Type Based XML Projection

VLDB 2006, Seoul, Korea

Véronique Benzaken[⋆] Giuseppe Castagna[♢] Dario Colazzo[⋆]
Kim Nguyễn[⋆]

[⋆] : Équipe Bases de Données, LRI, Université Paris-Sud 11, Orsay, France

[♢] : Équipe Langage, DI, École Normale Supérieure, Paris, France

Outline

Outline

Main memory query (XQuery) processing

Main memory query (XQuery) processing

Pros :

- Lightweight system

Main memory query (XQuery) processing

Pros :

- Lightweight system
- Easy to program and integrate into an existing architecture

Main memory query (XQuery) processing

Pros :

- Lightweight system
- Easy to program and integrate into an existing architecture

Cons :

- Unlike DBMS solution, whole document in memory (DOM $\rightarrow$ lots of metadata)

Main memory query (XQuery) processing

Pros :

- Lightweight system
- Easy to program and integrate into an existing architecture

Cons :

- Unlike DBMS solution, whole document in memory (DOM $\rightarrow$ lots of metadata)

We can use **pruning** to reduce the amount of data needed in main memory !

We can use **pruning** to reduce the amount of data needed in main memory !

Exemple :

```
/descendant-or-self::title/child::text()
```

We can use **pruning** to reduce the amount of data needed in main memory !

Exemple :

```
/descendant-or-self::title/child::text()
```

```
<bib>
 <book year="1994">
  <title>TCP/IP Illustrated</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
 </book>
 <book year="1992">
  <title>Advanced Programming in the Unix environment</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
 </book>
 ...
</bib>
```

Projecting XML Documents, Amélie Marian and Jérome Siméon, VLDB 2003.

Queries + document based optimizations, it has some drawbacks :

- Does not take into account backward axis.
- Performances degrade in the presence of `//`

Indeed, with a query like :

```
/descendant-or-self::title/child::text()
```

one has to iterate through the whole document.

Solution : Queries + **type** based optimisations

By using a DTD and a given set of queries, we can statically infer a *projector* for the set of queries and use it to prune the document.
Main advantages :

- Pruning is efficient
- Take into account // and backward axis
- Pruning is sound : executing the query on the projected document gives the same result as on the original.
- Pruning is precise and even exact for a large class of documents

Outline

Definition (DTD)

A DTD is a pair (X, E) where X is a distinguished name (the root) and E is a set of productions of the form

$$\{X_1 \to R_1,\ \ldots, X_n \to R_n\}$$

where each R_i is of the form :

$$a_i[\textit{Regexp}] \text{ or String}$$

where a_i is a unique `tag` name and Regexp a regular expression of X_i. Names(E) is the set of names occuring in E.

Definition (Projector)

For a given DTD (X, E) a type projector for (X, E) is a set of names $\mathcal{P}$ such that :

1. $\mathcal{P} \subseteq$ *Names(E)*
2. $\forall X_i \in \mathcal{P}$ *there is at least one derivation from* X *to* X_i *in* E

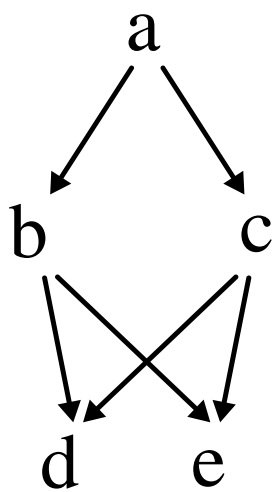
Definition (Pruning)

The pruning (or projection) of a document D valid w.r.t a DTD (X, E) with the projector $\mathcal{P}$ is a document D' where every node not generated by a name in $\mathcal{P}$ is erased (replaced by the empty sequence).

Definition (Pruning)

The pruning (or projection) of a document D valid w.r.t a DTD (X, E) with the projector $\mathcal{P}$ is a document D' where every node not generated by a name in $\mathcal{P}$ is erased (replaced by the empty sequence).

```
<!ELEMENT a   (b,c)>
<!ELEMENT b   (d,e)>
<!ELEMENT c   (d,e)>
<!ELEMENT d   (#PCDATA)>
<!ELEMENT e   (#PCDATA)>
```



/descendant-or-self::c

/descendant-or-self::c

$$X_a \to \mathtt{a}[X_b, X_c]$$
$$X_b \to \mathtt{b}[X_d, X_e] \quad X_c \to \mathtt{c}[X_d, X_e]$$
$$X_d \to \mathtt{String} \quad X_e \to \mathtt{String}$$

```
<a>
 <b>
  <d>LotsOfData</d>
  <e>SuperLongString<e/>
 </b>
 <c>
  <d>bar</d>
  <e>foo<e/>
 </c>
</a>
```

/descendant-or-self::c

$$X_a \to \mathtt{a}[X_b, X_c]$$
$$X_b \to \mathtt{b}[X_d, X_e] \quad X_c \to \mathtt{c}[X_d, X_e]$$
$$X_d \to \mathtt{String} \quad X_e \to \mathtt{String}$$

$$\mathcal{P} = \{X_a, X_c, X_d, X_e\}$$

```
<a>
 <b>
  <d>LotsOfData</d>
  <e>SuperLongString<e/>
 </b>
 <c>
  <d>bar</d>
  <e>foo<e/>
 </c>
</a>
```

```
<a>
 <b>
  <d>LotsOfData</d>
  <e>Superlongstring<e/>
 </b>
 <c>
  <d>foo</d>
  <e>bar<e/>
 </c>
</a>
```

/descendant-or-self::c

$X_a \to \mathtt{a}[X_b, X_c]$
$X_b \to \mathtt{b}[X_d, X_e]$ $\quad X_c \to \mathtt{c}[X_d, X_e]$
$X_d \to \mathtt{String}$ $\quad X_e \to \mathtt{String}$

$\mathcal{P} = \{X_a, X_c, X_d, X_e\}$

```
<a>
 <b>
  <d>LotsOfData</d>
  <e>SuperLongString<e/>
 </b>
 <c>
  <d>bar</d>
  <e>foo<e/>
 </c>
</a>
```

```
<a>
 <b>
  <d>LotsOfData</d>
  <e>Superlongstring<e/>
 </b>
 <c>
  <d>foo</d>
  <e>bar<e/>
 </c>
</a>
```

Definition (XPath (1.0))

$$
\begin{array}{lcl}
Q & ::= & \mathit{Axis} :: \mathit{Test} \\
 & | & \mathit{Axis} :: \mathit{Test}[\mathit{Expr}] \\
 & | & Q/Q
\end{array}
\qquad
\begin{array}{lcl}
\mathit{Expr} & := & Q \\
 & | & \mathit{Expr}\ op\ \mathit{Expr} \\
 & | & f(\mathit{Expr}, \ldots, \mathit{Expr}) \\
 & | & Base
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{Axis} & ::= & \texttt{self} \mid \texttt{child} \mid \texttt{descendant} \mid \texttt{parent} \mid \texttt{ancestor} \mid \ldots \\
\mathit{Test} & ::= & tag \mid \texttt{node()} \mid \texttt{text()}
\end{array}
$$

Definition (Simple path)

$$
\begin{array}{lcl}
Q & ::= & \mathit{Axis} :: \mathit{Test} \\
 & | & \mathit{Axis} :: \mathit{Test}[\mathit{Cond}] \\
 & | & Q/Q
\end{array}
\qquad
\begin{array}{lcl}
\mathit{Cond} & := & \mathit{SPath} \\
 & | & \mathit{Cond}\ \texttt{or}\ \mathit{Cond}
\end{array}
$$

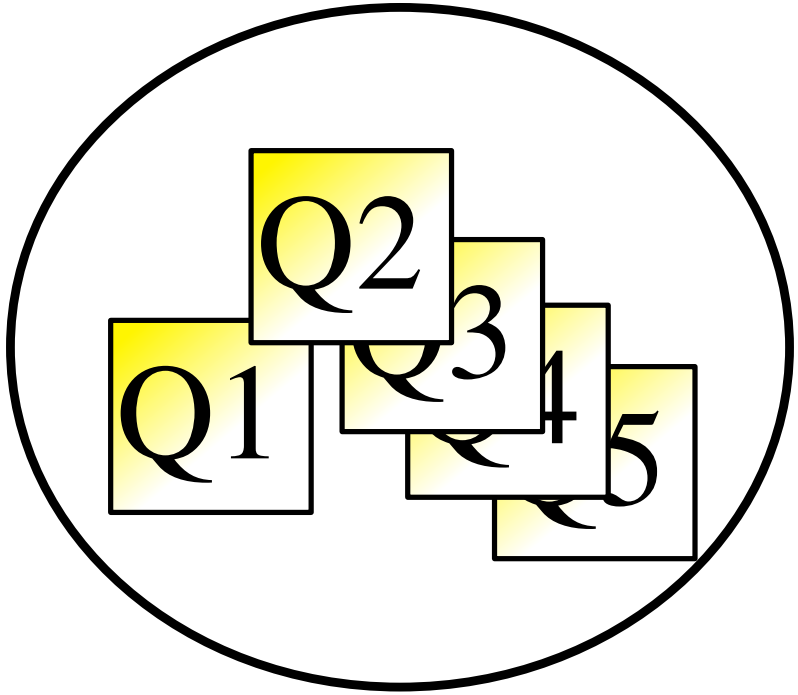
$$
\begin{array}{lcl}
\mathit{SPath} & ::= & \mathit{Axis} :: \mathit{Test} \\
 & | & \mathit{Axis} :: \mathit{Test}/\mathit{SPath}
\end{array}
$$

Outline

Static inference :

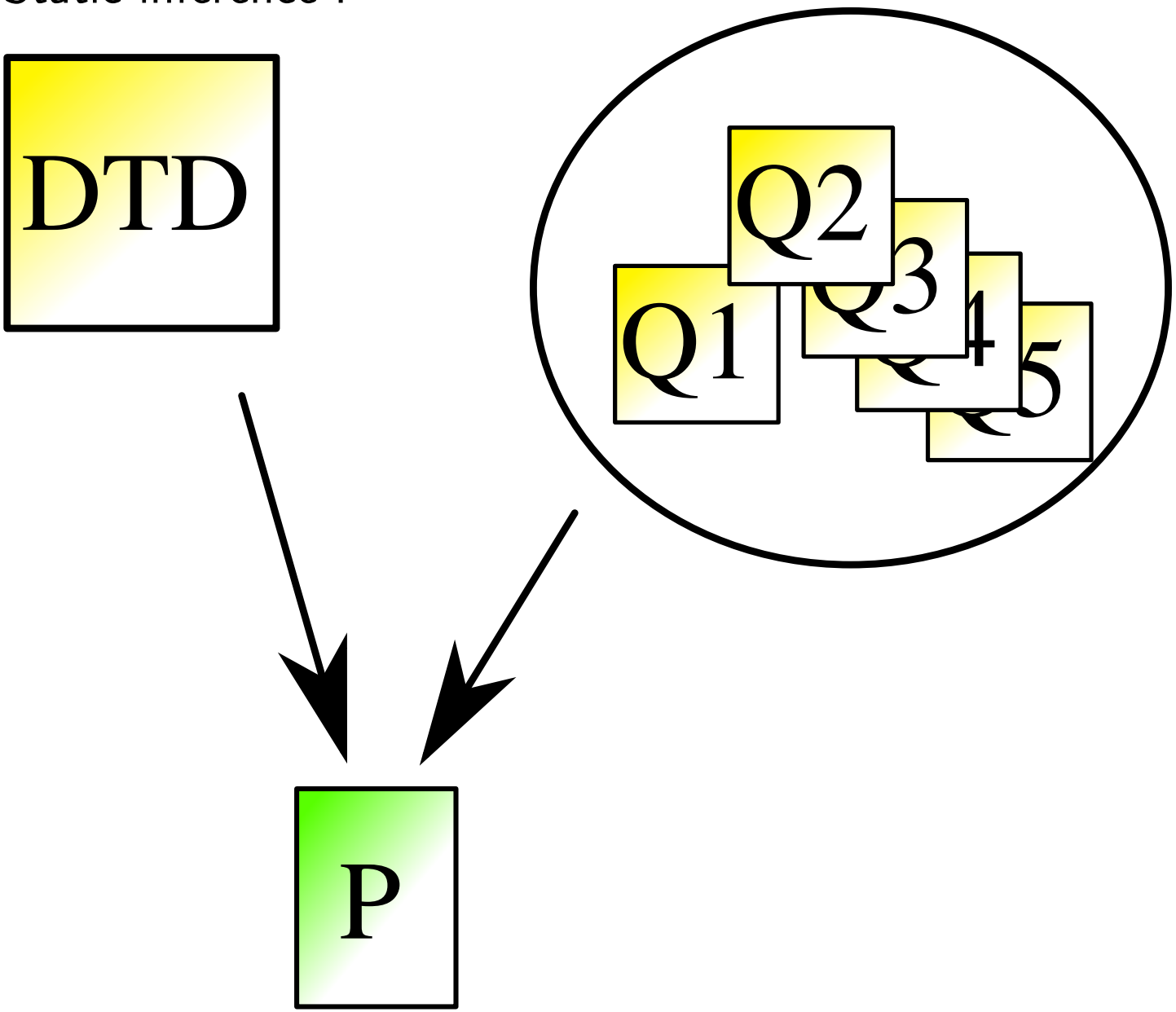
Overall process

Static inference :



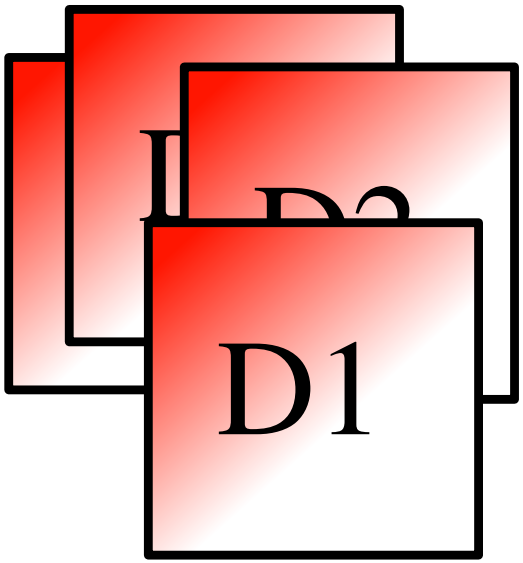
Overall process

Static inference :

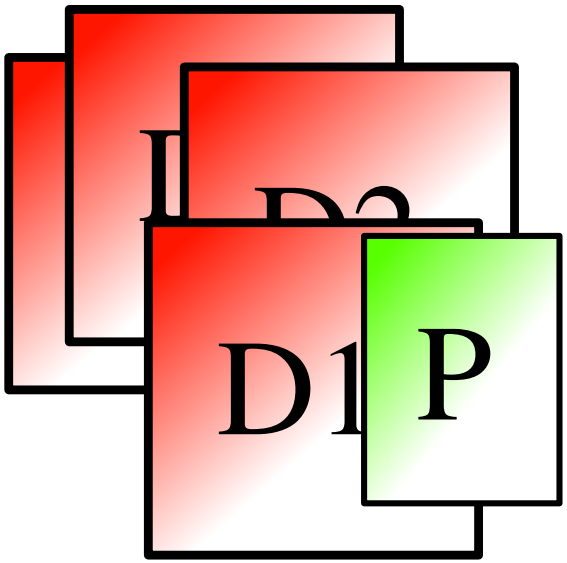


Overall process

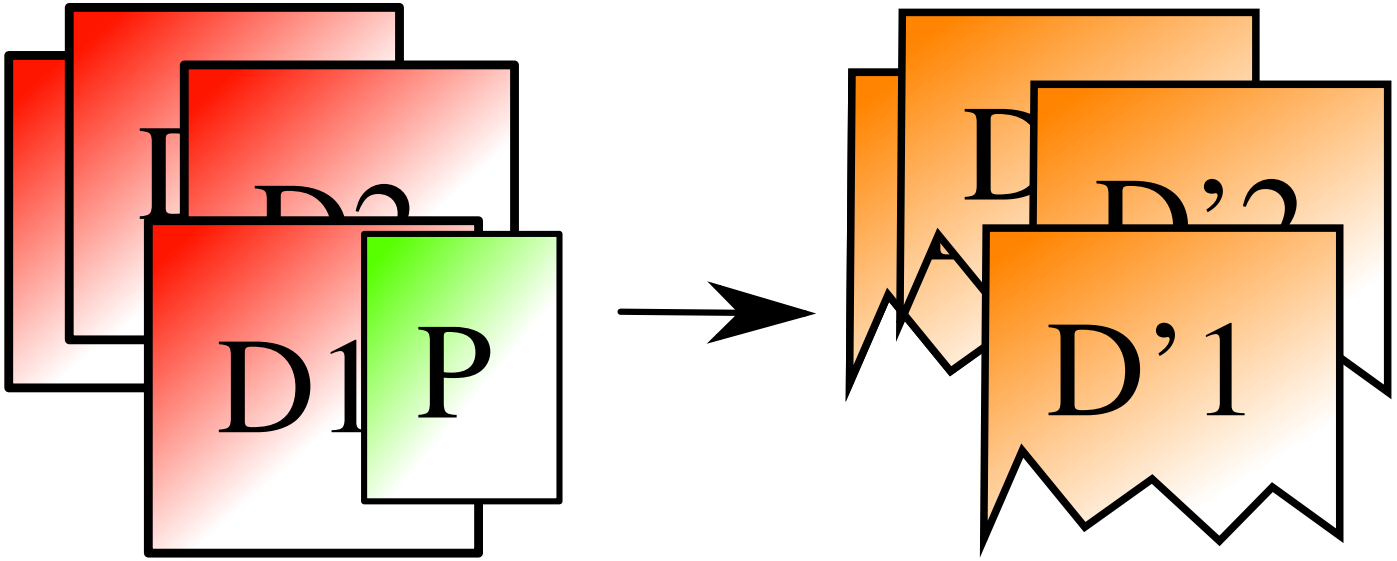
Runtime :



Runtime :

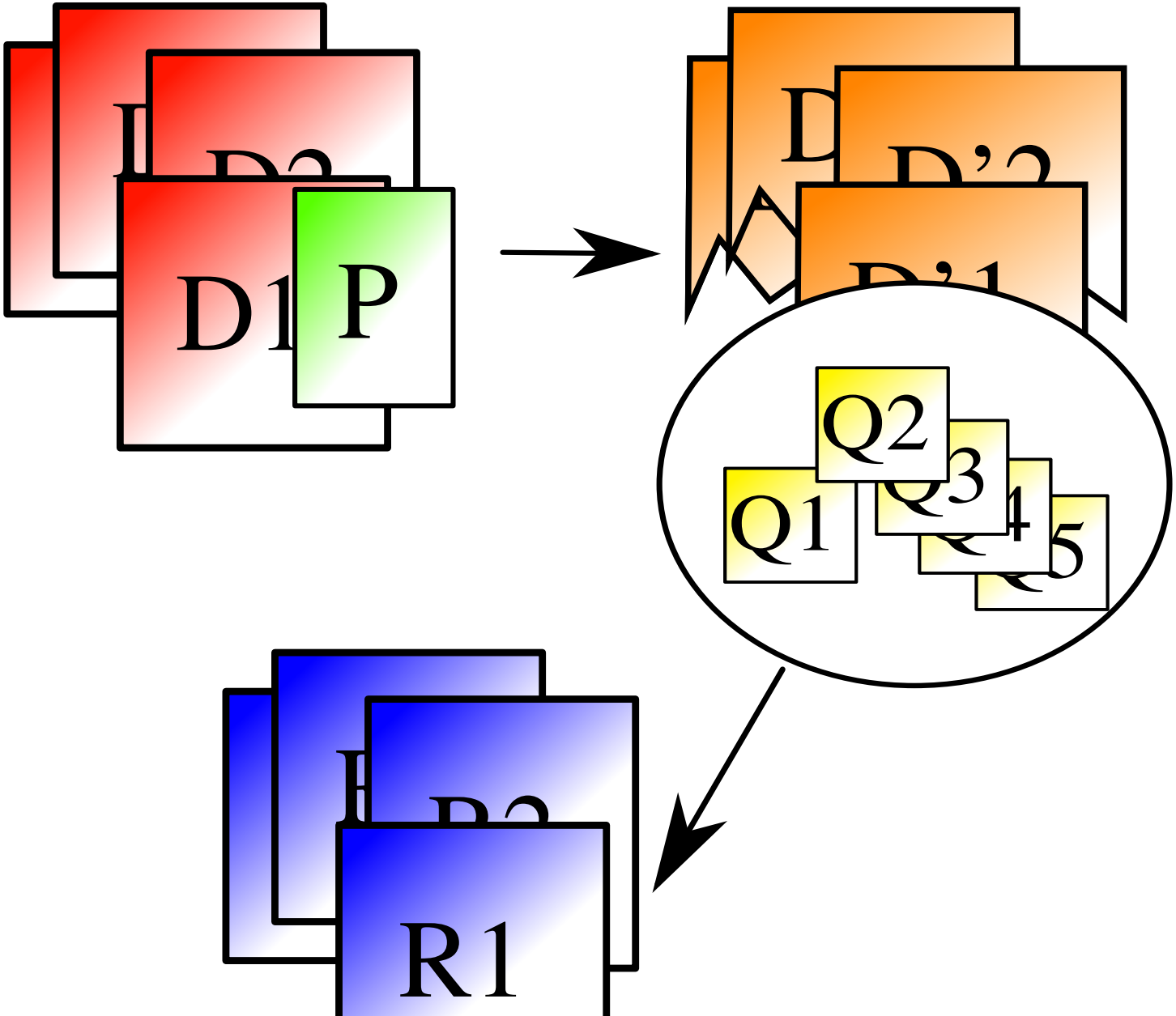


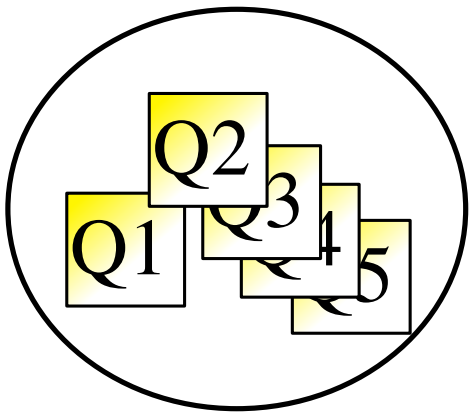
Runtime :



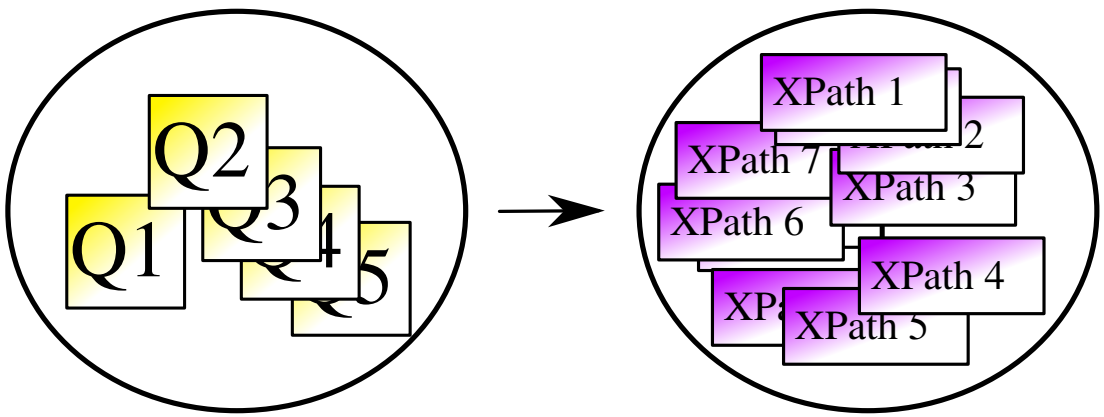
Overall process

Runtime :

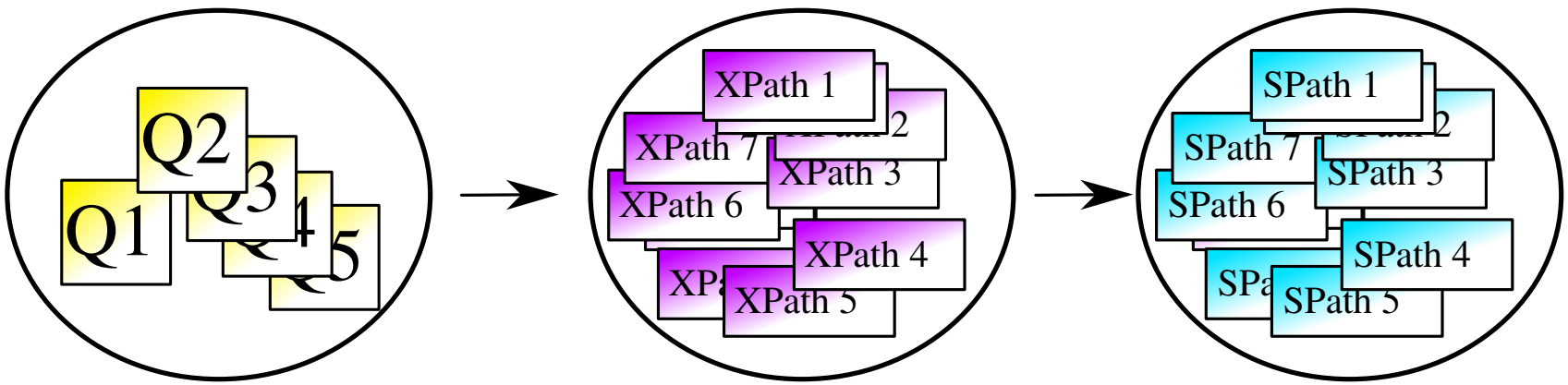



Q2
Q1

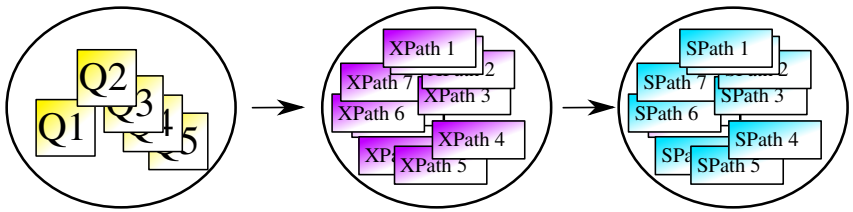
Static inference



Static inference

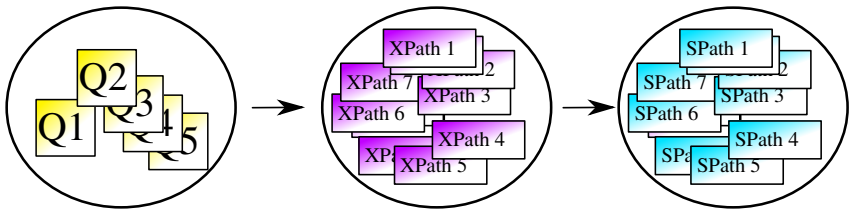


Static inference



SPath 1

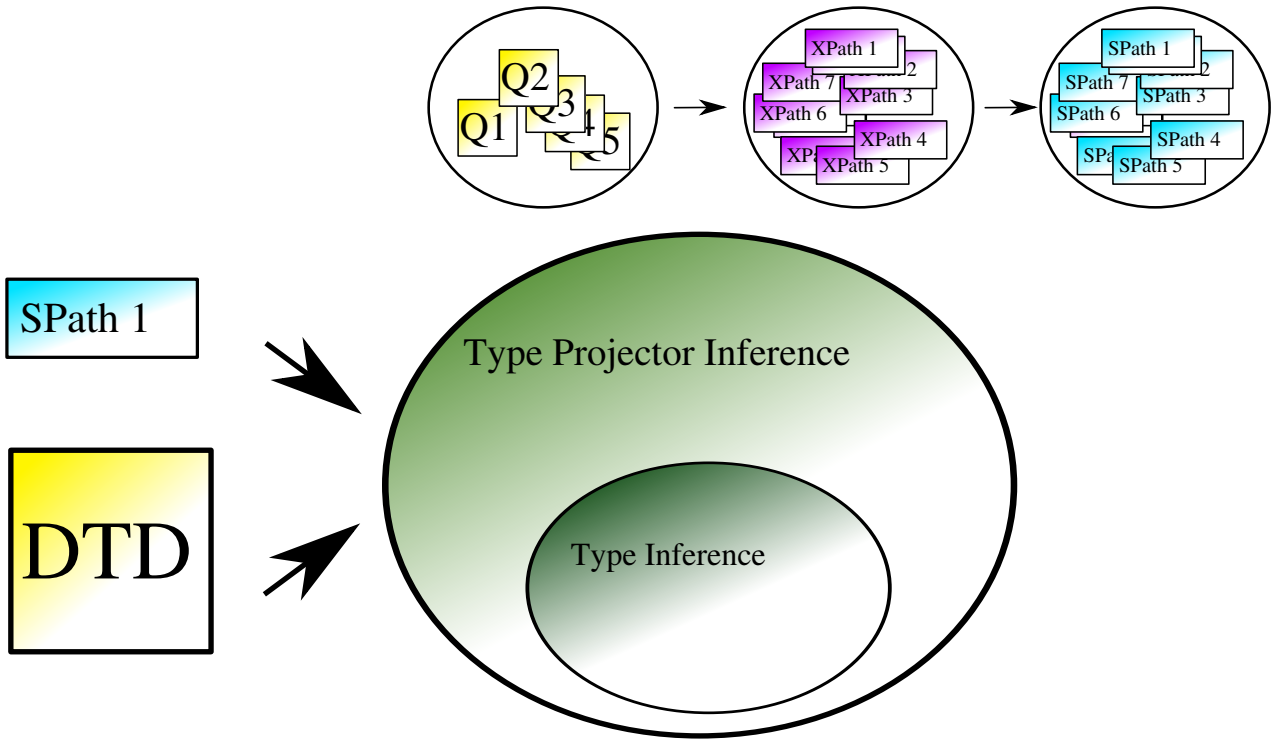
Static inference



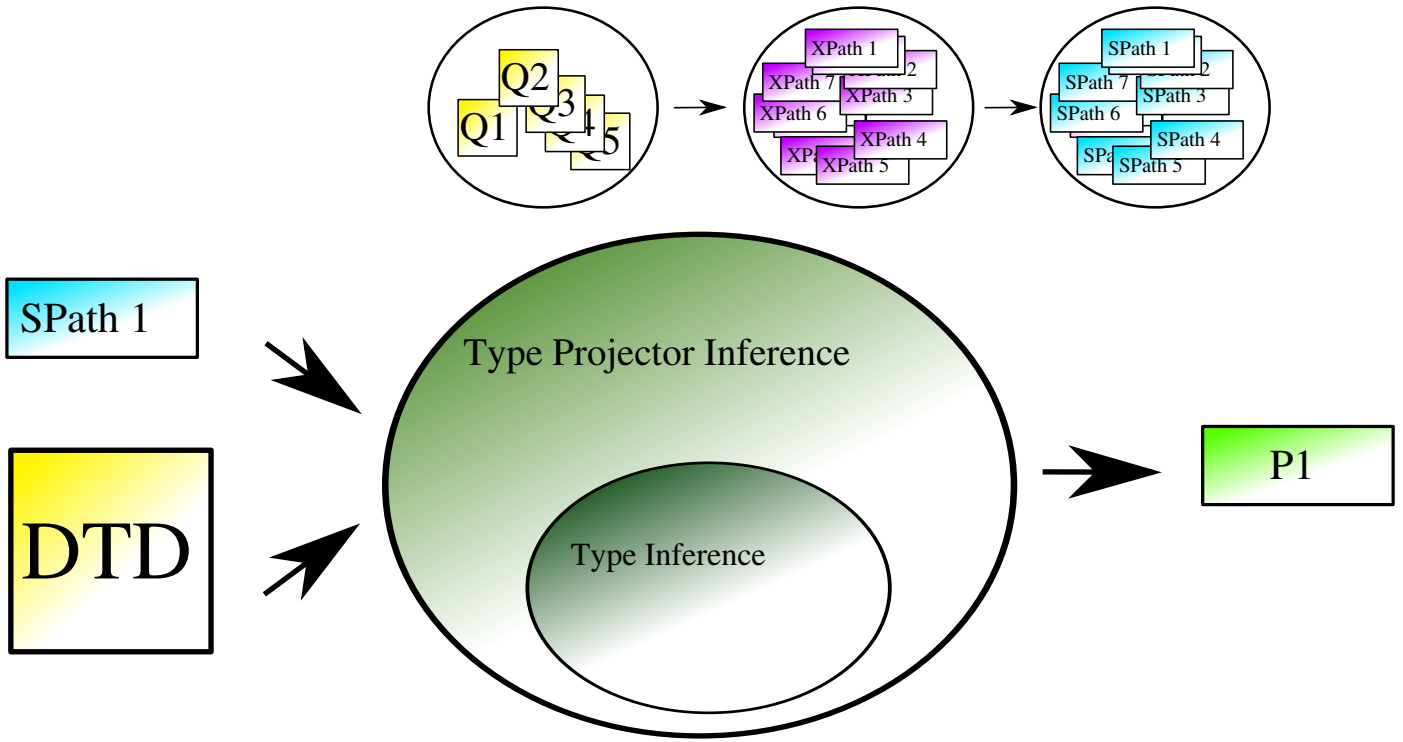
SPath 1

DTD

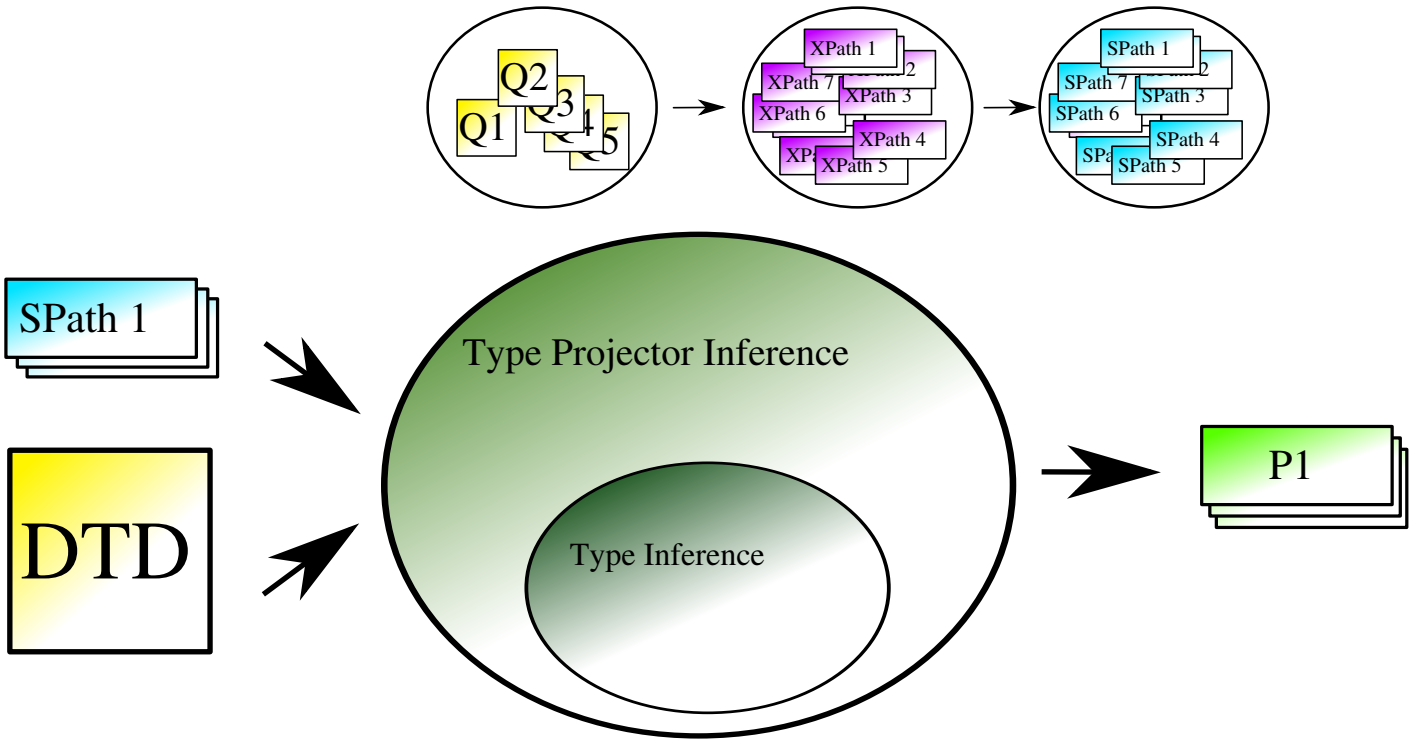
Static inference



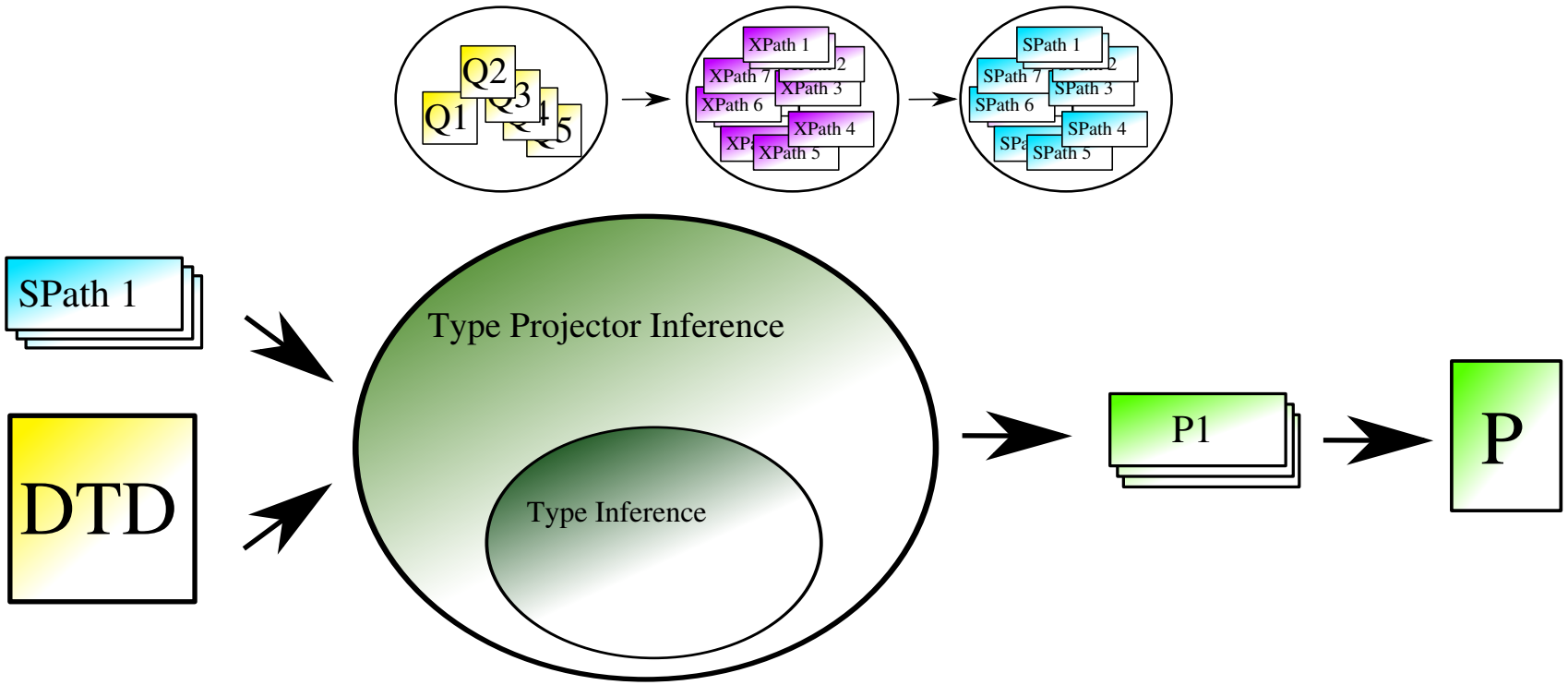
Static inference



Static inference



Static inference



$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.

$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

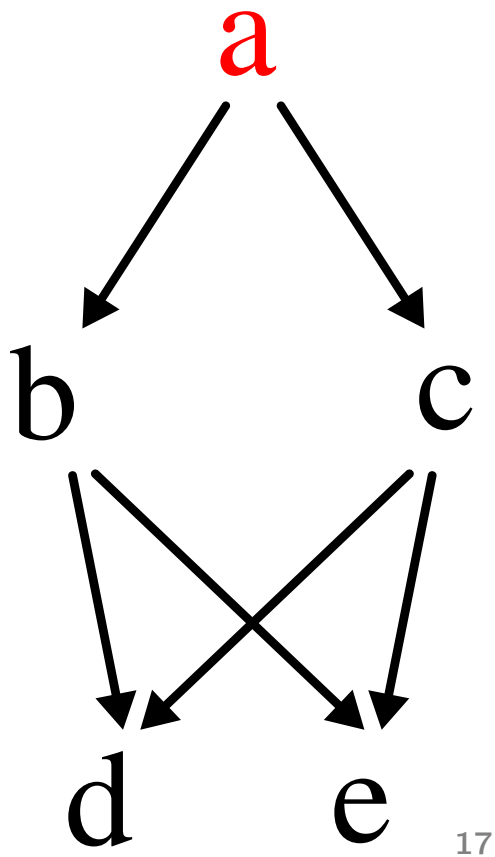
```
/self::a/child::b/child::d
```

$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

```
/self::a/child::b/child::d
```

1. typeinf(DTD,$\{X_a\}$,self::a)$= \{X_a\}$

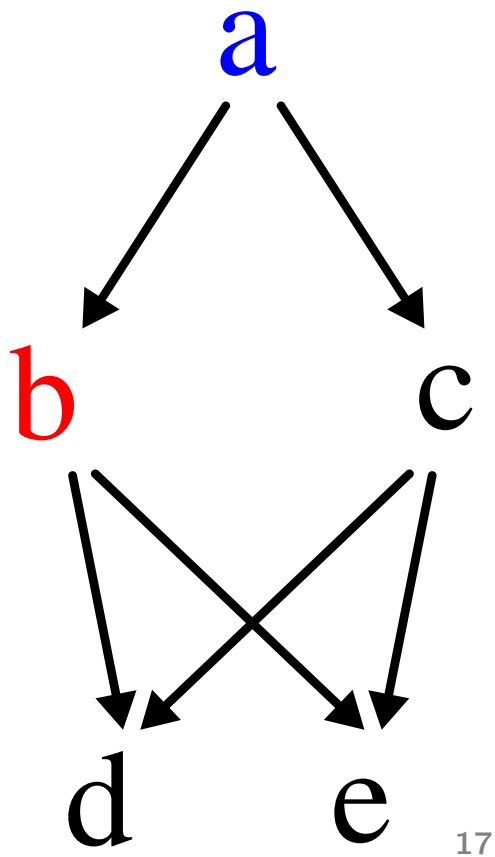


$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

`/self::a/child::b/child::d`

1. $\texttt{typeinf(DTD,}\{X_a\}\texttt{,self::a)} = \{X_a\}$
2. $\texttt{typeinf(DTD,}\{X_a\}\texttt{,child::b)} = \{X_b\}$

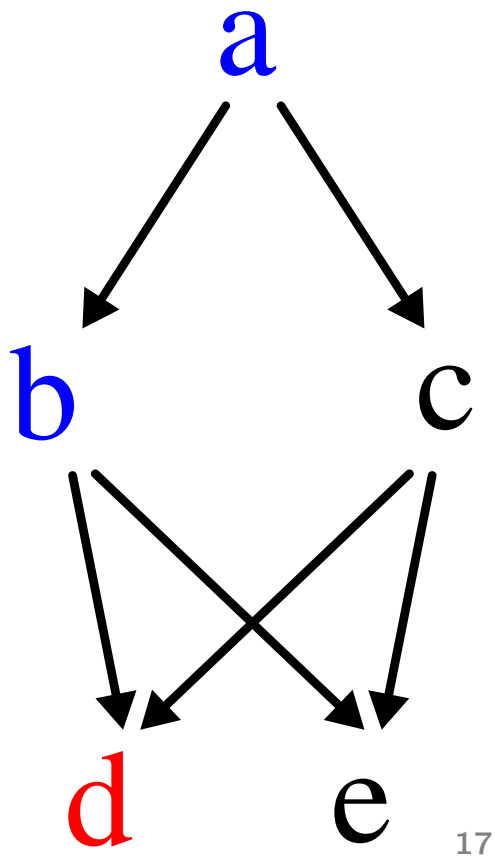


$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

`/self::a/child::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,self::a)`$= \{X_a\}$
2. `typeinf(DTD,`$\{X_a\}$`,child::b)`$= \{X_b\}$
3. `typeinf(DTD,`$\{X_b\}$`,child::d)`$= \{X_d\}$



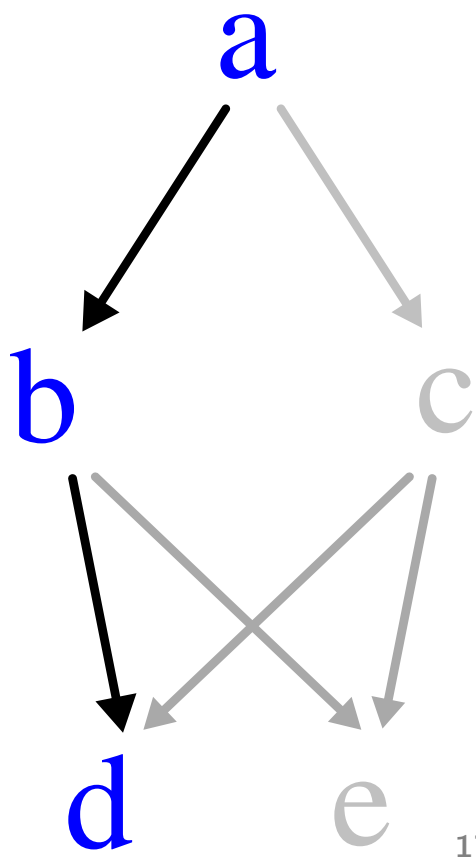
$$\texttt{typeinf(DTD,}\{X\}\texttt{,}path\texttt{)} = T$$

Type T is the set of names of types of nodes in the result.
$T = \varnothing \rightarrow$ empty query.

`/self::a/child::b/child::d`

1. $\texttt{typeinf(DTD,}\{X_a\}\texttt{,self::a)} = \{X_a\}$
2. $\texttt{typeinf(DTD,}\{X_a\}\texttt{,child::b)} = \{X_b\}$
3. $\texttt{typeinf(DTD,}\{X_b\}\texttt{,child::d)} = \{X_d\}$

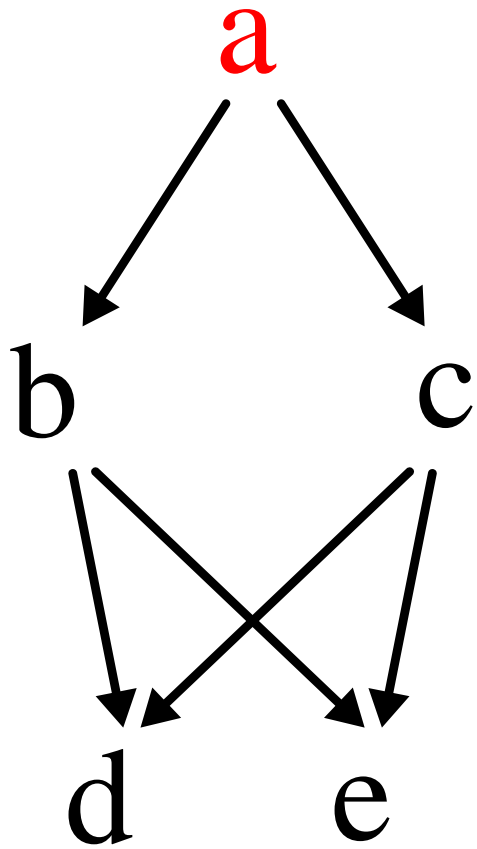
$\mathcal{P} = \{X_a, X_b, X_d\}$



```
//*/self::b/child::d
```

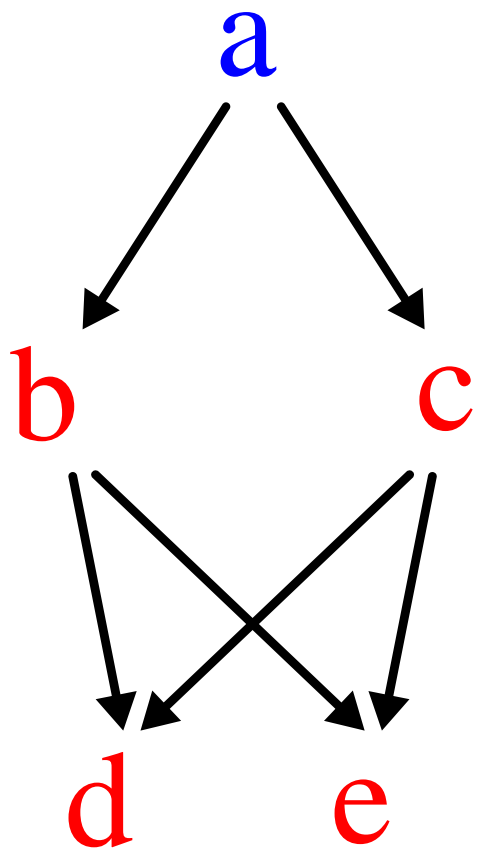
`//*/self::b/child::d`

1. `typeinf(DTD,{`X_a`},//*)`



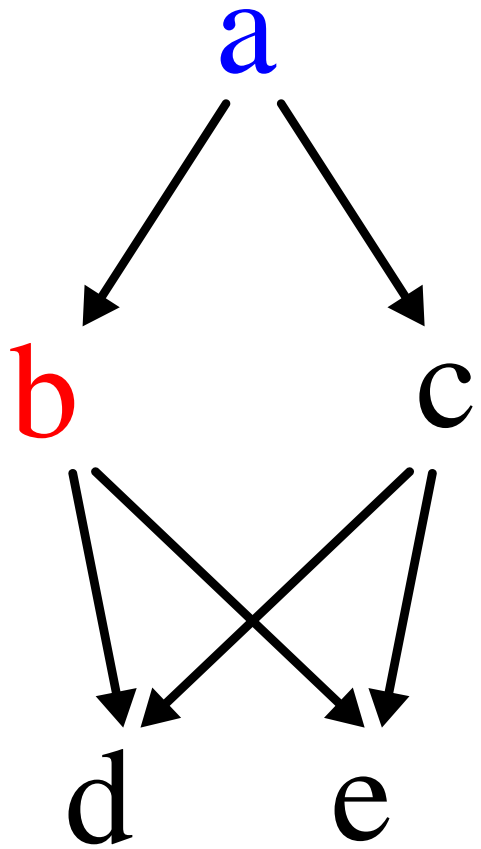
`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`



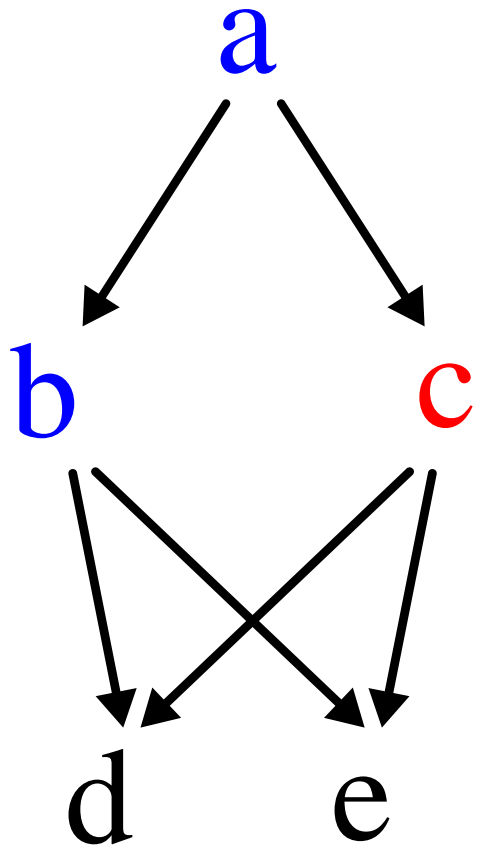
`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`
   1. `typeinf(DTD,`$\{X_b\}$`,self::b/child::d)` $= \{X_d\}$



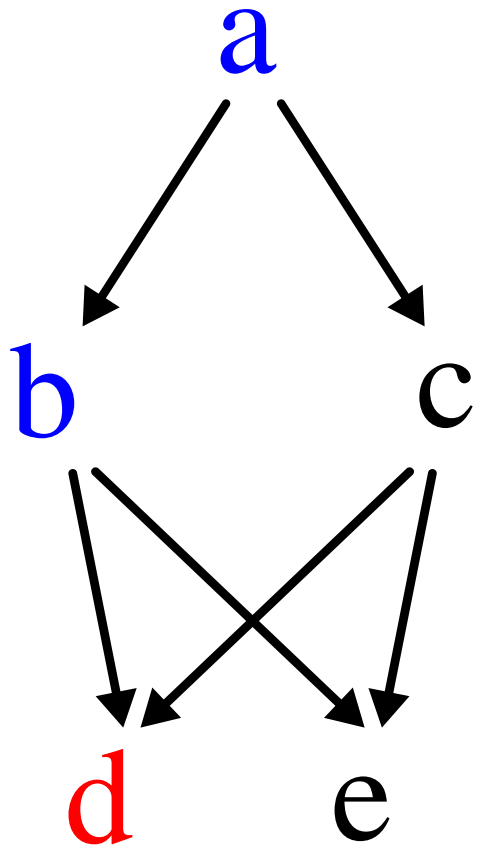
`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`
   1. `typeinf(DTD,`$\{X_b\}$`,self::b/child::d)` $= \{X_d\}$
   2. `typeinf(DTD,`$\{X_c\}$`,self::b/child::d)` $= \varnothing$



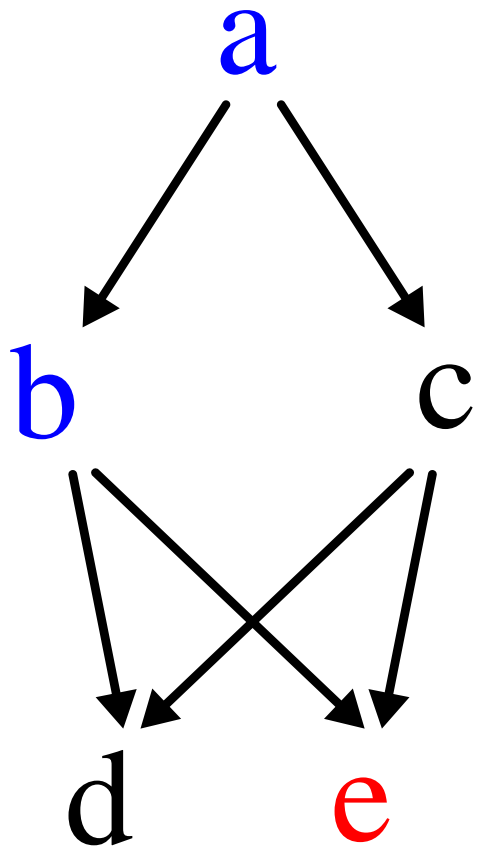
`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`
   1. `typeinf(DTD,`$\{X_b\}$`,self::b/child::d)` $= \{X_d\}$
   2. `typeinf(DTD,`$\{X_c\}$`,self::b/child::d)` $= \varnothing$
   3. `typeinf(DTD,`$\{X_d\}$`,self::b/child::d)` $= \varnothing$



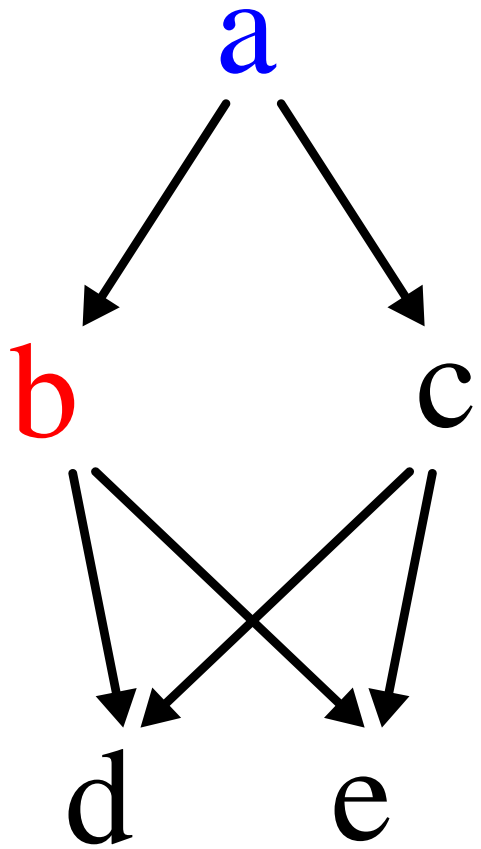
//*/self::b/child::d

1. typeinf(DTD,$\{X_a\}$,//*)
   1. typeinf(DTD,$\{X_b\}$,self::b/child::d)$= \{X_d\}$
   2. typeinf(DTD,$\{X_c\}$,self::b/child::d)$= \varnothing$
   3. typeinf(DTD,$\{X_d\}$,self::b/child::d)$= \varnothing$
   4. typeinf(DTD,$\{X_e\}$,self::b/child::d)$= \varnothing$



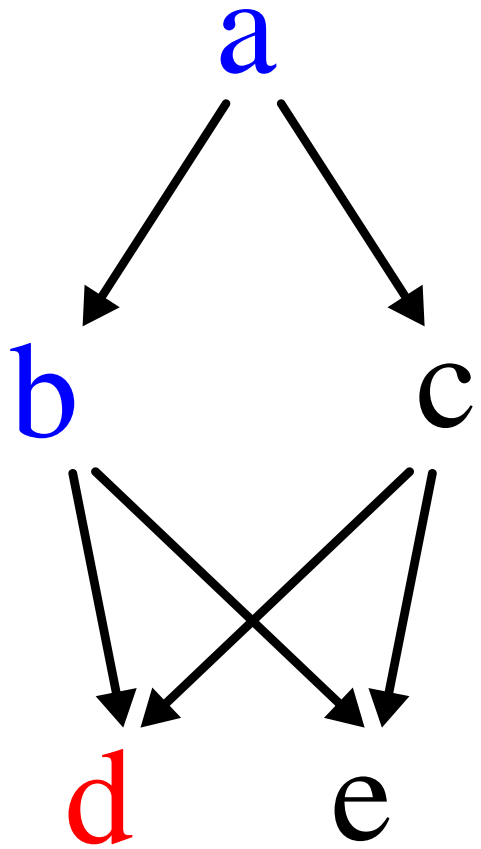
`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`
2. `typeinf(DTD,`$\{X_b\}$`,self::b)`$= \{X_b\}$



`//*/self::b/child::d`

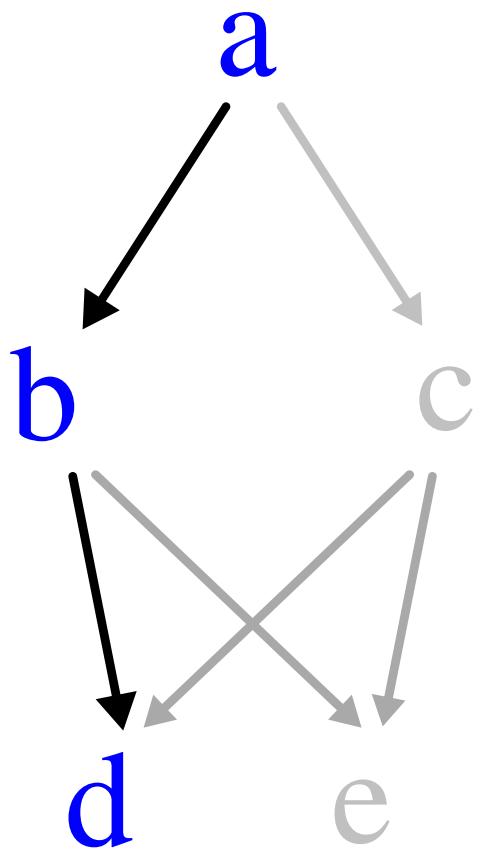
1. `typeinf(DTD,`$\{X_a\}$`,//*)`
2. `typeinf(DTD,`$\{X_b\}$`,self::b)`$= \{X_b\}$
3. `typeinf(DTD,`$\{X_b\}$`,child::d)`$= \{X_d\}$



`//*/self::b/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,//*)`
2. `typeinf(DTD,`$\{X_b\}$`,self::b)`$= \{X_b\}$
3. `typeinf(DTD,`$\{X_b\}$`,child::d)`$= \{X_d\}$

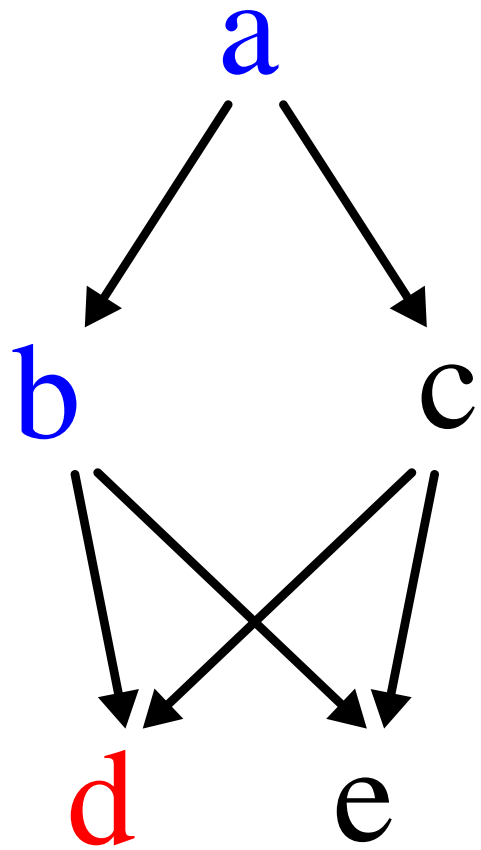
$$\mathcal{P} = \{X_a, X_b, X_d\}$$



```
/self::a/child::b/child::d/parent::node()/child::d
```

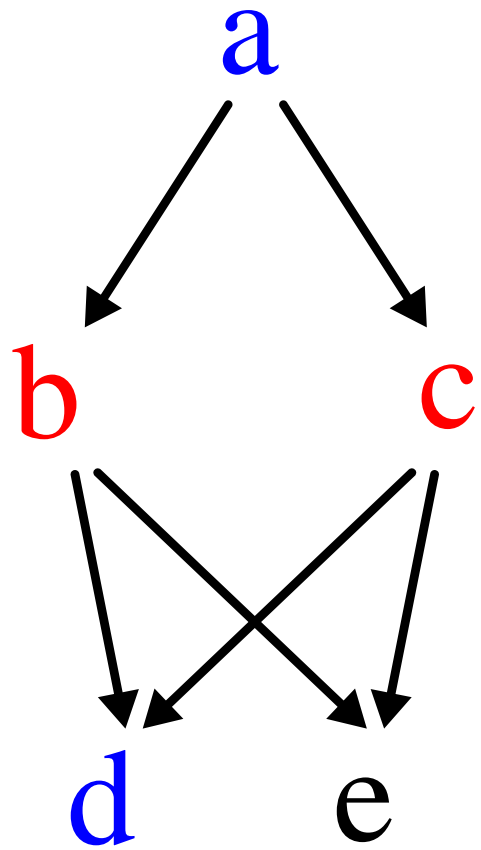
/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD,$\{X_a\}$,self::a)$= \{X_a\}$
2. typeinf(DTD,$\{X_a\}$,child::b)$= \{X_b\}$
3. typeinf(DTD,$\{X_b\}$,child::d)$= \{X_d\}$



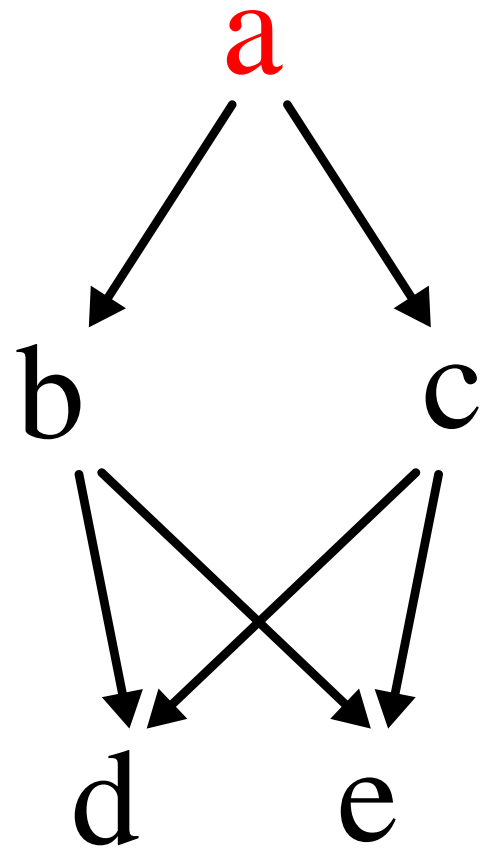
/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD,$\{X_a\}$,self::a)$= \{X_a\}$
2. typeinf(DTD,$\{X_a\}$,child::b)$= \{X_b\}$
3. typeinf(DTD,$\{X_b\}$,child::d)$= \{X_d\}$
4. typeinf(DTD,$\{X_d\}$,parent::node())$= \{X_b, X_c\}$



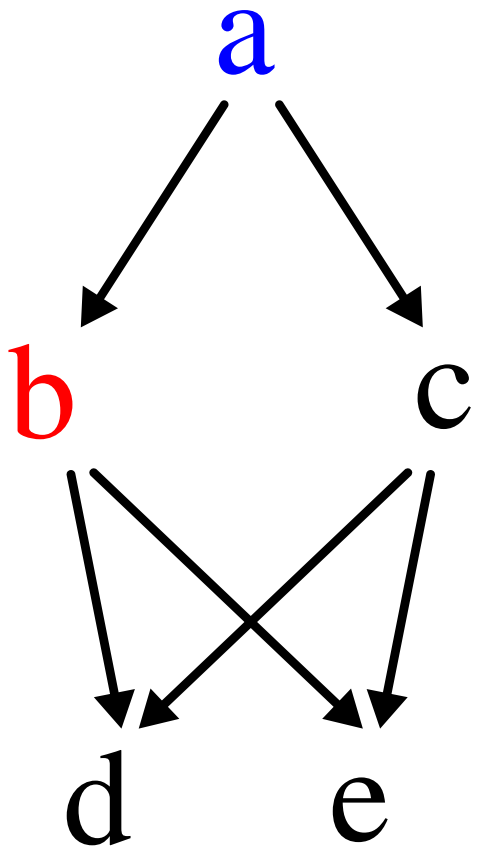
/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD,$\{X_a\}$,$\{X_a\}$,self::a)$= \{X_a\}$



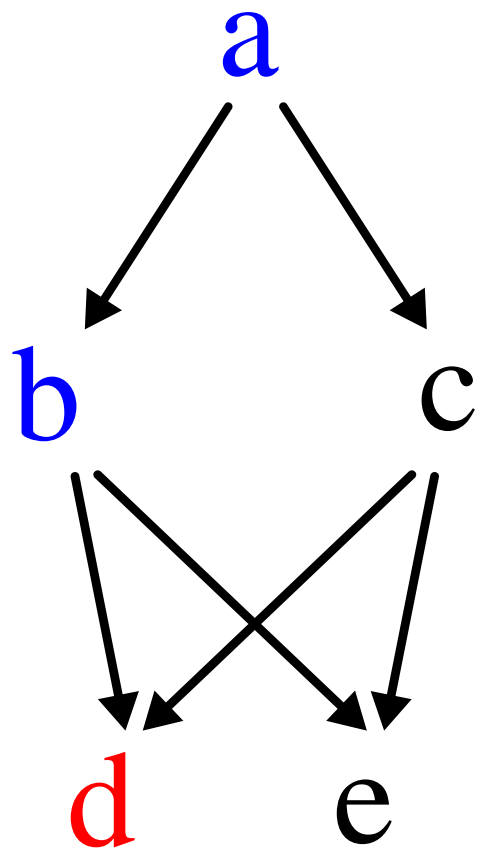
`/self::a/child::b/child::d/parent::node()/child::d`

1. `typeinf(DTD,`$\{X_a\}$`,`$\{X_a\}$`,self::a)`$= \{X_a\}$
2. `typeinf(DTD,`$\{X_a\}$`,`$\{X_a\}$`,child::b)`$= \{X_b\}$



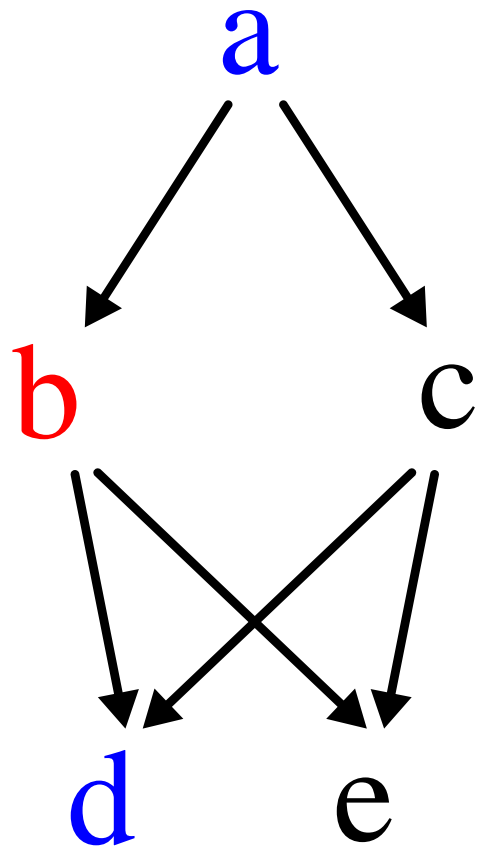
/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD,$\{X_a\}$,$\{X_a\}$,self::a)$= \{X_a\}$
2. typeinf(DTD,$\{X_a\}$,$\{X_a\}$,child::b)$= \{X_b\}$
3. typeinf(DTD,$\{X_b\}$,$\{X_a, X_b\}$,child::d)$= \{X_d\}$



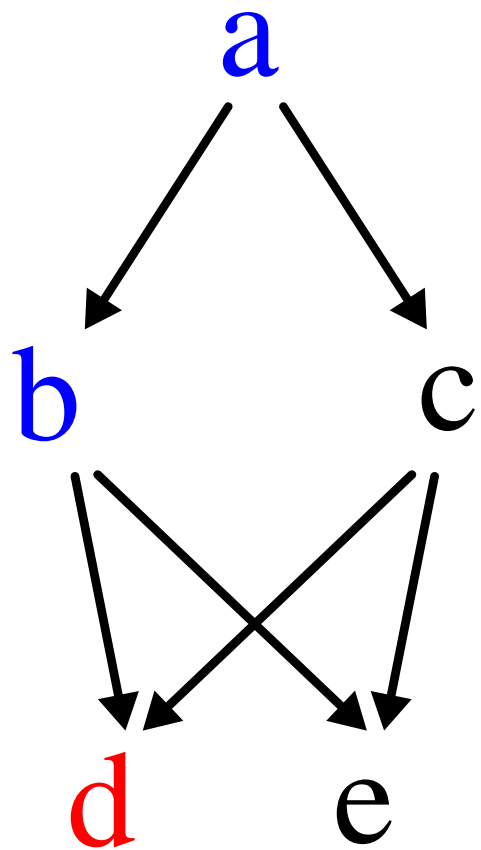
/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD,$\{X_a\}$,$\{X_a\}$,self::a)$= \{X_a\}$
2. typeinf(DTD,$\{X_a\}$,$\{X_a\}$,child::b)$= \{X_b\}$
3. typeinf(DTD,$\{X_b\}$,$\{X_a, X_b\}$,child::d)$= \{X_d\}$
4. typeinf(DTD,$\{X_d\}$,$\{X_a, X_b, X_d\}$,parent::node())$= \{X_b\}$



/self::a/child::b/child::d/parent::node()/child::d

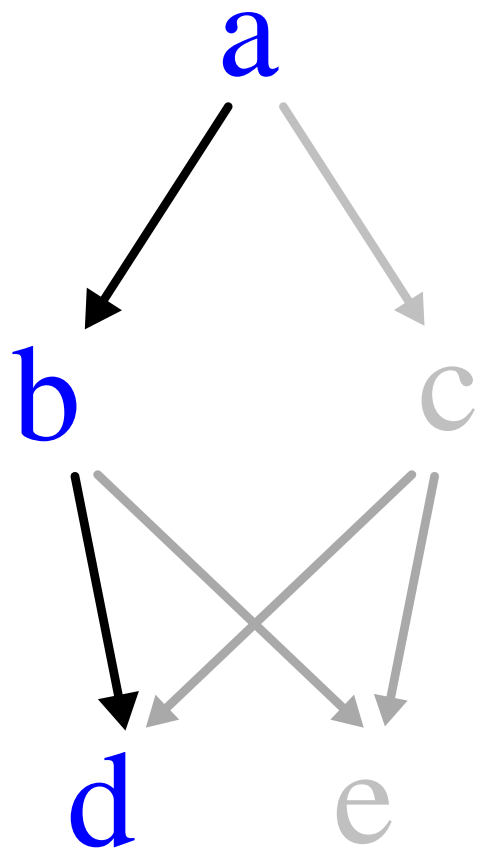
1. typeinf(DTD, $\{X_a\}$, $\{X_a\}$, self::a) $= \{X_a\}$
2. typeinf(DTD, $\{X_a\}$, $\{X_a\}$, child::b) $= \{X_b\}$
3. typeinf(DTD, $\{X_b\}$, $\{X_a, X_b\}$, child::d) $= \{X_d\}$
4. typeinf(DTD, $\{X_d\}$, $\{X_a, X_b, X_d\}$, parent::node()) $= \{X_b\}$
5. typeinf(DTD, $\{X_b\}$, $\{X_a, X_b, X_d\}$, child::d) $= \{X_d\}$



/self::a/child::b/child::d/parent::node()/child::d

1. typeinf(DTD, $\{X_a\}$, $\{X_a\}$, self::a) $= \{X_a\}$
2. typeinf(DTD, $\{X_a\}$, $\{X_a\}$, child::b) $= \{X_b\}$
3. typeinf(DTD, $\{X_b\}$, $\{X_a, X_b\}$, child::d) $= \{X_d\}$
4. typeinf(DTD, $\{X_d\}$, $\{X_a, X_b, X_d\}$, parent::node()) $= \{X_b\}$
5. typeinf(DTD, $\{X_b\}$, $\{X_a, X_b, X_d\}$, child::d) $= \{X_d\}$

$$\mathcal{P} = \{X_a, X_b, X_d\}$$



Outline

Theorem (Soundness)

Let D be a document valid w.r.t a DTD (X, E) and p a path. Let $\mathcal{P}$ the type projector deduced from p. Let D' be the projection of D with $\mathcal{P}$.

$$\mathtt{eval}(p, D) = \mathtt{eval}(p, D')$$

Completeness

Pruning is precise . . .

Pruning is precise . . .

Theorem (Completeness)

$\mathcal{P} = \{X_1, \ldots, X_n\}$ the type projector associated with a path p and a DTD (X, E). Let $\mathcal{P}' = \mathcal{P} \backslash \{X_i\}$. There exists D a document and it's projection D' such that :

$$\texttt{eval}(p, D) \neq \texttt{eval}(p, D')$$

Completeness holds with :

- some restrictions on the DTD (*star-guarded*, non-recursive,. . .)

Pruning is precise . . .

Theorem (Completeness)

$\mathcal{P} = \{X_1, \ldots, X_n\}$ the type projector associated with a path p and a DTD (X, E). Let $\mathcal{P}' = \mathcal{P} \backslash \{X_i\}$. There exists D a document and it's projection D' such that :

$$\texttt{eval}(p, D) \neq \texttt{eval}(p, D')$$

Completeness holds with :

- some restrictions on the DTD (*star-guarded*, non-recursive,. . .)
- some restrictions on the path (no upward axis in predicates, . . .)

Pruning is precise . . .

Theorem (Completeness)

$\mathcal{P} = \{X_1, \ldots, X_n\}$ the type projector associated with a path p and a DTD (X, E). Let $\mathcal{P}' = \mathcal{P} \backslash \{X_i\}$. There exists D a document and it's projection D' such that :

$$\texttt{eval}(p, D) \neq \texttt{eval}(p, D')$$

Completeness holds with :

- some restrictions on the DTD (*star-guarded*, non-recursive,. . .)
- some restrictions on the path (no upward axis in predicates, . . .)

Pruning is precise . . .

Theorem (Completeness)

$\mathcal{P} = \{X_1, \ldots, X_n\}$ the type projector associated with a path p and a DTD (X, E). Let $\mathcal{P}' = \mathcal{P} \backslash \{X_i\}$. There exists D a document and it's projection D' such that :

$$\texttt{eval}(p, D) \neq \texttt{eval}(p, D')$$

Completeness holds with :

- some restrictions on the DTD (*star-guarded*, non-recursive,. . .)
- some restrictions on the path (no upward axis in predicates, . . .)

Outline

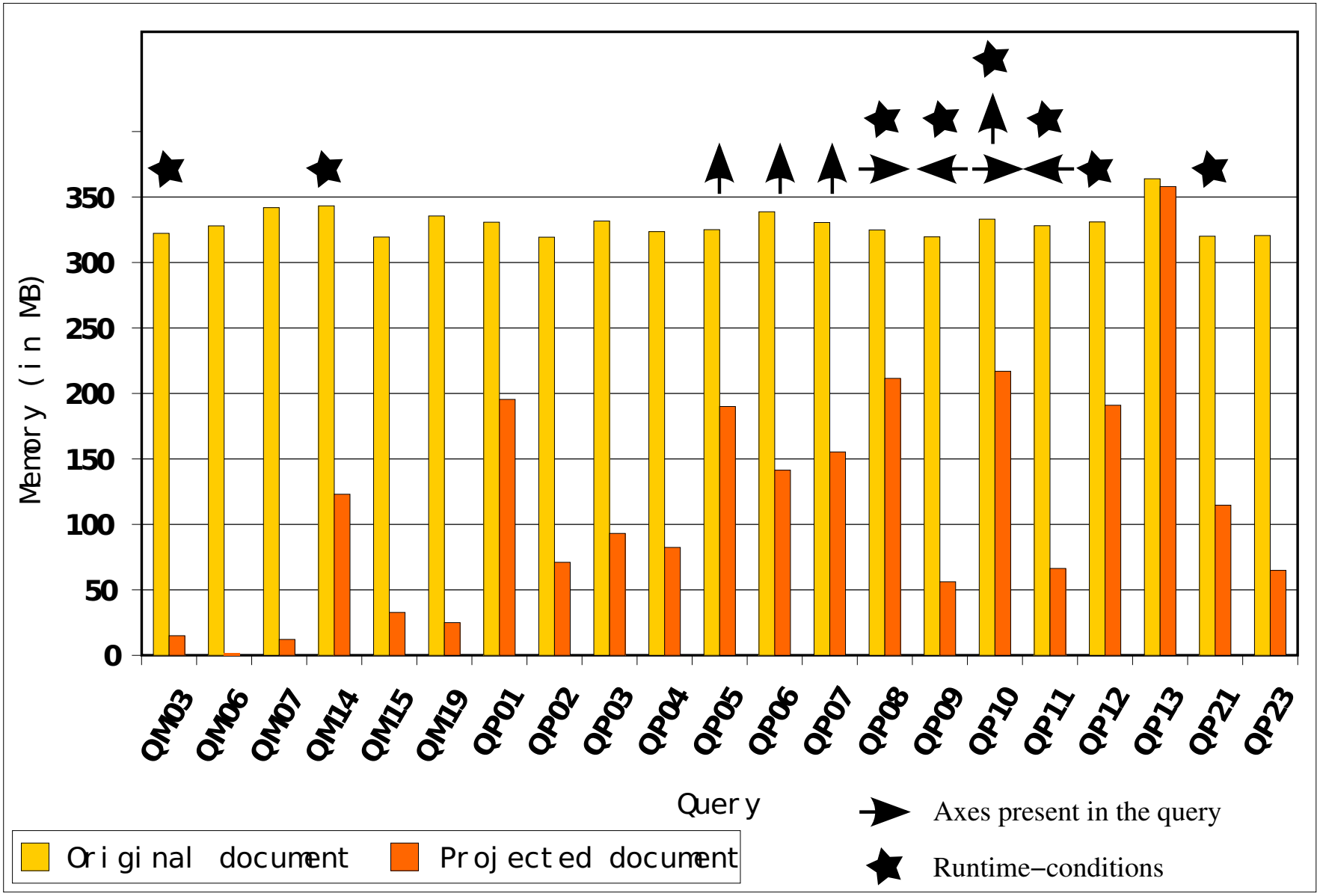
Protocol :

- Linux desktop, 512MB Ram, 3Ghz x86 CPU (and no swap).
- Implementation in OCaml.
- Validity of the result checked with the Galax query engine.
- Pruning tested on XMark and XPathMark benchmarks.

Protocol :

- Linux desktop, 512MB Ram, 3Ghz x86 CPU (and no swap).
- Implementation in OCaml.
- Validity of the result checked with the Galax query engine.
- Pruning tested on XMark and XPathMark benchmarks.

- Prunning is one pass bufferless traversal of the document.
- In practice, computing the type projector is fast.



Memory used to process a 56 MB document with Galax.

Gain

	QM03	QM06	QM07	QM14	QM15	QM19
	★			★		
Original Size (MB)	930	2048	1100	202	2048	964
Pruned Size(MB)	25	5,3	42	139	24	24
Memory Usage (MB)	374	90	380	512	245	512
% of original size	2.5	0.3	3.4	69.6	1.15	2.5
Gain in Speed ($\times$ faster)	17.8	110.1	28.2	3.9	62.6	7.5

Comparison with previous work

Qualitative : with one exception, type projectors are always equal or more efficient.

Comparison with previous work

Qualitative : with one exception, type projectors are always equal or more efficient.

Performances : pruning is linear in time (in the size of the document) and constant in memory. *It can be done while validating the document*.

Comparison with previous work

Qualitative : with one exception, type projectors are always equal or more efficient.

Performances : pruning is linear in time (in the size of the document) and constant in memory. *It can be done while validating the document*.

Features : handles backward as well as following/preceding axes.

Outline

Conclusion

- Formal foundations ensure validity of the pruning and it's efficiency.

Conclusion

- Formal foundations ensure validity of the pruning and it's efficiency.
- Handle any XQuery query, either directly or by approximating it.

Conclusion

- Formal foundations ensure validity of the pruning and it's efficiency.
- Handle any XQuery query, either directly or by approximating it.
- Whereas approximations are made for runtime conditions, the technique still preserve a *high degree of precision*.

Conclusion

- Formal foundations ensure validity of the pruning and it's efficiency.
- Handle any XQuery query, either directly or by approximating it.
- Whereas approximations are made for runtime conditions, the technique still preserve a *high degree of precision*.
- *No additional cost at runtime*.

Future work

Many directions :

- adapt our approach to work on *untyped* documents by using data-guides or path summaries.

Future work

Many directions :

- adapt our approach to work on *untyped* documents by using data-guides or path summaries.
- extend the formalism to handle XML-Schema rather than DTDs.

Future work

Many directions :

- adapt our approach to work on *untyped* documents by using data-guides or path summaries.
- extend the formalism to handle XML-Schema rather than DTDs.
- integration with classical databases techniques.

Future work

Many directions :

- adapt our approach to work on *untyped* documents by using data-guides or path summaries.
- extend the formalism to handle XML-Schema rather than DTDs.
- integration with classical databases techniques.
- integration with a query engine.