A Logic Based Language for Engineering Agent Applications

Keith L Clark
Imperial College
University of Queensland
(joint work with Peter J Robinson, UQ)
Key features of QuLog

- **Dynamic relations** – an agent’s current beliefs
  - Defined solely by updatable facts

- **Static Relations** – an agent’s knowledge
  - Conditional rules, committed choice rules, and non-updatable facts.
  - More declarative and concise than Prolog – closer to predicate logic
    - Evaluable expressions as call arguments
    - Including list and set comprehension expressions
  - Non-deterministic pattern matching support for string + list processing

- **Functions** – syntactic sugar for functional relations
  - Committed choice rewrite rules
  - Relation queries as commitment tests
  - All functions must be named – no lets or lambdas
  - Strict evaluation

- **Actions** – an agent’s behavioral knowledge
  - Committed choice action rules
  - Dynamic fact updates, thread forking, I/O and message communication
  - Actions may call functions and relations but not vice versa
Key features continued

- **Multi-threaded** - threads execute action calls
- **Type declarations** - required for functions
- **Moded type declarations** - required for relations and actions
- **Modes specify** which arguments:
  - must be ground - the default mode optional ! annotation
  - will be ground after a successful call - ? annotation
  - may not be ground after a successful call - ?? annotation
  - will be unchanged - @ annotation
  - Hybrid modes e.g. list(?nat)
- **New types can be defined**. Union types allowed.
  - Primitive type atomic is union of nat, int, num, atom, string
- **Run-time test for any type**
- **Enforced type and mode safe** for meta-level and remote querying
- **Function call arguments** must all be ground at time of the call
QuLog Information Agent

Fixed Knowledge Rules & Facts
Dynamic Belief Facts

Updates

info@host
Message Handling Thread

Error Handling Thread for Responses to Remote Queries

Temporary Action Thread for a Query

Query Thread Monitor

fork

kill

pedro

Queries and info from other agents
QuLog Pro-active Reasoning Agent

Knowledge Rules & Facts + Belief Facts

Updates

Error Handling Thread for Responses to Remote Queries

Permanent Action Thread Monitoring Beliefs Inferring & Informing

Temporary Action Thread For received Query

Query Thread Monitor

Queries and info from other agents using a shared ontology

pedro

kill
QuLog Agents

- **Agents** are named *multi-threaded QuLog processes*
- Names are registered with a Pedro communication server
- One thread is designated default message handler
- Threads share the Beliefs and Knowledge - no sharing of variables
- **Internal thread co-ordination**
  - using *atomic updates* of beliefs and *suspendable queries* to the beliefs
  - Internal messages
- **Agent co-ordination** via either:
  - *Addressed messages* routed via the Pedro server
  - *Notifications* sent to the Pedro server forwarded to agents that have lodged a *covering subscription*
The Glue - Pedro Communication Server

- Primarily a **pub/sub server**
- Agents publish *notifications* – as QuLog terms
- Routed to *all* other agents with a *current* Pedro *subscription* that *covers* the notification
- The **dynamic subscriptions** have the form: $NTrmPtn :: Conds$
- A subscription covers a notification term $NTrm$ if $NTrmPtn = NTrm \& Conds$ holds
- Pedro *remembers* subscriptions *but not* notifications
  - a content based *router* *not* a *blackboard*
- Can handle 10,000 notifications per second with 1000 subscriptions
Dynamic Relations

dyn $r(t_1,\ldots,t_k)$
$r(a_1,\ldots,a_k)$  % all ground facts
....
Static Relations

rel $s(mt_1, \ldots, mt_k)$ \% moded type expressions

$s(a_1, \ldots, a_k)$ \% any term arguments
$s(a_1, \ldots, a_k) \leq Conds$ \% uncommitted clauses

$s(a_1, \ldots, a_k) :: Test$ \% committed clauses
$s(a_1, \ldots, a_k) :: Test \leq Conds$
Functions

fun \( f(t_1, \ldots, t_k) \rightarrow t \)

\( f(a_1, \ldots, a_k) \rightarrow \text{Exp} \) % all committed rules

\( f(a_1, \ldots, a_k) :: \text{Test} \rightarrow \text{Exp} \)
Actions

\[ \text{act } a(m_a,..,mt_k) \text{ } \% \text{ moded type expressions} \]

\[ a(a_1,..,a_k) \text{ } \% \text{ all committed rules} \]
\[ a(a_1,..,a_k) \]
\[ a(a_1,..,a_k) \vdash \text{ Test } \rightarrow \text{ Act}_1 ; \ldots ; \text{ Act}_k \]
Negated and quantified conditional tests

• Negated tests: not exists \( EVars \) Predication

  \[
  \text{person}(P) & \text{not exists } C,A (\text{child}\_of(C,P) & \text{age}(C,A) & A>18)
  \]

• Quantified conditional tests:
  \[
  \text{forall } AVars (EVars1 \text{ Conj1 } \implies \text{exists } EVars2 \text{ Conj2})
  \]

  \[
  \text{forall } C,A (\text{child}\_of(C,P) & \text{age}(C,A) \implies \text{not } A > 18)
  \]

  or stronger condition:

  \[
  \text{forall } C,A (\text{child}\_of(C,P) \implies \text{exists } A \text{ age}(C,A) & \text{not } A > 18)
  \]
Set and list comprehension expressions

\{ \text{Vars} :: \exists \text{EVars Conj} \} \quad \text{[Vars :: \exists \text{EVars Conj}]} \quad 

e.g. \{ \text{A :: \exists} \text{C age_of(C,A)} \& \text{A>17} \} 

Set of ages over 17 of recorded children, no duplicates

\text{[A :: \exists} \text{C age_of(C,A)} \& \text{A>17]} 

List of ages over 17 of recorded children, poss. duplicates

Set and list expressions \text{must have} ground values
Remote Queries

\[ r(\ldots) \leq C_1 \land \\
(TypedVars :: Conds) \text{ query\_to Ag} \land \\
C_2 \land \\
\ldots \]

Action handing of received remote query in message handling thread of Ag

Query query\_from Client;
respond\_remote\_query(Query, Client)
Watch debugging

• No step by step query tracing
• Instead a watch can be placed on any number of relations, functions and actions
• Log displayed each time a watched def. is used
• Invisible writes are inserted in the rules
Source

http://staff.itee.uq.edu.au/pjr/HomePages/QulogFiles/qulog0.9.tgz