Reasoning about Knowledge in Distributed Systems Using Datalog

Matteo Interlandi

University of Modena and Reggio Emilia

Datalog 2.0 Workshop - 11 September 2012, Wien

Motivations

Preamble

The Knowledge Model

Knowlog

Conclusions

Motivations

- Why use Datalog to program distributed systems?
  - Conciseness [1]
  - Executable programs generated directly from high-level specifications [1]
  - Database techniques applied to distributed systems [1, 3]
  - Matching between implementation and specification properties [1, 2]
  - …

Motivations Cont'd

- **BUT**
  - Still something is missing: the capability to express what a node **knows**
    - We are able to think about what a node **knows** and not about communication details
    - Specifications become more intuitive and therefore less error-prone
    - Nice formalization for both data and code communication
    - Separation between functional and non-functional properties
- **Knowlog**: Datalog leveraged with epistemic modal operators for designing distributed systems

- ✓ Motivations
- Preamble
- The Knowledge Model
- Knowlog
- Conclusions

- ✓ Motivations
- **Preamble**
- The Knowledge Model
- Knowlog
- Conclusions

Preamble

- Based on Dedalus [4]
- **BUT**
  - No asynchronous rules
    - We want to push non-functional properties outside the logic
    - In the future we will investigate how non functional properties affect the logic
  - We use *accessible* relations as communication means
    - We want to restrict the set of relations used to transmit facts
    - More close to data integration approaches [5]

Dedalus^K: Datalog in Time and Space

- Datalog with a notion of time…
  - Tuples by default are *ephemeral*: they exist just in one time-step
  - Tuples can be persisted using frame rules
  - Multiple instances $\mathbf{I}$[n], one for each time-step n
  - Two sets of rules: *Deductive* and *Inductive* [4]
- …and space
  - A set of *accessible* relations partitioned among nodes
  - Each *adb* relation contains a **location specifier term** [6]
  - Facts are exchanged using *adb* relations by specifying the desired location

- ✓ Motivations
- ✓ Preamble
- The Knowledge Model
- Knowlog
- Conclusions

- ✓ Motivations
- ✓ Preamble
- **The Knowledge Model**
- Knowlog
- Conclusions

States, Runs and Systems

- The **local state** s_i of node i is defined by the tuple $(\mathcal{P}_\mathrm{i}, \mathbf{I}_\mathrm{i})$ where $\mathcal{P}_\mathrm{i}$ is the program of node i and $\mathbf{I}_\mathrm{i}$ is an instance over $\mathcal{P}_\mathrm{i}$
- A **global state** g is a tuple in the form $(s_1, \ldots, s_n)$ where s_i is the node i's state
- A **run** is a function that binds time values to global states:
  - $r : \mathbf{N} \rightarrow G$ where $G = \{S_1, \ldots, S_n\}$ with S_i the set of possible local states for node i
  - Given a run r and a time t, the tuple (r, t) is referred as a **point**
- A **system** $\mathcal{S}$ is a non empty set of runs
- An **interpreted system** is a tuple $(\mathcal{S}, \pi)$ with $\mathcal{S}$ a system and π an interpretation

Knowledge in Distributed Systems [7]

- **Situation to model:** “for what node *i* knows, the system could be at point where ψ is true”
  - Knowledge is determined by *i*’s local state
  - *i* cannot distinguish two point in the system in which it has the same local state
    - Given two points with global states respectively *g* and *g′* and an *indistinguishable* relation $\sim_i$, $g \sim_i g'$ if node *i* has the same local state both in *g* and *g′*
  - An interpreted system can be modeled using a *Kripke* structure

$$\mathcal{M} = (W, \mathcal{A}_1, \ldots, \mathcal{A}_n, D, \pi)$$

with *W* the set of possible global states, $\mathcal{A}_i = \sim_i$, *D* the domain and π an interpretation

- Assumption: D is the same in every possible world

The modal operator K [7]

- Given a Kripke structure $\mathcal{M}$, a world $w \in W$ and a valuation v on $\mathcal{M}$, the *satisfaction relation* for a formula ψ is:
  - $(\mathcal{M}, w, v) \models R(t_1, \ldots, t_n)$ iff $(v(t1), \ldots, v(tn)) \in \pi(w)(R)$
  - $(\mathcal{M},w,v) \models \neg\psi$ iif $(\mathcal{M},w,v) \not\models \psi$
  - $(\mathcal{M},w,v) \models \psi \wedge \phi$ iff$(\mathcal{M},w,v) \models \psi$ and $(\mathcal{M},w,v) \models \phi$
  - $(\mathcal{M},w,v) \models \forall \psi$ iif $(\mathcal{M},w,v[x/a]) \models \psi$ for every $a \in U$
  - $(\mathcal{M},w,v) \models K_i\psi$ iff $(\mathcal{M},u,v) \models \psi$ for all u such that $(w,u) \in \mathcal{A}_i$
- The modal operator K_i express what a node i "knows"

The Axiom System S5 [7]

- The definition of knowledge has the **S5** properties
  1. Distributed Axiom: $\models (K_i\psi \wedge K_i(\psi \rightarrow \phi)) \rightarrow K_i\phi$
  2. Knowledge Generalization Rule: For all structures $\mathcal{M}$, if $\mathcal{M} \models \psi$ then $\mathcal{M} \models K_i\psi$
  3. Truth Axiom: $\models K_i\psi \rightarrow \psi$
  4. Positive Introspection Axiom: $\models K_i\psi \rightarrow K_iK_i\psi$
  5. Negative Introspection Axiom: $\models \neg K_i\psi \rightarrow K_i\neg K_i\psi$

- ✓ **Motivations**
- ✓ **Preamble**
- ✓ **The Knowledge Model**
- **Knowlog**
- **Conclusions**

- ✓ Motivations
- ✓ Preamble
- ✓ The Knowledge Model
- **Knowlog**
- Conclusions

Knowlog^K

- A rule in Knowlog^K has the form:

$$\Box(H \leftarrow B_1,\ldots, B_n).$$

with each literal in the form ΔR.

  - ○ Symbols ☐ and Δ denoting a (possibly empty) sequence of modal operators *K.*
  - ○ ☐ is called *modal context* and is used to assign to each node, the rules the node is responsible for
  - ○ A ***communication rule*** has no modal context, but every body atom is in the form $K_i\Delta R$, while head atom has the form $K_j\Delta R'$, with $i \neq j$.

An Example: the Two Phase Commit

- Inspired by [8]
- Phases:
  - voting phase - the coordinator submits to all the transaction's participants the willingness to perform a distributed commit. Each participant sends a vote to the coordinator
  - decision phase - the coordinator collects all votes and decides if performing global commit or abort. The decision is then issued to the participants
- Assumption:
  - No failures
  - No time-out actions

The 2PC Coordinator

\\Initialization

r1: K_C(log(Tx_id,State)@next:-log(Tx_id,State)).

r2: K_C(part_cnt(count<N>):-participants(N)).

r3: K_C(start_transaction(Tx_id):-log(Tx_id,State),State==“Vote-req”,
¬log(Tx_id,State_2),State_2!=“Vote-req”).

r4: K_C(transaction(Tx_id,State):-start_transaction(Tx_id),log(Tx_id,State)).

r5: $K_C$participants(p1).

r6: $K_C$participants(p2).

\\Decision Phase

r7: K_C(yes_cnt(Tx_id,count<Part>):-vote(Vote,Tx_id,Part),Vote == "yes").

r8: K_C(log(Tx_id,"commit”):-part_cnt(C),yes_cnt(Tx_id,C1),C==C1,
State=="vote-req”,transaction(Tx_id,State)).

r9: K_C(log(Tx_id,"abort"):-vote(Vote,Tx_id,Part),Vote == "no",
transaction(Tx_id,State), State =="vote-req").

\\ Communication

r10: $K_X$transaction(Tx_id, State):-$K_C$participants(X), $K_C$transaction(Tx_id,State).

The 2PC Coordinator

\\Initialization

r1: K_C(log(Tx_id,State)@next:-log(Tx_id,State)).

r2: K_C(part_cnt(count<N>):-participants(N)).

r3: K_C(start_transaction(Tx_id):-log(Tx_id,State),State==“Vote-req”,
¬log(Tx_id,State_2),State_2!=“Vote-req”).

r4: K_C(transaction(Tx_id,State):-start_transaction(Tx_id),log(Tx_id,State)).

r5: $K_C$participants(p1).

r6: $K_C$participants(p2).

\\Decision Phase

r7: K_C(yes_cnt(Tx_id,count<Part>):-vote(Vote,Tx_id,Part),Vote == "yes").

r8: K_C(log(Tx_id,"commit”):-part_cnt(C),yes_cnt(Tx_id,C1),C==C1,
State=="vote-req”,transaction(Tx_id,State)).

r9: K_C(log(Tx_id,"abort"):-vote(Vote,Tx_id,Part),Vote == "no",
transaction(Tx_id,State), State =="vote-req").

\\ Communication

r10: $K_X$transaction(Tx_id, State):-$K_C$participants(X), $K_C$transaction(Tx_id,State).

The 2PC Coordinator

\\Initialization

r1: K_C(log(Tx_id,State)@next:-log(Tx_id,State)).

r2: $K_($part_cnt(count<N>):-participants(N))

r3: $K_($ … e-req”).

r4: $K_($ … .,State)).

r5: $K_($

r6: $K_($

\\

r7: K … yes").

r8: K … ate)).

r9: K … transaction(Tx_id,State), State == "vote-req").

\\ Communication

r10: $K_X$transaction(Tx_id, State):-$K_C$participants(X), $K_C$transaction(Tx_id,State).

Knowledge-Oriented Programming [9]

- Modal operators for expressing:
  - State of knowledge
  - Actions

The 2PC Coordinator

\\Initialization

r1: K_C(log(Tx_id,State)@next:-log(Tx_id,State)).

r2: K_C(part_cnt(count<N>):-participants(N)).

r3: K_C(start_transaction(Tx_id):-log(Tx_id,State),State==“Vote-req”,
¬log(Tx_id,State_2),State_2!=“Vote-req”).

r4: K_C(transaction(Tx_id,State):-start_transaction(Tx_id),log(Tx_id,State)).

r5: $K_C$participants(p1).

r6: $K_C$participants(p2).

\\Decision Phase

r7: K_C(yes_cnt(Tx_id,count<Part>):-vote(Vote,Tx_id,Part),Vote == "yes").

r8: K_C(log(Tx_id,"commit”):-part_cnt(C),yes_cnt(Tx_id,C1),C==C1,
State=="vote-req”,transaction(Tx_id,State)).

r9: K_C(log(Tx_id,"abort"):-vote(Vote,Tx_id,Part),Vote == "no",
transaction(Tx_id,State), State =="vote-req").

\\ Communication

r10: $K_X$transaction(Tx_id, State):-$K_C$participants(X), $K_C$transaction(Tx_id,State).

The 2PC Coordinator (with syntactic sugar)

#Program initialization @C

r1: log(Tx_id,State)@next:-log(Tx_id,State).

r2: part_cnt(count<N>):-participants(N).

r3: start_transaction(Tx_id):-log(Tx_id,State),State==“Vote-req”,
¬log(Tx_id,State_2),State_2!=“Vote-req”.

r4: transaction(Tx_id,State):-start_transaction(Tx_id),log(Tx_id,State).

r5: participants(p1).

r6: participants(p2).

#Program decisionPhase @C

r7: yes_cnt(Tx_id,count<Part>):-vote(Vote,Tx_id,Part),Vote == "yes”.

r8: log(Tx_id,"commit”):-part_cnt(C),yes_cnt(Tx_id,C1),C==C1,
State=="vote-req”,transaction(Tx_id,State).

r9: log(Tx_id,"abort"):-vote(Vote,Tx_id,Part),Vote == "no",
transaction(Tx_id,State), State =="vote-req”.

\\ Communication

r10: $K_X$transaction(Tx_id, State):-$K_C$participants(X), $K_C$transaction(Tx_id,State).

Operators for Knowledge in Group of Nodes [7]

- Given a non empty set of nodes G
  - $(\mathcal{M},w,v) \models E_G\psi$ iff $(\mathcal{M},w,v) \models K_i\psi$ for all $i \in G$
  - $(\mathcal{M},w,v) \models D_G\psi$ iff $(\mathcal{M},u,v) \models \psi$ for all u that are $(w,u) \in \bigcap_{i\in G} R_i$
- The Knowledge Axiom, Distribution Axiom, Positive Introspection Axiom, and Negative Introspection Axiom hold also for E_G and D_G
- In addition:
  - $\models D_{\{i\}}\psi \leftrightarrow K_i\psi$
  - $\models D_G\psi \rightarrow D_{G'}\psi$ if $G \subseteq G'$

Incorporating Higher level of Knowledge: Knowlog

r8: K_C(log(Tx_id,"commit”):-$E_X$vote(“yes”,Tx_id), participants(X),
State=="vote-req”,transaction(Tx_id,State)).

r9: K_C(log(Tx_id,"abort"):-$D_x$vote(Vote,Tx_id),Vote == "no",participants(X),
transaction(Tx_id,State), State =="vote-req").

- E_G is used when a fact, to be considered true, is correctly replicated in every node $i \in G$
  - in front of communication rules emulates the multicast primitive
  - as a model context
- D_G is employed when facts that are fragmented inside relations distributed in G must be assembled in one place

Reified Form

- Each Knowlog rule is rewritten in its *reified f*orm:
  - each relation contains a **knowledge accumulator term**
  - knowledge operators are pushed into the accumulator term
  - for each accessible relation also the location term is filled accordingly
  - 4 new built-in relations:
    - `⊕(X,Y,Z)` to concatenate epistemic operators
    - `K(X,Y)`, `E(<X>,Y)`, `D(<X>,Y)` to build knowledge accumulator terms
  - if E_G in front of a communication rule, a set of new communication rules is generated, each one with K_i and $i \in G$

Reified Form: Examples

- **Examples**
  - K_A(cursor(Index):-new(Index)) →
    cursor(Ka,Index):-new(Ka,Index)
  - $K_A K_B$vote(Tx_id):-$K_B$vote(Tx_id),$K_B$path(A,B) →
    vote(KaKb,#A,Tx_id):-vote(Kb,#B,Index),path(Kb,#B,A),
    K(B,Kb), K(A,Ka), ⊕(Ka,Kb,KaKb)

  - $E_X$message(Id):-$K_A$info(Id,Value), $K_A$nodes(X) →
    K_1message(Id):-$K_A$info(Id,Value)
    K_2message(Id):-$K_A$info(Id,Value)
    K_3message(Id):-$K_A$info(Id,Value)

…

- ✓ **Motivations**
- ✓ **Preamble**
- ✓ **The Knowledge Model**
- ✓ **Knowlog**
- **Conclusions**

- ✓ Motivations
- ✓ Preamble
- ✓ The Knowledge Model
- ✓ Knowlog
- **Conclusions**

Conclusions

- Knowlog: Datalog + the epistemic modal operators for:
  - high-level specifications of distributed systems
  - communication of data and code
- Future works:
  - Definition of the Knowlog framework
  - Operational semantics, complexity, expressiveness
  - How do nodes “learn”?
  - How do non-functional properties may affect the logic
    - Definition of synchronous and asynchronous systems
  - Proof-of-concept implementation

References

[1] Joseph M. Hellerstein. *The declarative imperative: experiences and conjectures in distributed logic*. In *SIGMOD Rec.* 39, September 2010, 5-19.

[2] T. J. Ameloot, F. Neven, and J. Van den Bussche. *Relational transducers for declarative networking*. In *PODS'11*, Athens, Greece, USA, 283-292.

[3] Wenchao Zhou, Micah Sherr, Tao Tao, Xiaozhou Li, Boon Thau Loo, and Yun Mao. *Efficient querying and maintenance of network provenance at internet-scale*. In *SIGMOD'10*, Indianapolis, Indiana, USA, 615-626.

[4] Peter Alvaro, William R. Marczak, Neil Conway, et al. *Dedalus: datalog in time and space*. *Datalog'10*, Springer-Verlag, Berlin, Heidelberg, 262-281.

[5] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. *Inconsistency tolerance in P2P data integration: An epistemic logic approach*. In *Inf. Syst.*, June 2008, 33, 4-5, 360-384.

References (2)

[6] Boon Thau Loo, Tyson Condie, Inos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. *Declarative networking: language, execution and optimization*. In SIGMOD’06. Chicago, IL, USA, 97-108.

[7] Ronald Fagin, Joseph Y. Halpern, Moshe Y. Vardi, and Yoram Moses. *Reasoning about Knowledge*. 2003, MIT Press, Cambridge, MA, USA.

[8] Peter Alvaro, Tyson Condie, Neil Conway, Joseph M. Hellerstein, and Russell Sears. *I do declare: consensus in a logic language*. *SIGOPS Oper. Syst. Rev.* 43, 4, January 2010, 25-30.

[9] Yoram Moses and Orit Kislev. *Knowledge-oriented programming*. PODC '93, Ithaca, NY, USA, 261-270.

THANKS!!