Deciding Correctness for Simple Transducer Networks

Tom J. Ameloot

Hasselt University &
Transnational University of Limburg

Introduction

Cloud computing, declarative networking

Asynchronous communication

Correctness (determinacy)

Formalizations

Ensuring Correctness

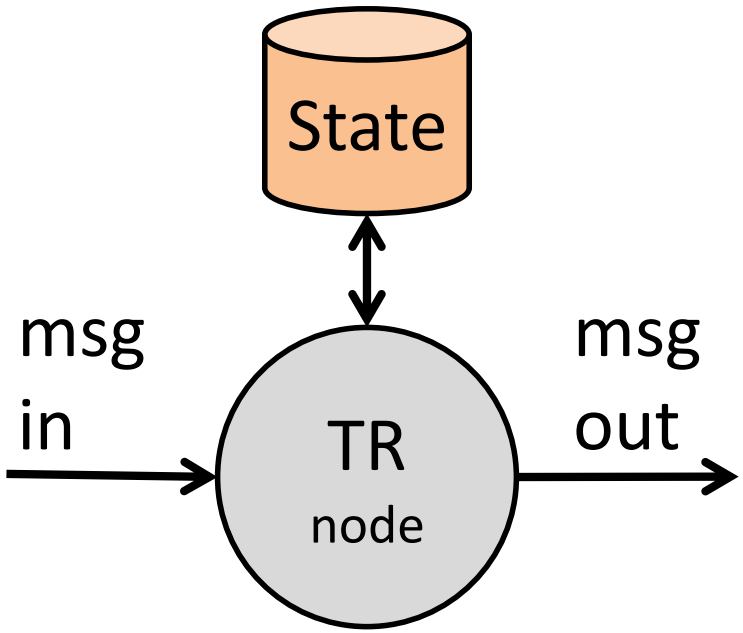
Constructive approach

- enforced through coordination
- emerging correctness (CRDTs, monotone programs)

Decision procedure

Formalize Distributed Programs

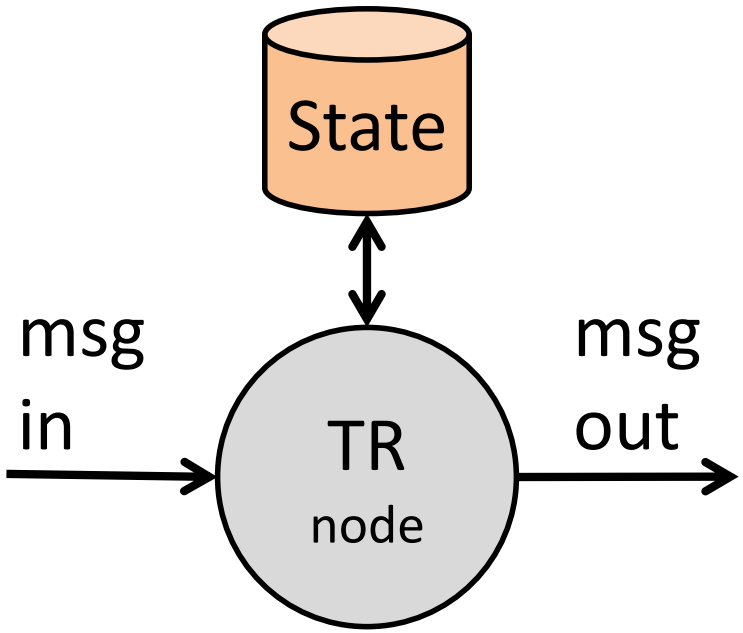
Relational transducer = collection of queries



local query language, e.g. $\text{UCQ}^{\neg}$

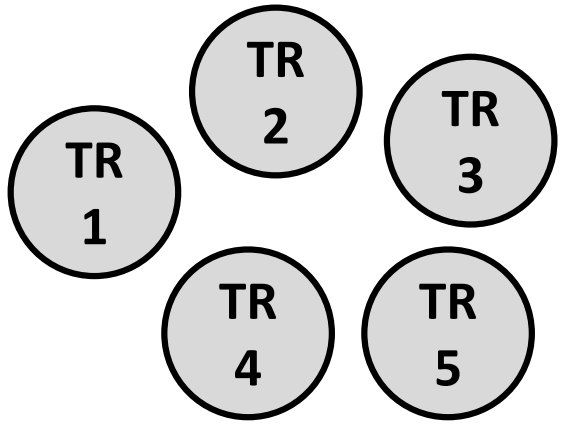
Formalize Distributed Programs

Relational transducer = collection of queries



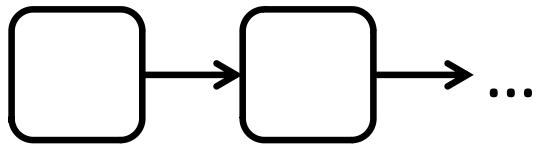
local query language, e.g. UCQ$^{\neg}$

Transducer network



- heterogeneous

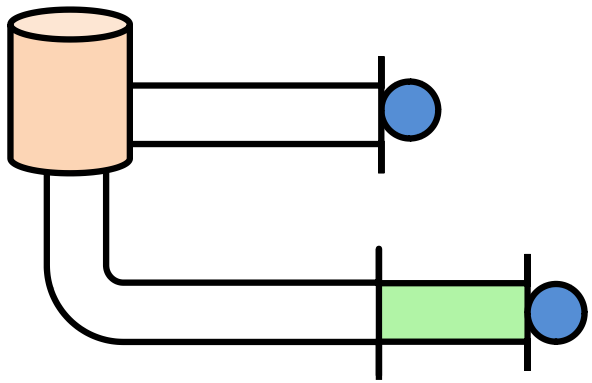
Run



- nondeterministic
- finite
- infinite

Confluence [ICDT 2012]

Formalize correctness as confluence:

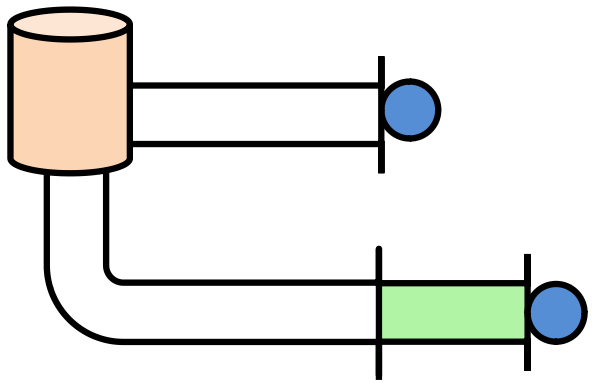


Confluence semantics: collect outputs over all finite runs

Simple transducer networks: UCQ$^\neg$, recursion-free, inflationary, static, message-bounded, message-positive

Confluence [ICDT 2012]

Formalize correctness as confluence:



Confluence semantics: collect outputs over all finite runs

Simple transducer networks: UCQ$^\neg$, recursion-free, inflationary, static, message-bounded, message-positive

Examples (single-node)

Confluent:

$$A_{msg}(u) \leftarrow R(u).$$
$$B_{msg}(u) \leftarrow S(u).$$
$$T(u) \leftarrow A_{msg}(u), B_{msg}(u).$$

message join

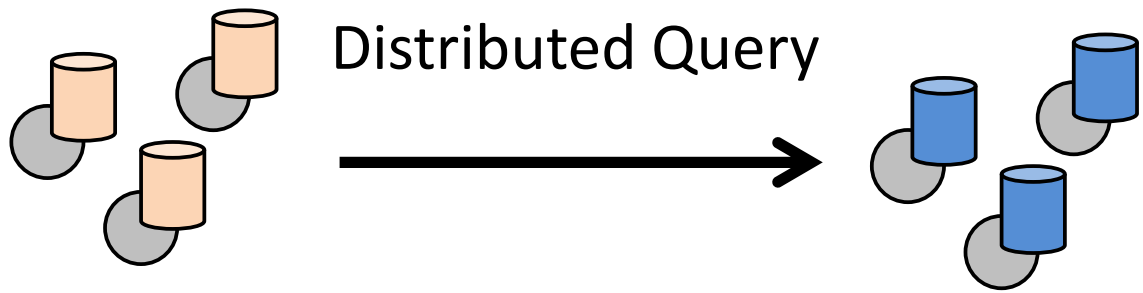
Diffluent:

$$A_{msg}(u) \leftarrow R(u).$$
$$B_{msg}(u) \leftarrow S(u).$$
$$B(u) \leftarrow B_{msg}(u).$$
$$T(u) \leftarrow A_{msg}(u), \neg B(u).$$

Confluence: previous results [ICDT 2012]

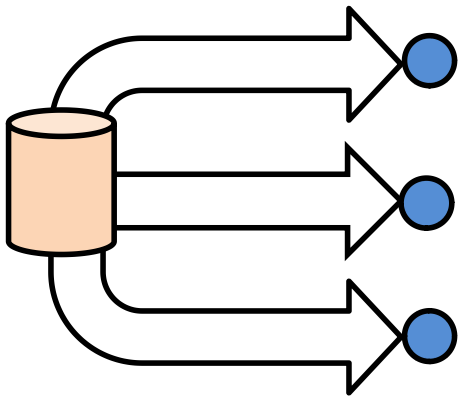
Deciding diffluence for simple transducer networks is NEXPTIME-complete.

Simple transducer networks capture the distributed queries expressible in $\text{UCQ}^{\neg}$.



Consistency

Formalize correctness as consistency:

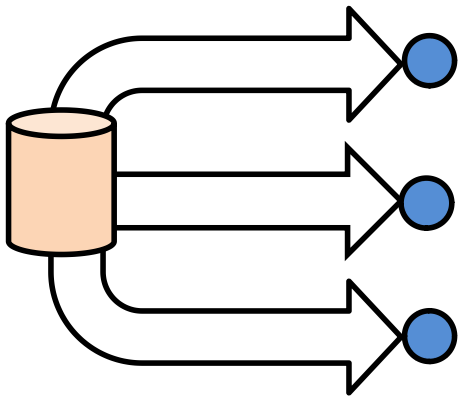


- All **infinite fair** runs ...
- Implies confluence
- More practically relevant

Consistency semantics: output of arbitrary infinite fair run

Consistency

Formalize correctness as consistency:



- All **infinite fair** runs ...
- Implies confluence
- More practically relevant

Consistency semantics: output of arbitrary infinite fair run

Examples (single-node)

Confluent but inconsistent:

$$A_{msg}(u) \leftarrow R(u).$$
$$B_{msg}(u) \leftarrow S(u).$$
$$T(u) \leftarrow A_{msg}(u), B_{msg}(u).$$

Consistent:

$$A_{msg}(u) \leftarrow R(u).$$
$$B_{msg}(u) \leftarrow S(u), A_{msg}(u).$$
$$T(u) \leftarrow B_{msg}(u).$$

message chain, fairness

Consistency: new results

Deciding inconsistency for simple transducer networks is NEXPTIME-complete.

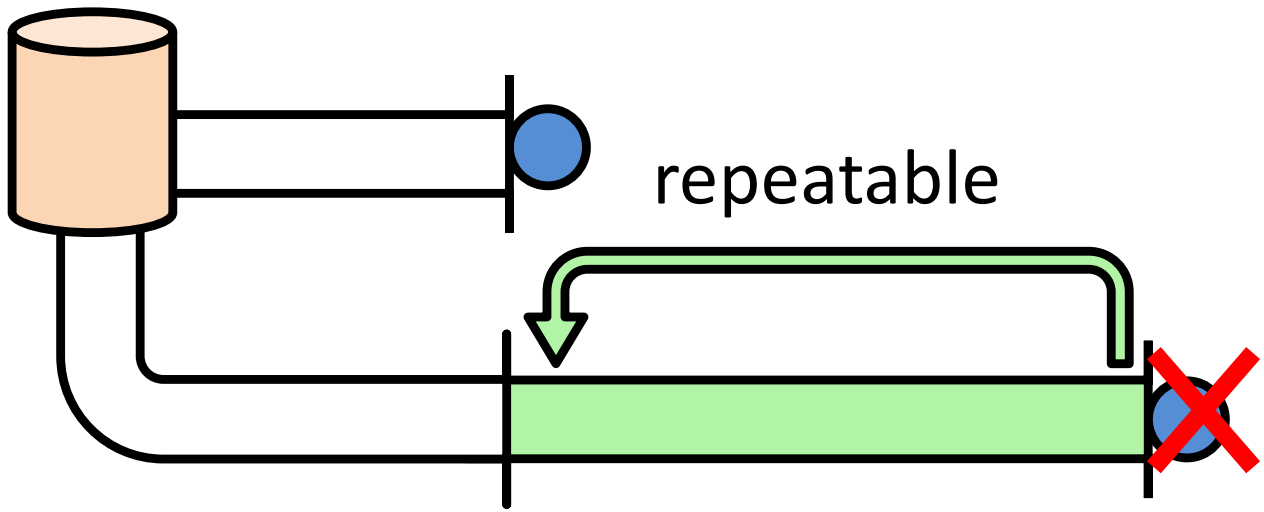
consistency seemed harder

Consistent simple transducer networks, under the consistency semantics, capture all distributed queries expressible in $\text{UCQ}^{\neg}$.

consistency semantics is simpler

Decision Procedure: sketch

Deciding inconsistency
Input: simple transducer network



- If accept: clearly inconsistent
- If inconsistent: projection of infinite runs (difficult)
- NEXPTIME procedure (also lower bound)

Expressivity: sketch

UCQ$^{\neg}$ upper bound through confluence upper bound

Expressivity: sketch

UCQ$^\neg$ upper bound through confluence upper bound

UCQ$^\neg$ lower bound illustrated for distributed query

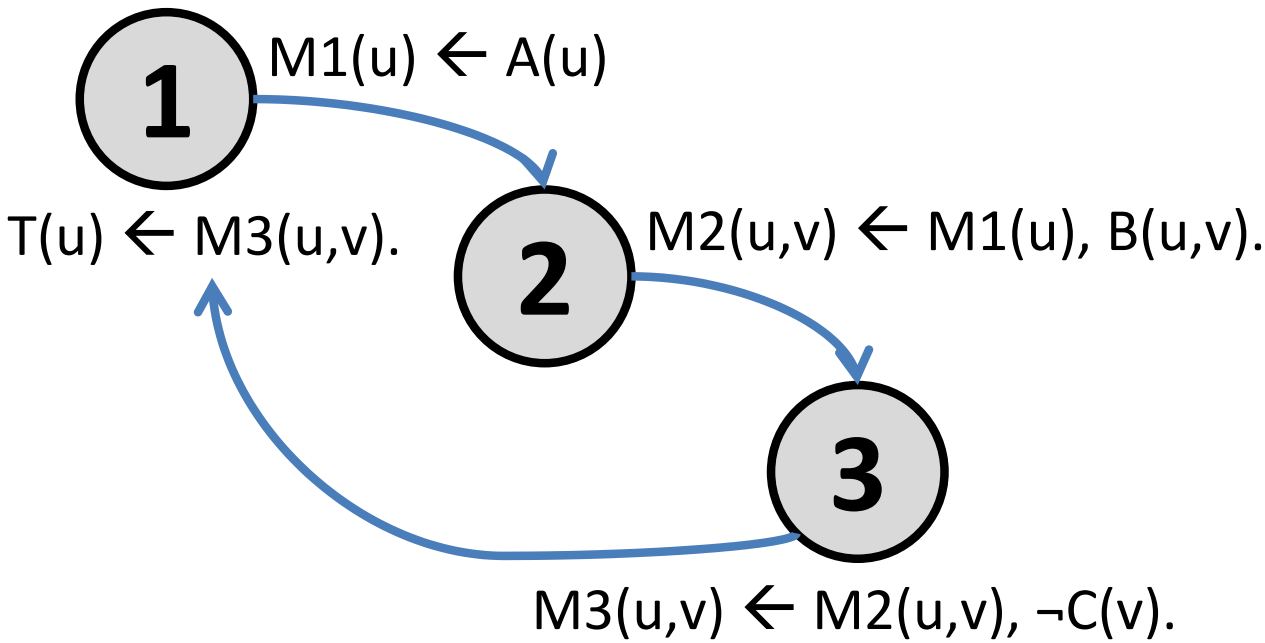
$$1.T(u) \leftarrow 1.A(u), 2.B(u, v), \neg 3.C(v).$$

Expressivity: sketch

UCQ$^\neg$ upper bound through confluence upper bound

UCQ$^\neg$ lower bound illustrated for distributed query

$$1.T(u) \leftarrow 1.A(u), 2.B(u,v), \neg 3.C(v).$$



- recursion-free
- chain tolerates message delays

Directions for Further Work

More complete picture: confluence and consistency

Unfortunate: strong restrictions, yet high complexity

Better:

- constructive approach to correctness
- coordinated vs emerging

Thank you