QONCEPT:

Semantic **Q**uery **O**ptimisation i**n** **CEP** **T**echnologies

Olga Poppe

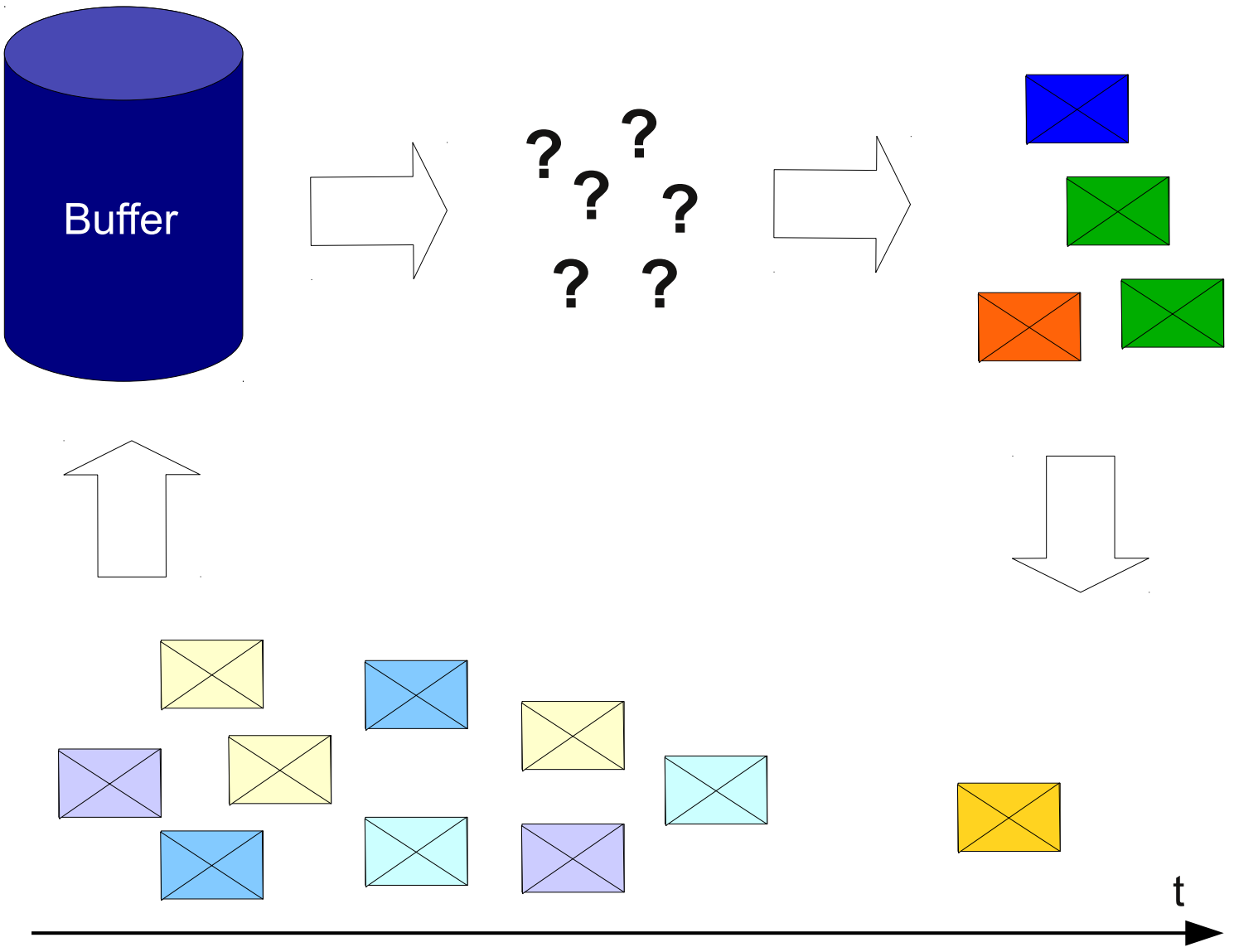
Institute for Informatics, LM**U** Munich, Germany

May 12, 2011

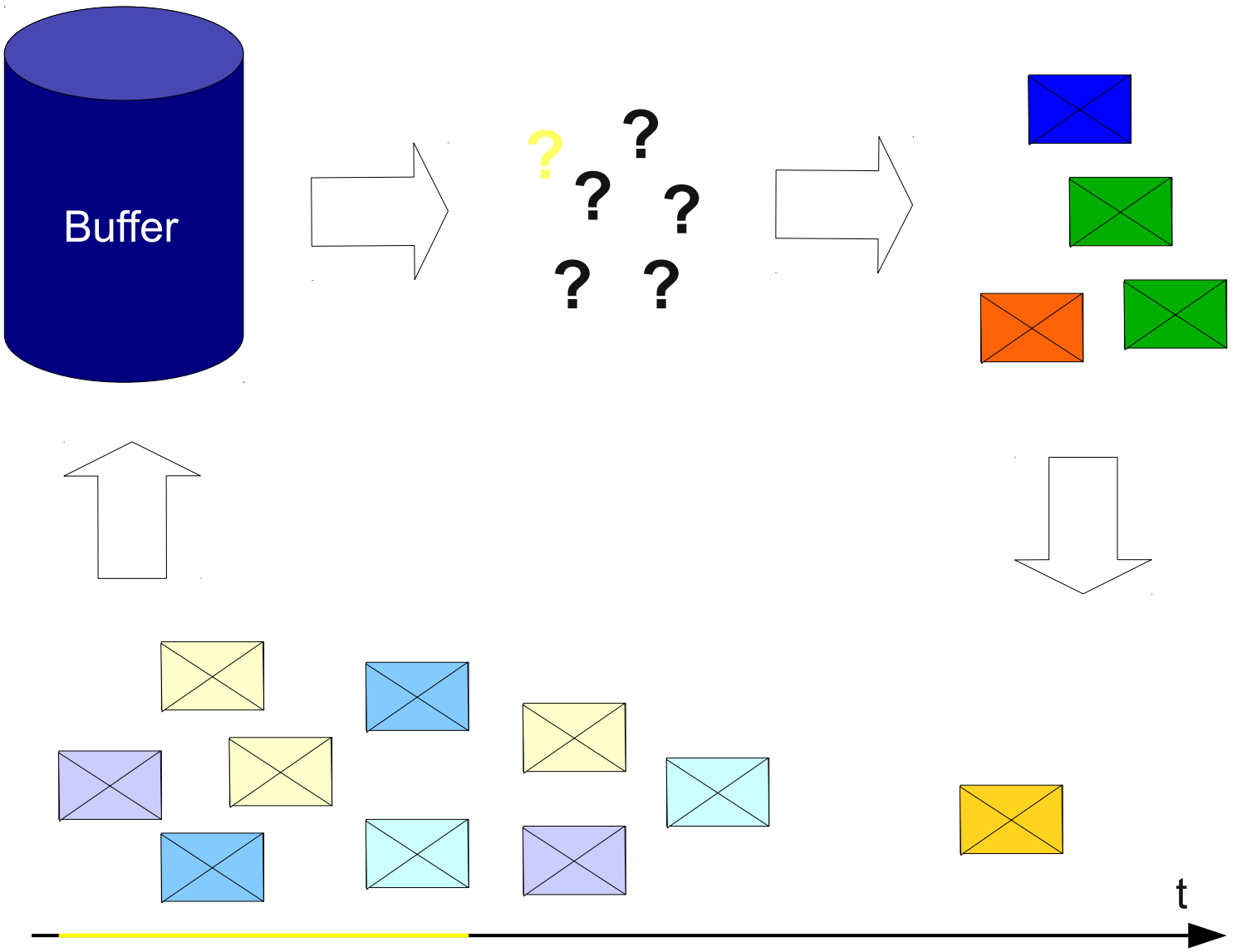
Outline

Outline

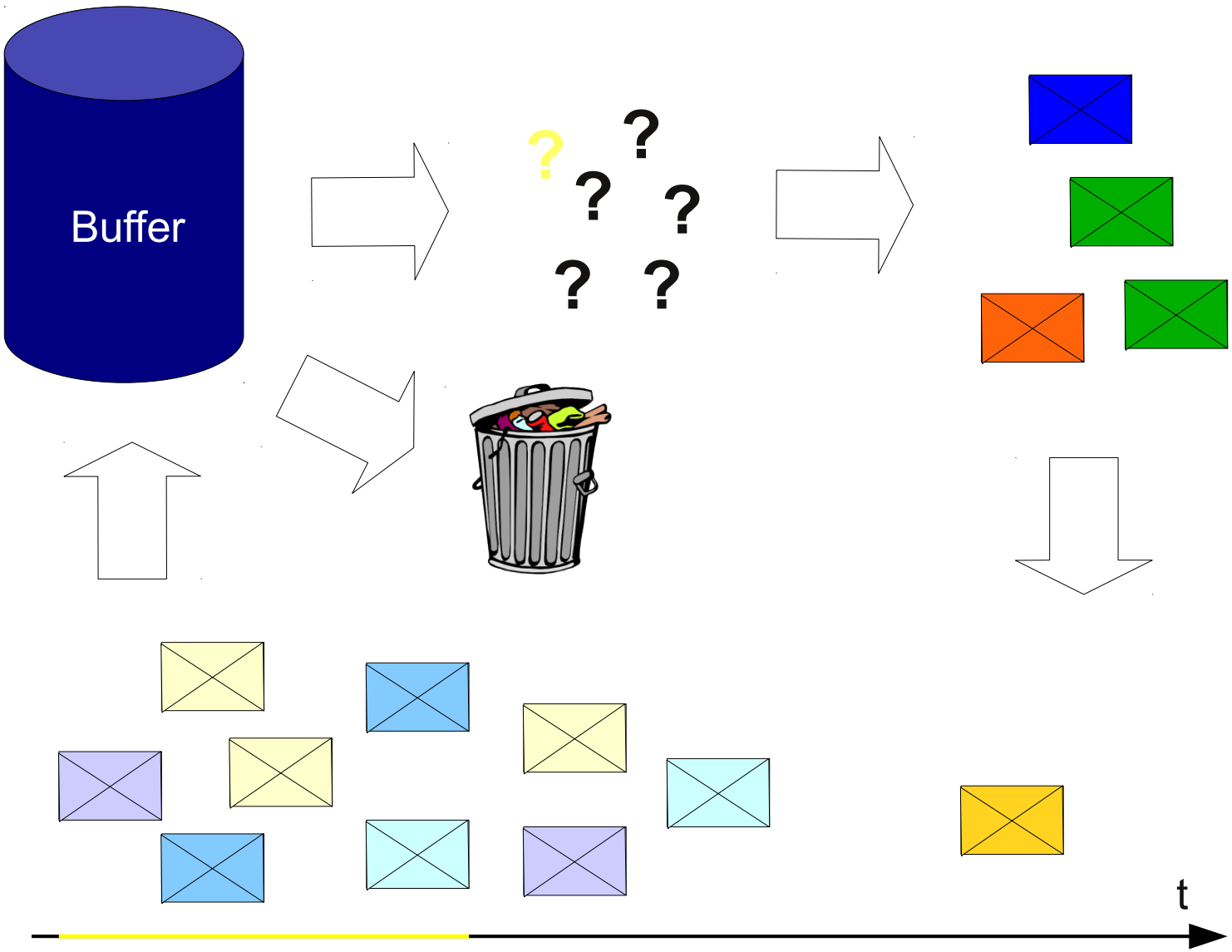
Complex Event Processing



Complex Event Processing



Complex Event Processing



Main Notions

Schema	Chess rules
State	Position of figures
Events	Moves of figures
Application programs	Players
Verification program	Referee
Semantic optimisation	Help players to make best decision(s) at each point of time

Semantic Optimisation (SO)

SO is the use of metadata for query optimisation

Origins of application semantics:

1. Application logic
   - US visa application
   - Auction
   - Trip booking
   - Product order
2. Physical laws
   - Sensor network

Formalism representing application semantics in DB systems:

1. Relation schema + Constraints

DB vs. Warehause vs. CEP

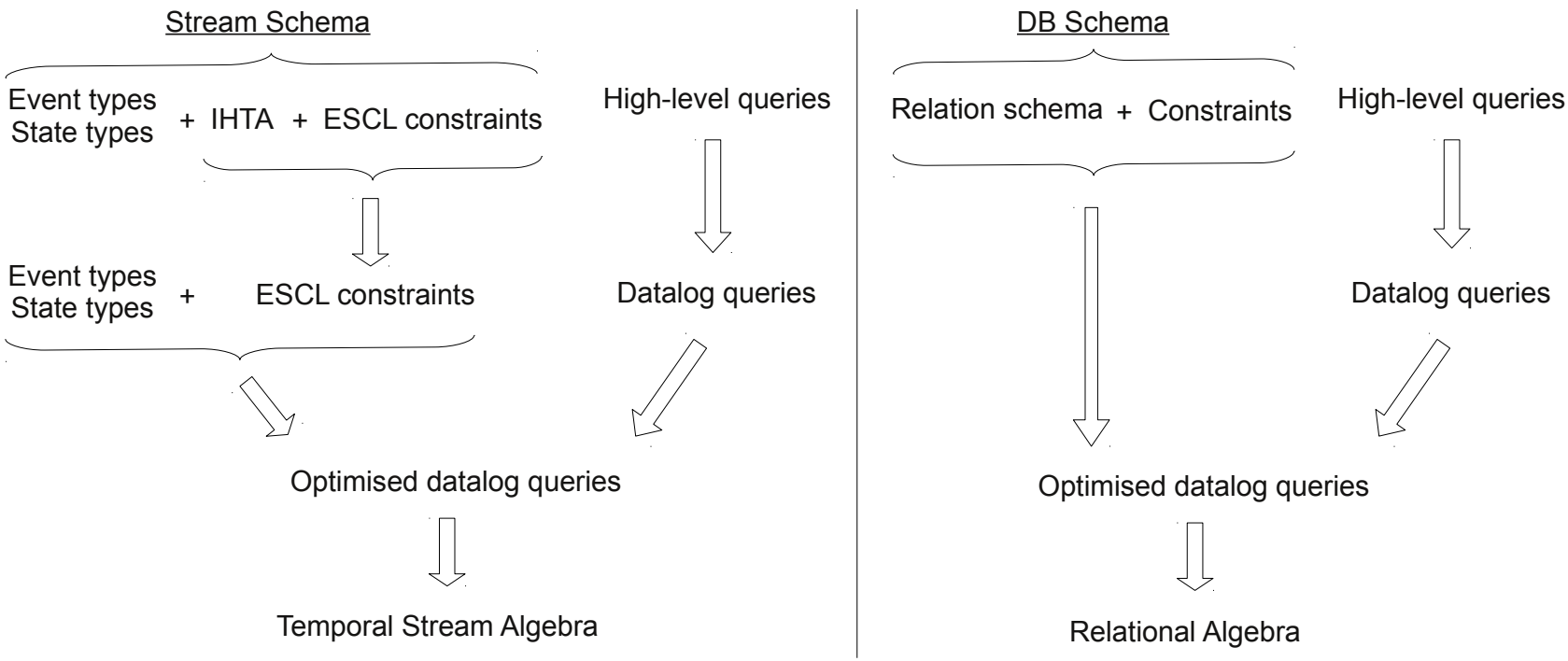
	DB	Data warehouse	CEP
Data	fully available	partially available	partially available
Queries	ad hoc, prepared	standing	standing
Evaluation frequency	unknown	known	known
Continuous evaluation	−	+	+
States	−	−	+

Peculiarities of SO of CEP compared to SO of DB

1. Time dependency

   $\Rightarrow$ Instantiating Hierarchical Timed Automata

   $\Rightarrow$ Event Stream Constraint Language



Peculiarities of SO of CEP compared to SO of DB

1. Time dependency

   $\Rightarrow$ Instantiating Hierarchical Timed Automata

   $\Rightarrow$ Event Stream Constraint Language

2. Complex events (Incremental views)

   $\Rightarrow$ Constraint propagation

3. Standing queries moving data

   $\Rightarrow$ Multi-query optimisation is important

   $\Rightarrow$ Static SO may be expensive

Outline

Main Notions: IHTA

Schema	Chess rules	IHTA
State	Position of figures	State of IHTA
Events	Moves of figures	Input of IHTA
Application programs	Players	CEP programs
Verification program	Referee	Stream verification
SO	Help players to make best decision(s) at each point of time	SO of CEP programs using IHTA

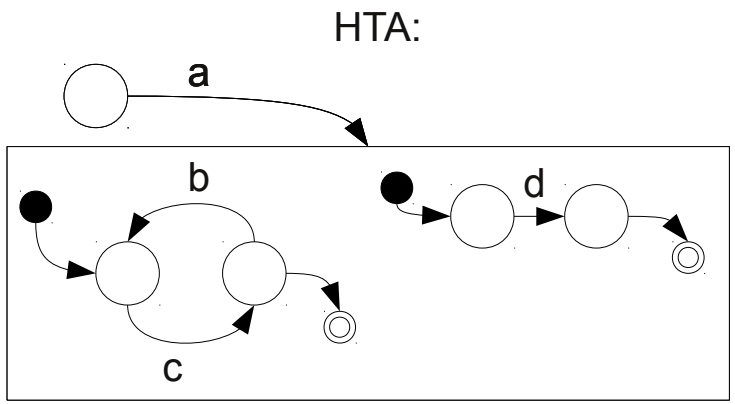
Hierarchical Timed Automata [6]:

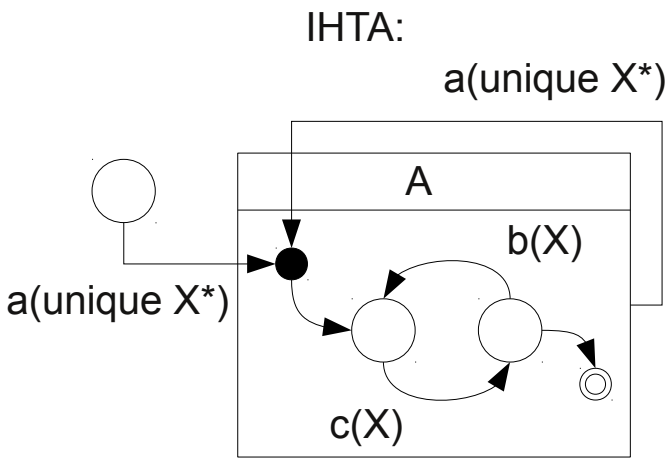
1. Approximate specification
2. Numerous complex temporal and causal relations
3. Modular, means for abstraction, extensible

IHTA vs. HTA

Instantiating Hierarchical Timed Automata ≈ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

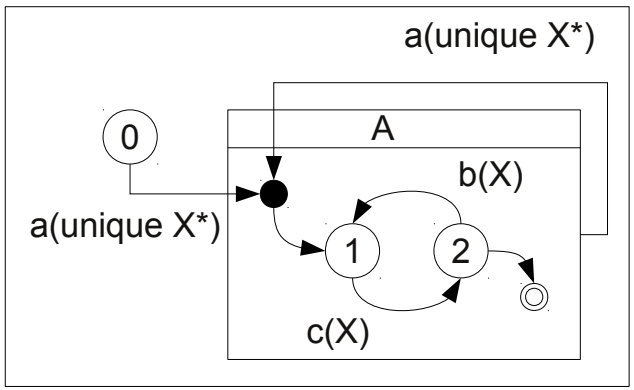




IHTA vs. HTA

Instantiating Hierarchical Timed Automata ≈ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data



Configuration:

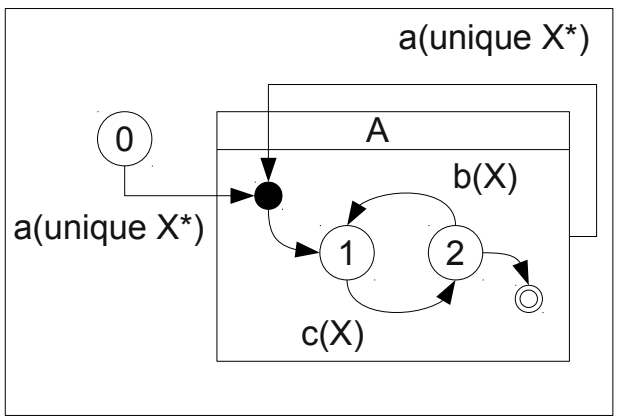
root
State = 0 Variable bindings=... Identifier bindings =...

IHTA vs. HTA

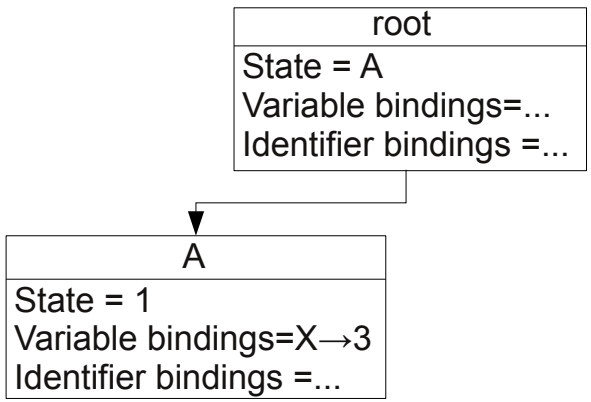
Instantiating Hierarchical Timed Automata ≈ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

Stream: a(3)



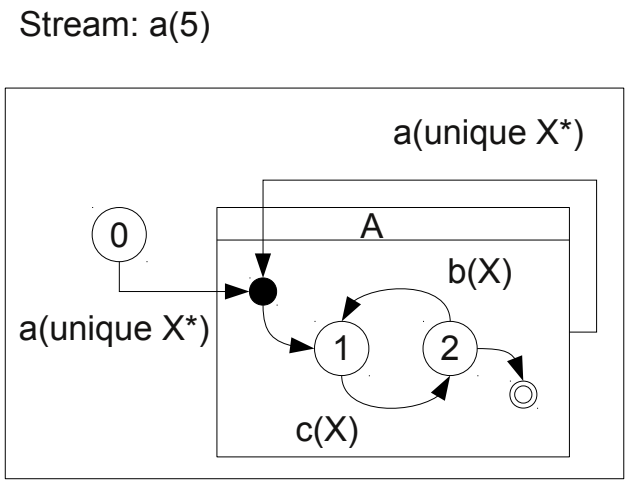
Configuration:

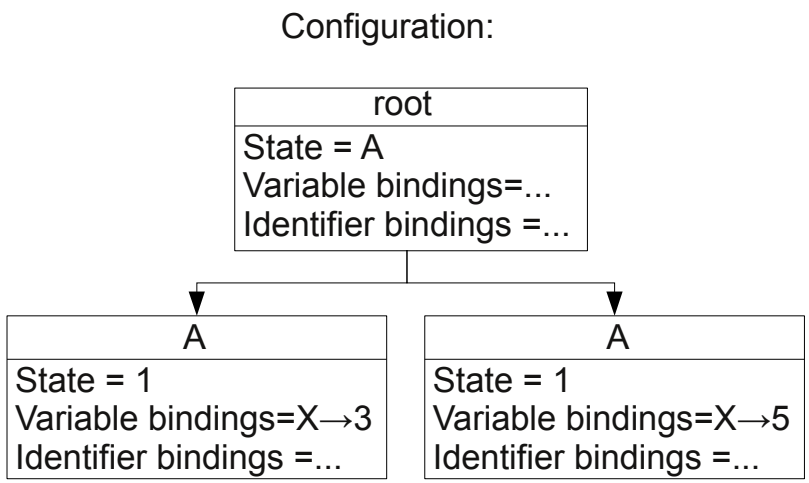


IHTA vs. HTA

Instantiating Hierarchical Timed Automata $\approx$ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

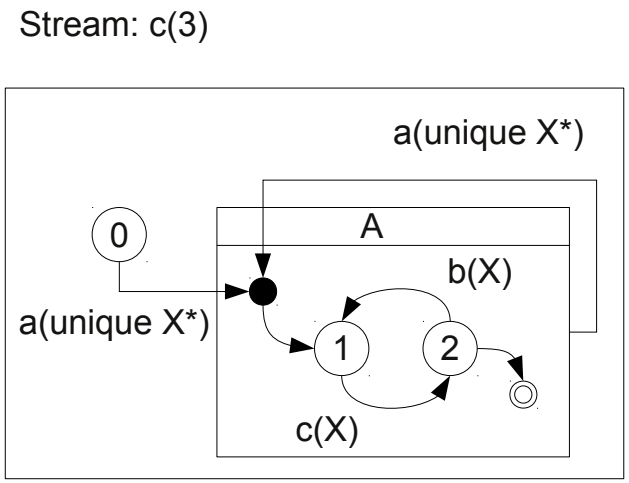


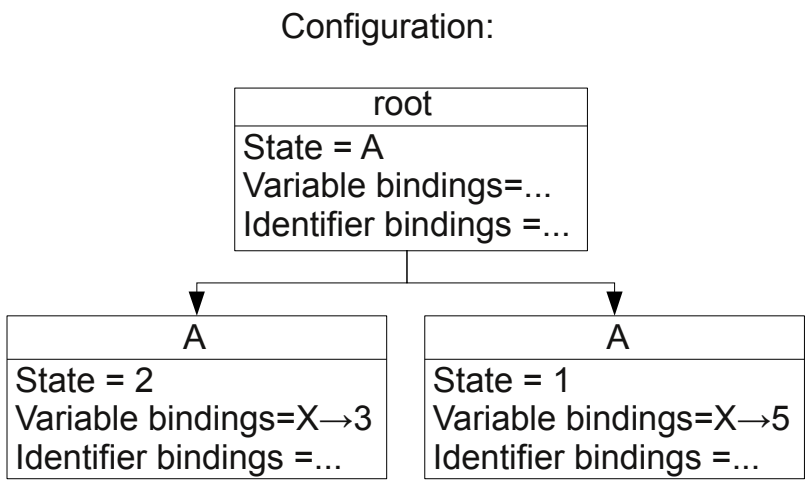


IHTA vs. HTA

Instantiating Hierarchical Timed Automata ≈ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

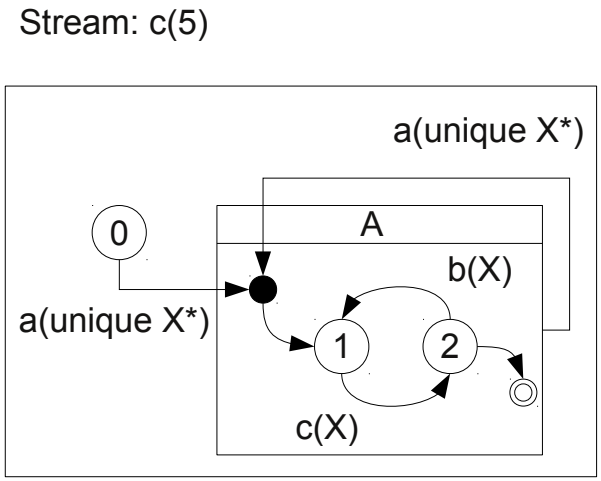


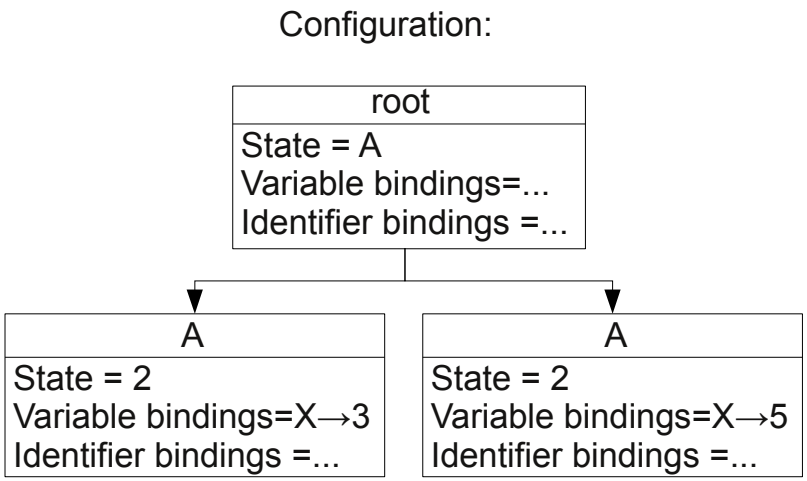


IHTA vs. HTA

Instantiating Hierarchical Timed Automata $\approx$ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

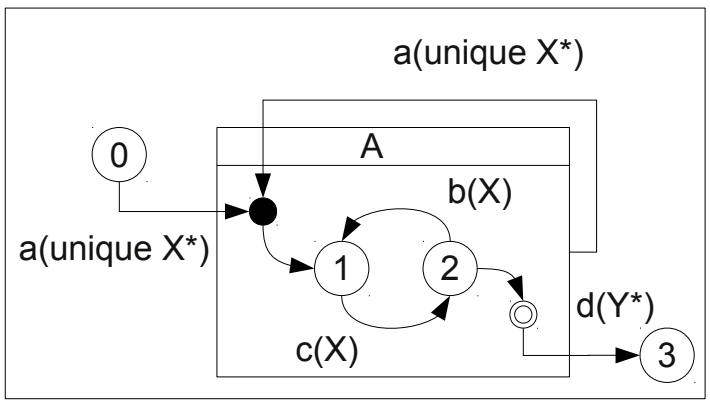




Instantiating Hierarchical Timed Automata ≈ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data

Stream: d(2)

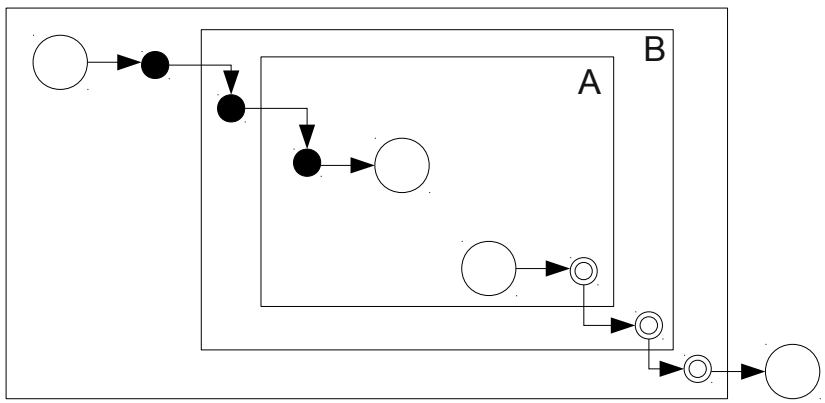


Configuration:

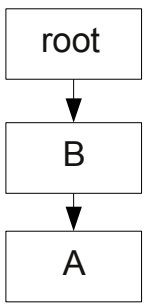
root
State = 3 Variable bindings=Y→2 Identifier bindings =...

Instantiating Hierarchical Timed Automata $\approx$ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data



Configuration:

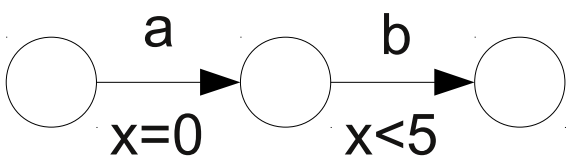


IHTA vs. HTA

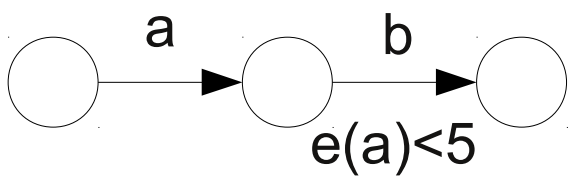
Instantiating Hierarchical Timed Automata $\approx$ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data
3. Temporal constraints on the *beginning* and the end of the occurrence time of any event accepted by the instance

HTA:



IHTA:

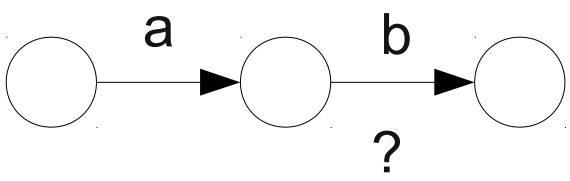


IHTA vs. HTA

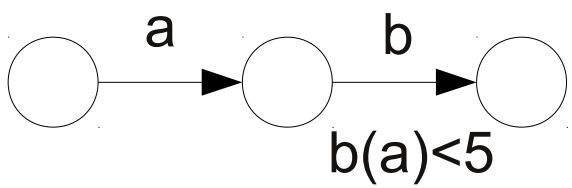
Instantiating Hierarchical Timed Automata $\approx$ HTA +

1. Arbitrary number of concurrent processes
2. Events and states carry data
3. Temporal constraints on the *beginning* and the end of the occurrence time of any event accepted by the instance

HTA:



IHTA:



IHTA vs. HTA

Instantiating Hierarchical Timed Automata $\approx$ HTA +

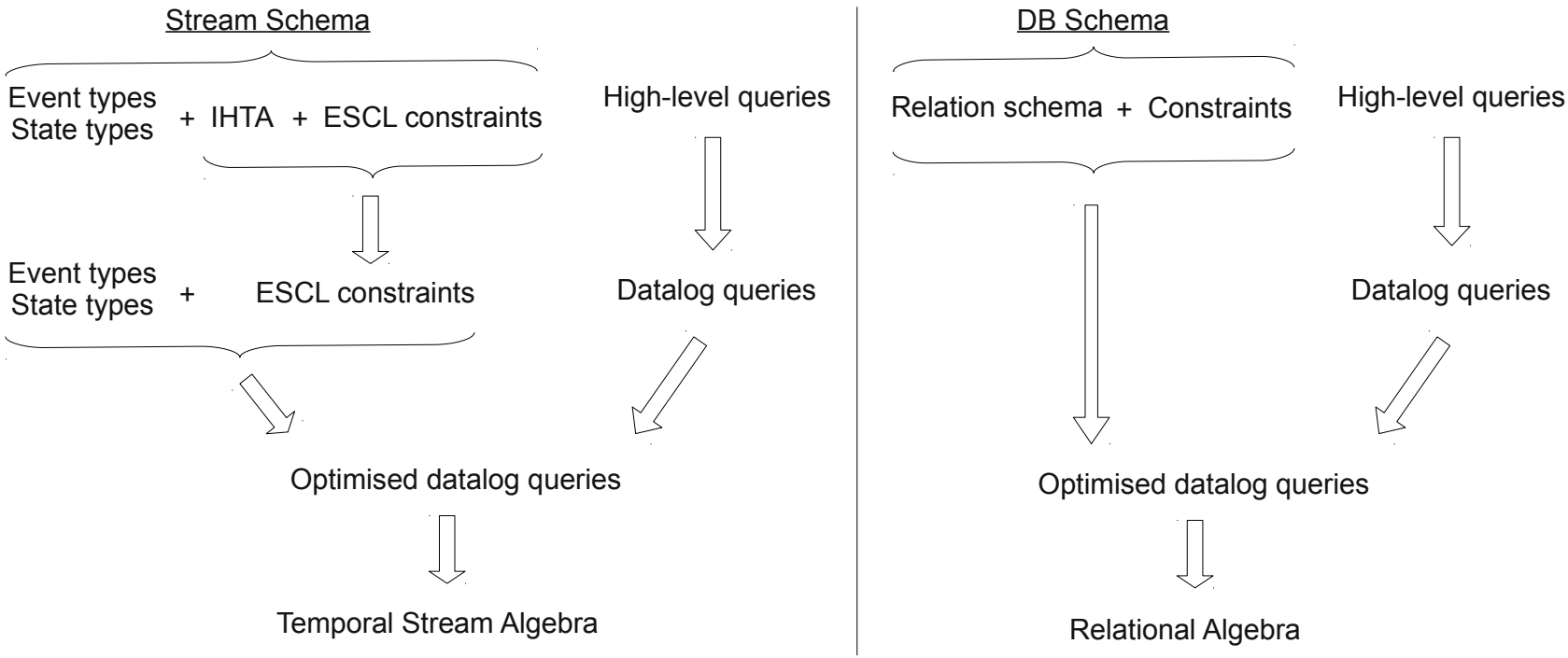
1. Arbitrary number of concurrent processes
2. Events and states carry data
3. Temporal constraints on the *beginning* and the end of the occurrence time of any event accepted by the instance
4. Goal:
   HTA: synchronisation, deadlock checking, reachability...
   IHTA: stream schema

IHTA vs. HTA

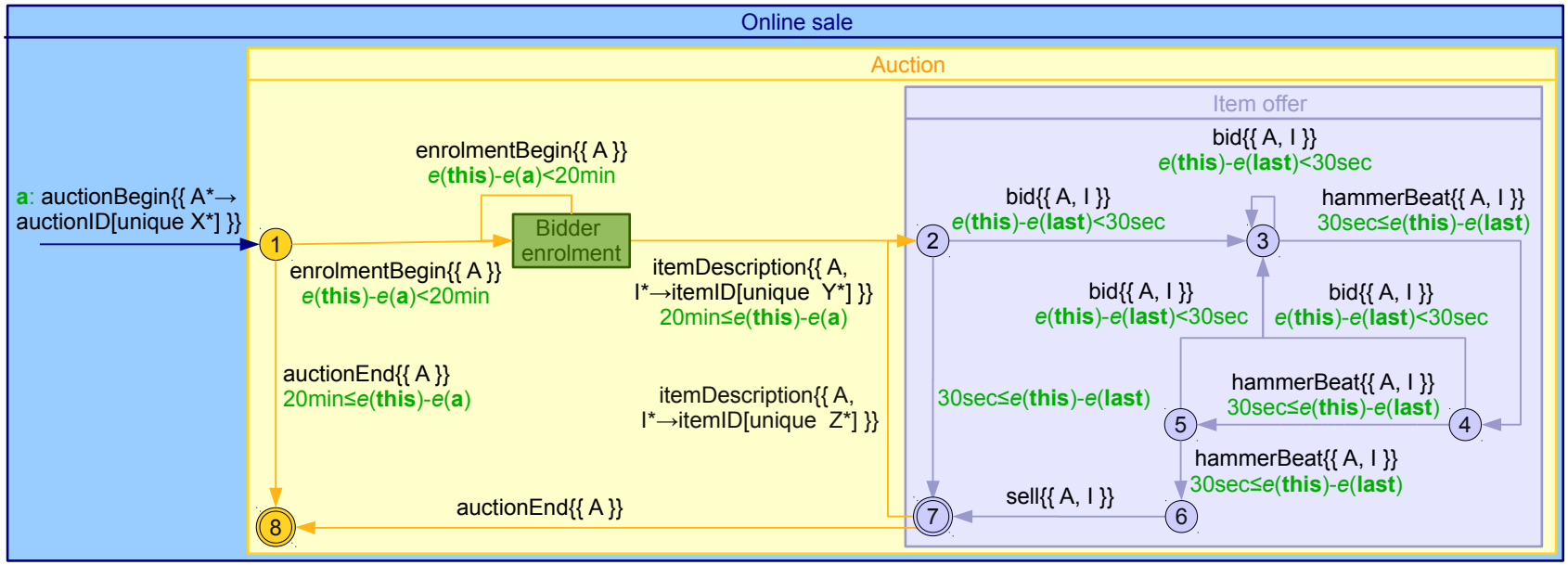
Instantiating Hierarchical Timed Automata $\approx$ HTA –

1. Synchronisation channels
2. Prioritisation
3. History states
4. ...

Overview


Stream Schema
Event types
State types
+ IHTA + ESCL constraints
High-level queries
Event types
State types
+ ESCL constraints
Datalog queries
Optimised datalog queries
Temporal Stream Algebra
DB Schema
Relation schema + Constraints
High-level queries
Datalog queries
Optimised datalog queries
Relational Algebra

Constraints Expressed by IHTA



1. Cardinality — all automata
2. Functional depenencies
   - Temporal relations — all timed automata
   - Causal relations — all automata
   - Relations between event data — IHTA

Outline

Requirements to a Constraint Language for CEP

Kinds of constraints in CEP applications:

- Cardinality
- Functional dependencies:
  - Causality
  - Temporal relations
  - Spatial relations
  - Data relations

Simple but expressive constraints for CEP by means of:

1. (Incomplete and unordered) Event and state queries
2. Identification of matched events and states
3. Quantification

Related Work

Constraints languages:

- DDL [12],
- DTD [22, 21], [15, 16], [7], [14], [13], [18],
- Regular expressions [10], [9], [11], [23],
- Denial rules [8],
- Language independent formulas [22, 21], [7], [11], [5], [3], [24], [2, 19].

Particular kind(s) of constraints:

- Cardinality constraints [22, 21], [7], [12], [3], [24], [2, 19],
- Causal dependencies [22, 21], [8],
- Conditions on event data [10], [9], [3], [23],
- Temporal dependencies [22, 21], [7], [10], [9], [11], [12], [5], [3], [23], [8].

Event Stream Constraint Language

ESCL $\approx$ Denial rules
+ Identifiers
+ Temporal and other relations
+ Quantifiers
+ Modules
+ Local definitions

Event Stream Constraint Language

There is exactly one item description for each item presented in an auction.

```
WHILE state: auction{{ auctionID[var A] }}
LET
    IF    event i: itemDescription{{ auctionID[var A], itemID[var I] }}
    THEN ∃=1 i
    END
END
```

Bidder with ID 007 may not participate in any auction because of his bad behaviour.

```
WHILE state: auction{{ auctionID[var A] }}
LET
    IF event: bid{{ auctionID[var A], itemID[var I], bidderID[var B] }}
    THEN var B ≠ 007
    END
END
```

Local Definitions

Price for an item increases during an auction.

```
WHILE
      state: itemOffer{{ itemID[var I] }}
LET
    LET
        DETECT
              event: e{ itemID[var I], value[var V1] }
        ON or {
              event: bid{{ itemID[var I], value[var V1] }},
              event: hammerBeat{{ itemID[var I], value [var V1] }}
            }
        END
    IN
        IF    and { event: itemDescription{{ itemID[var I], value[var V2] }},
                    event: e{{ itemID[var I], value[var V1] }} }
        THEN var V2 ≤ var V1
        END
        IF    and { event i: e{{ itemID[var I], value[var V1] }},
                    event b: bid{{ itemID[var I], value [var V2] }},
                    i before b }
        THEN var V1 < var V2
        END
        IF    and { event i: e{{ itemID[var I], value[var V1] }},
                    event h: hammerBeat{{ itemID[var I], value [var V2] }},
                    i before h }
        THEN var V1 ≤ var V2
        END
        IF    and { event: e{{ itemID[var I], value[var V1] }},
                    event: sell{{ itemID[var I], value [var V2] }} }
        THEN var V1 ≤ var V2
        END
    END
END
```

Translation of Constraints into a Processable Form

Normalisation:

- Incomplete and unordered queries
- Disjunctions in the body
- Local definitions
- Modules

```
∃=1 i ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s

var B ≠ 007 ←
    event b: bid[ auctionID[var A], itemID[var I], bidderID[var B] ],
    state s: auction[ auctionID[var A] ],
    b during s
```

Outline

- Generic

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s
```

Constraint:

```
∃=1 i ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s
```

Constraint:

```
∃=1 i ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s,
    { ∃=1 i }
```

Constraint:

```
∃=1 i ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s,
    { ∃=1 i },
    { 007 ≠ 007 }
```

Constraint:

```
var B ≠ 007 ←
    event b: bid[ auctionID[var A], itemID[var I], bidderID[var B] ],
    state s: auction[ auctionID[var A] ],
    b during s
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s,
    { ∃=1 i },
    { 007 ≠ 007 }
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[0], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    { ∃=1 i }
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[0], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    { ∃=1 i }
```

Query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[0], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    { ∃=1 i }
```

Optimised query:

```
bidNumber[
  auctionID[var A], itemID[var I], number[0], bidderID[007]
] ←
   event i: itemDescription[ auctionID[var A], itemID[var I] ]
```

Original query:

```
bidNumber[
  auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
   event i: itemDescription[ auctionID[var A], itemID[var I] ],
   var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
   state s: auction[ auctionID[var A] ],
   i during s,
   b during s
```

Semantic Rewriting of CEP Queries

Let S_Q be set of queries and
S_C be set of constraints

FOR EACH $H_Q \leftarrow B_Q \in S_Q$
  FOR EACH $H_C \leftarrow B_C \in S_C$
    IF B_C subsums B_Q with σ
    THEN rewrite B_Q to $B_Q, H_C\sigma$
    END
  END
END

Return S_Q

Semantic Rewriting of CEP Queries

Let S_Q be set of queries and
S_C be set of constraints

FOR EACH $H_Q \leftarrow B_Q \in S_Q$
  FOR EACH $H_C \leftarrow B_C \in S_C$
    IF B_C subsums B_Q with σ and
      a SO heuristic is applicable to $H_Q \leftarrow B_Q$ and $H_C\sigma$
    THEN rewrite B_Q to $B_Q, H_C\sigma$
      apply SO heuristics to B_Q
    END
  END
END

Return S_Q

SO Heuristics from DB

1. Result by contradiction
2. Result by transformation
3. Join elimination
4. Join introduction
5. Predicate elimination
6. Predicate introduction

Original query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[count(all var X)], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ],
    var X → event b: bid[ auctionID[var A], itemID[var I], bidderID[007] ],
    state s: auction[ auctionID[var A] ],
    i during s,
    b during s
```

Optimised query:

```
bidNumber[
   auctionID[var A], itemID[var I], number[0], bidderID[007]
] ←
    event i: itemDescription[ auctionID[var A], itemID[var I] ]
```

Subsumption of CEP Queries

Dimensions:

1. Event and state literals
2. Temporal and other relations
3. Complex events / query propagation
4. Cardinality
5. Constraints

Let Q be a query,
let Q' be a semantically optimised query Q using a constraint set C.

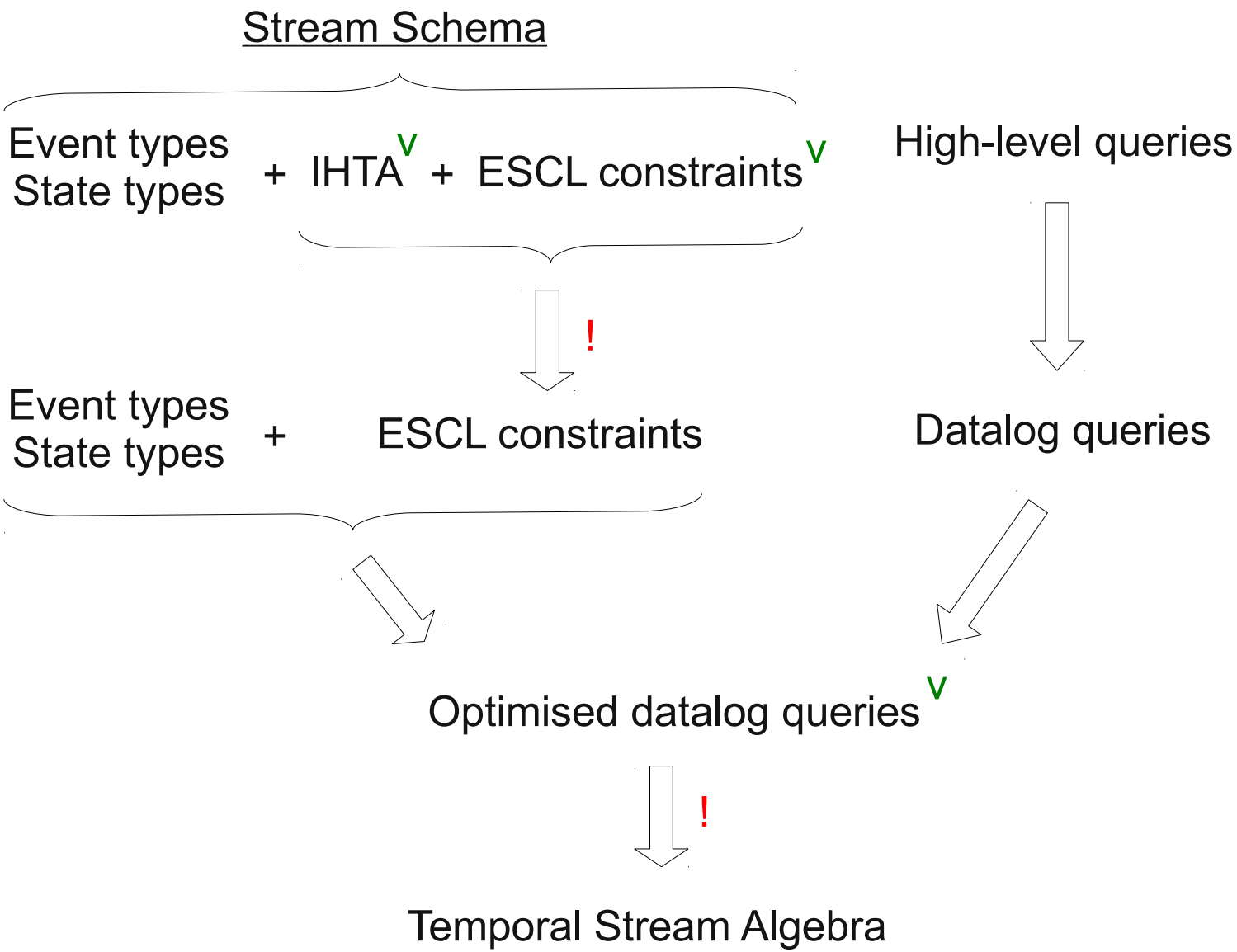
Q and Q' must be *semantically equivalent*, i.e.
return the same results with restect to all streams satisfying C.

The SO algorithm is proven to

- terminate,
- be correct,
- be in $O(Q \cdot C \cdot S \cdot H)$ where
  Q is the number of queries,
  C is the number of constraints,
  S is the costs for subsumption of CEP queries,
  H is the costs for the application of SO heuristics

Outline

Overview


Stream Schema
Event types
State types
+ IHTA
v
+ ESCL constraints
v
High-level queries
!
Event types
State types
+ ESCL constraints
Datalog queries
Optimised datalog queries
v
!
Temporal Stream Algebra

Conclusion

Complete work:

1. IHTA: Instantiating Hierarchical Timed Automata (Sandro Giessl)
2. ESCL: Event Stream Constraint Language
3. Algorithm for semantic rewriting of CEP queries
4. Auction use case
5. Related work on SO of CEP

Current work:

1. Algorithm for constraint-based subsumption of CEP queries
2. Algorithms propagating constraints from base events or states to derived events or states (ThanhSon Dang)

Future work:

1. Algorithm deriving constraints from IHTA
2. Visual editor for IHTA, ESCL constraints and CEP queries
3. Sensor network use case
4. Dynamic semantic optimisation of CEP queries
5. Cost models and evaluation

Bibliography I

[1] R. Alur and D. L. Dill.
Automata for modeling real-time systems.
In *Proc. Int. Conf. on Automata, Languages and Programming*, pages 322–335. Springer, 1990.

[2] R. Avnur and J. M. Hellerstein.
Eddies: Continuously adaptive query processing.
In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 261–272. ACM, 2000.

[3] S. Babu, U. Srivastava, and J. Widom.
Exploiting k-constraints to reduce memory overhead in continuous queries over data streams.
ACM Transactions on Database Systems, 29(3):545–580, 2004.

[4] J. Bengtsson and W. Yi.
Timed automata: Semantics, algorithms and tools.
In *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2004.

[5] F. Bry and M. Eckert.
Temporal order optimizations of incremental joins for composite event detection.
In *Proc. Int. Conf. on Distributed Event-Based Systems*. ACM, 2007.

[6] A. David and M. D. Mölle.
From HUPPAAL to UPPAAL - a translation from Hierarchical Timed Automata to Flat Timed Automata, 2001.

[7] Y. Diao and M. Franklin.
Query processing for high-volume XML message brokering.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 261–272. VLDB Endowment, 2003.

Bibliography II

[8] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan.
Runtime semantic query optimization for event stream processing.
In *Proc. Int. Conf. on Data Engineering*, pages 676–685. IEEE Computer Society, 2008.

[9] L. Ding, N. Mehta, E. A. Rundensteiner, and G. T. Heineman.
Joining punctuated streams.
In *Proc. Int. Conf. on Extending Database Technology*, volume 2992 of *Lecture Notes in Computer Science*, pages 587–604. Springer, 2004.

[10] L. Ding, E. A. Rundensteiner, and G. T. Heineman.
MJoin: A metadata-aware stream join operator.
In *Proc. Int. Workshop on Distributed Event-Based Systems*, pages 1–8. ACM, 2003.

[11] P. M. Fischer, K. S. Esmaili, and R. J. Miller.
Stream schema: Providing and exploiting static metadata for data stream processing.
In *Proc. Int. Conf. on Extending Database Technology*, volume 426 of *ACM Int. Conf. Proc.*, pages 207–218. ACM, 2010.

[12] L. Golab, T. Johnson, N. Koudas, D. Srivastava, and D. Toman.
Optimizing away joins on data streams.
In *Proc. Int. Workshop on Scalable Stream Processing System*, pages 48–57. ACM, 2008.

[13] T. J. Green, G. Miklau, M. Onizuka, and D. Suciu.
Processing XML streams with deterministic automata.
In *Proc. Int. Conf. on Database Theory*, pages 173–189. Springer, 2002.

[14] A. K. Gupta and D. Suciu.
Stream processing of XPath queries with predicates.
In *Proc. Int. Conf. on Management of Data*, pages 419–430. ACM Press, 2003.

Bibliography III

[15] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier.
FluXQuery: An optimizing XQuery processor for streaming XML data.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 1309–1312. VLDB Endowment, 2004.

[16] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier.
Schema-based scheduling of event processors and buffer minimization for queries on structured data streams.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 228–239. VLDB Endowment, 2004.

[17] S. Lasota and I. Walukiewicz.
Alternating Timed Automata.
volume 9. ACM, 2008.

[18] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou.
A transducer-based XML query processor.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 227–238. VLDB Endowment, 2002.

[19] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman.
Continuously adaptive continuous queries over streams.
In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 49–60. ACM, 2002.

[20] U. Schöning.
Theoretische informatik kurz gefasst, 1992.

[21] H. Su, E. A. Rundensteiner, and M. Mani.
Semantic query optimization in an automata-algebra combined XQuery engine over XML streams.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 1293–1296. VLDB Endowment, 2004.

[22] H. Su, E. A. Rundensteiner, and M. Mani.
Semantic query optimization for XQuery over XML streams.
In *Proc. Int. Conf. on Very Large Data Bases*, pages 277–288. VLDB Endowment, 2005.

Bibliography IV

[23] P. A. Tucker, D. Maier, T. Sheard, and P. Stephens.
Using production schemas to characterize strategies for querying over data streams.
IEEE Transactions on Knowledge and Data Engineering, 19(9):1227–1240, 2007.

[24] S. D. Viglas and J. F. Naughton.
Rate-based query optimization for streaming information sources.
In *Proc. Int. Conf. on Management of Data*, pages 37–48. ACM, 2002.