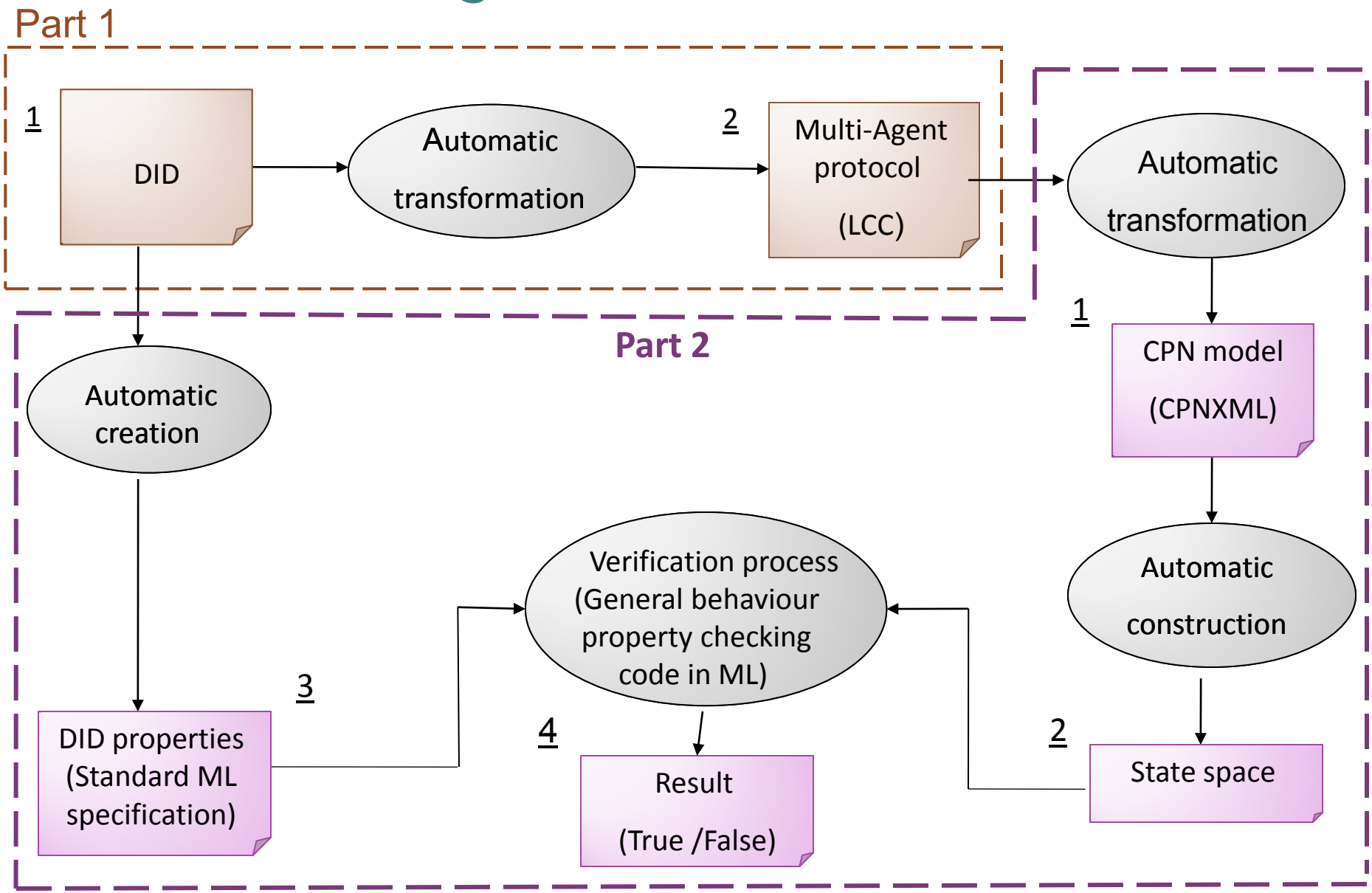
Model Checking

- Our model checking is used to check the semantics of the AIF specification against the semantics of the LCC specification.



- Our architecture is general:
  - AIF (DID)
  - Patterns

Model Checking



Conclusion

- This research describes an approach to bridging the gap between argument specification and multi-agent implementation using AIF as an example of an argumentation language and LCC as an example of a multi-agent implementation (coordination) language.

- The proposed approach uses a combination of transformational synthesis and model checking steps to automatically transform the new specification argument language (DID) to peer-to-peer protocols by using a LCC techniques-based synthesis method.

- The resulting LCC protocol is validated to prove that the LCC techniques-based synthesis method is correctly implemented and to demonstrate that the resulting LCC protocol derived from DID behaves correctly as expected.