Schemas for safe and efficient XML processing

Dario Colazzo

Université Paris Sud - INRIA

Ecole Thématique BDA 2010 - Les Houches

Plan

- XML & XML schema
- Correctness and result analysis
- Schema based projection
- Constraint based approach for efficient subtyping and validation

Plan

- XML & XML schema
- Correctness and result analysis
- Schema based projection
- Constraint based approach for efficient subtyping and validation.

Informal,
examples, and main ideas

~ 15 minutes

Plan

- XML & XML schema
- Correctness and result analysis
- Schema based projection
- Constraint based approach for efficient subtyping and validation

Informal,
examples, and main ideas

~ 15 minutes

More formal,
examples + some defs

~ 30 minutes

What are XML schemas useful for ?

- To define structural constraints over documents: this is usefeul in many contexts.
- How: mainly by means of **regular expressions.**
- Main schema languages: DTDs, XML Schema, Relax-NG.
- For all of them, methods for automatic **validation** exist.
- For XML queries over XML **valid** documents we can
  - automatically check that the query correctly manipulate the input
  - automatically infer a schema for data produced by the query

XML query type-checking

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional ∀-correctness

Problems with an ∃-notion

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional ∀-correctness

Problems with an ∃-notion

We need ∀ quantification on sub-queries

and ∃ quantification on instances

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional $\forall$-correctness

Problems with an $\exists$-notion

We need $\forall$ quantification on sub-queries $\forall\exists$ correctness

and $\exists$ quantification on instances [PhD Thesis, ICFP04]

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional ∀-correctness

Problems with an ∃-notion

We need ∀ quantification on sub-queries   ∀∃ correctness

and ∃ quantification on instances   [PhD Thesis, ICFP04]

Query correctness

The quite famous biblio DTD

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Is it correct? Yes, intuitevely

Not according to the traditional ∀-correctness

Problems with an ∃-notion

We need ∀ quantification on sub-queries ∀∃ correctness

and ∃ quantification on instances [PhD Thesis, ICFP04]

Main tools for ∀∃ correctness

- A type system allowing to infer types of query paths:
  - doc//(author | editor) : (author | author)+
  - doc//(author | editor)/second : (second)+
- As a consequence, the type system allows to find types of elements **never needed** by the query (**all XPath axes can be handled**)
- This has been used for **type-based projection**: first types of needed nodes are inferred, and then this information is used to prune the input D in order to obtain a much smaller document D’ such that

$$Q(D)=Q(D')$$

Type based projection

Example

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

Query:

```
<res>
for x in doc//(author | editor)
return <nom>x/last</nom>
</res>
```

Example

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
.....
......
```

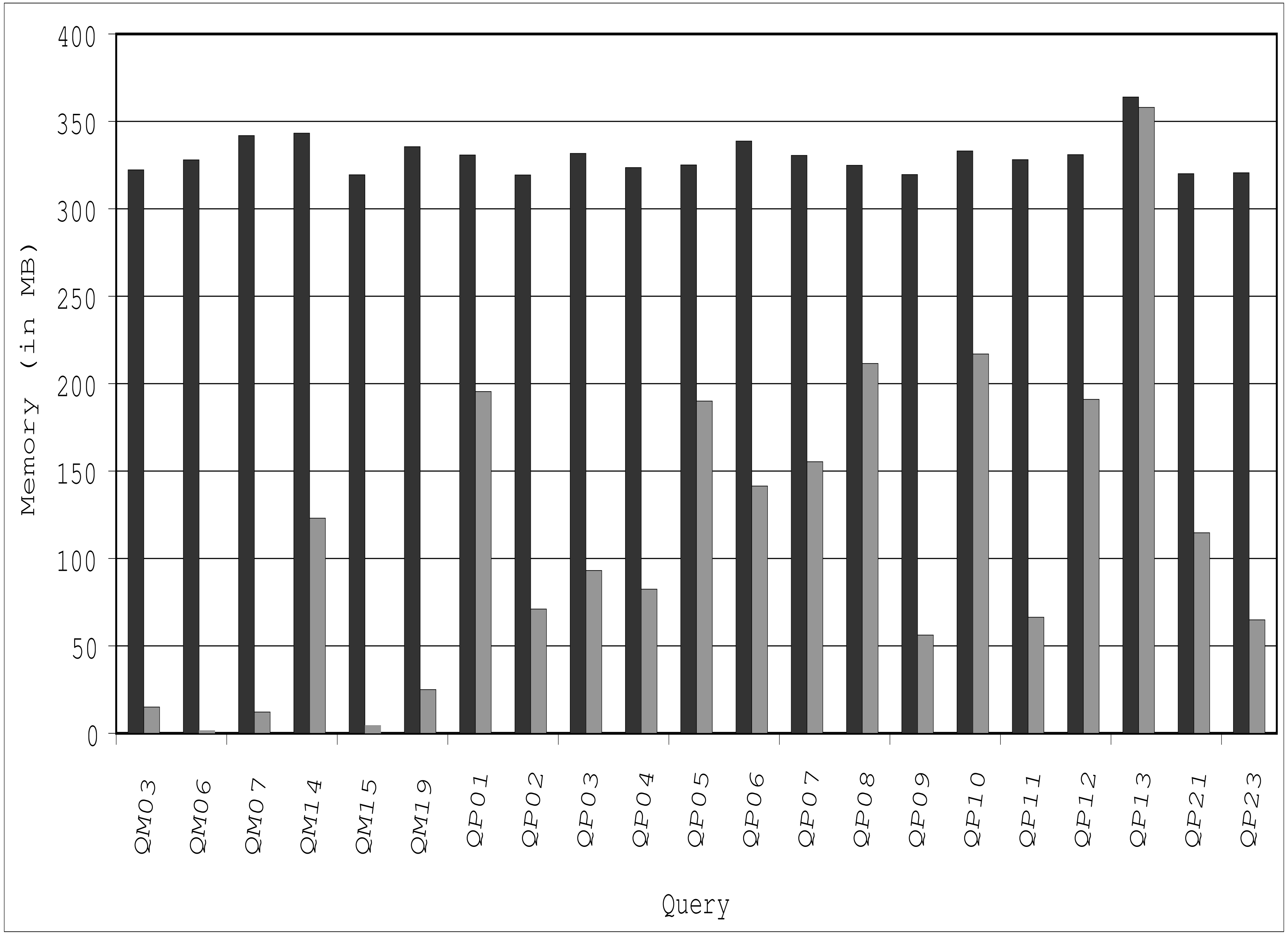
Type projector τ=(bib, book, author, editor, last)

Used at loading time: only τ elements are kept, the other ones are not loaded

Much less memory consumption: we can query quite big documents!

```
<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>

    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

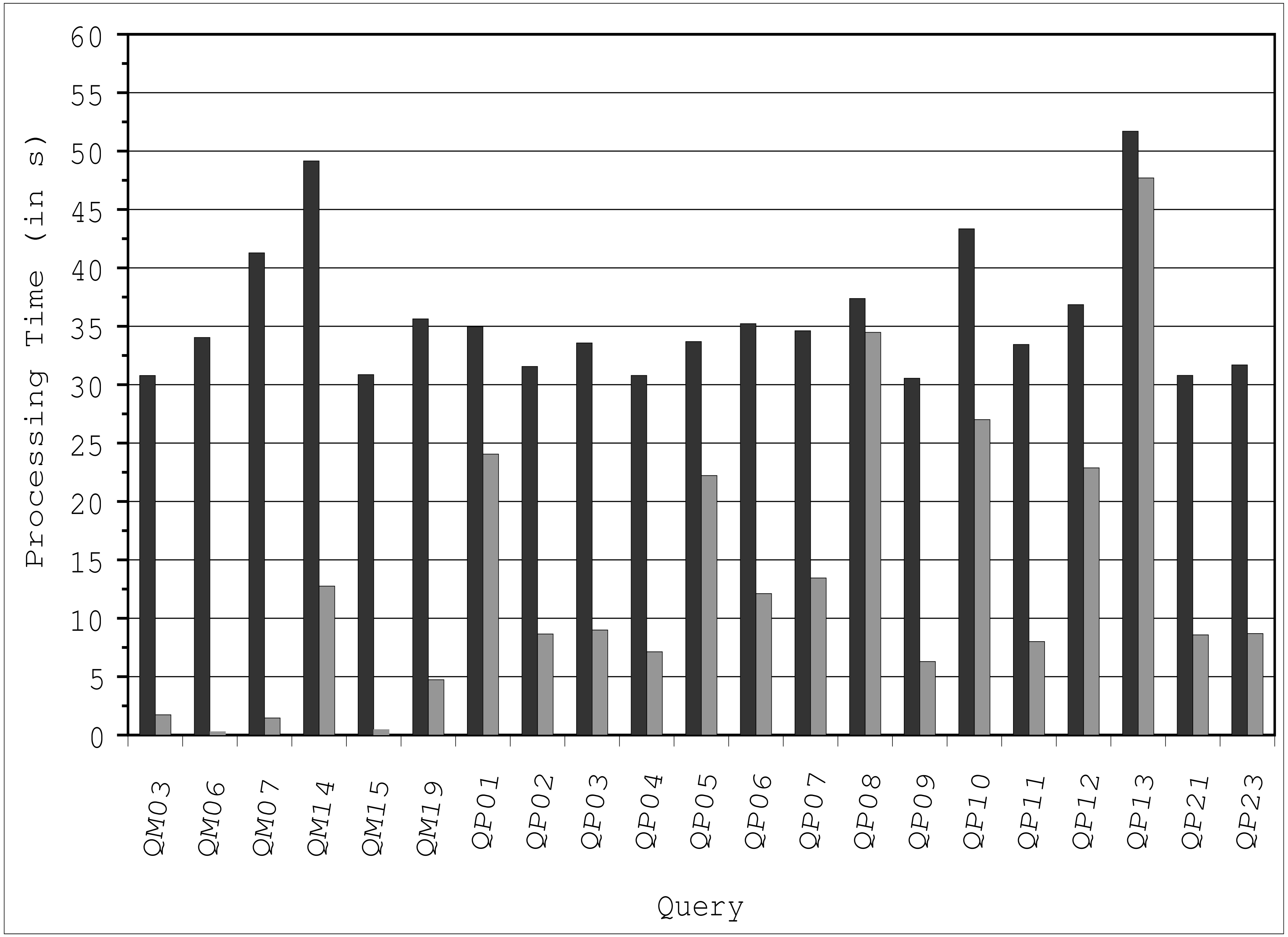
Test results: space [VLDB06]



XMark QMi

XPathMark QPj

Test results: time [VLDB06]



XMark QMi

XPathMark QPj

What about updates ?

- Type-based projection still ensures optmizations
- Amine Baazizi will give you more details
- Marina Sahakyan can answer questions about efficient implementation of the technique

Let’s go back to type inference

- Very important problem, crucial for **result anlaysis**
- Given Q over a schema S, does Q produce values of another expected schema S’

$$Q : S \dashrightarrow S' \quad ?$$

- Method:
  - automatic inference of a schema S_{out} for Q result values
  - automatic checking of inclusion $S_{out} \subseteq S'$
- Problem: schema inclusion has high complexity.
- We found out that for a wide class of schemas it can be efficiently checked. Next subject.

Constraints based subtype checking and validation

REs and XML types

- REs define element content models in XML schemas

  DTD : <!ELEMENT book (title, (author | editor)*, price?)>

REs and XML types

- REs define element content models in XML schemas

```
DTD :   <!ELEMENT book  (title, (author | editor)*, price?)>
```

REs and XML types

- REs define element content models in XML schemas

  DTD : <!ELEMENT book (title, (author | editor)*, price?)>

- Our syntax

$$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^*$$

REs and XML types

- REs define element content models in XML schemas

  DTD : <!ELEMENT book (title, (author | editor)*, price?)>

- Our syntax

$$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^*$$

$$\mathrm{title} \cdot (\mathrm{author} + \mathrm{editor})^* \cdot (\mathrm{price} + \varepsilon)$$

Interleaving and counting

$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T \& T \mid T[n..m]$

Interleaving and counting

$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$

- $m \in N \cup \{*\}$

Interleaving and counting

$$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$$

- $m \in N \cup \{*\}$
- $a = a[1..1]$ $a? = a + \varepsilon$ $a^* = a[1..*]$

Interleaving and counting

$$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$$

- $m \in N \cup \{*\}$
- $a = a[1..1]$ $\quad a? = a + \varepsilon$ $\quad a^* = a[1..*]$
- $L(b[1..4]) = \{b, bb, bbb, bbbb\}$
- $L(a\&b) = \{ab, ba\}$

Interleaving and counting

$$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$$

- $m \in N \cup \{*\}$
- $a = a[1..1]$ $a? = a + \varepsilon$ $a^* = a[1..*]$
- $L(b[1..4]) = \{b, bb, bbb, bbbb\}$
- $L(a\&b) = \{ab, ba\}$

Interleaving and counting

$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$

- $m \in N \cup \{*\}$
- $a = a[1..1]$ $a? = a + \varepsilon$ $a^* = a[1..*]$
- $L(b[1..4]) = \{b, bb, bbb, bbbb\}$
- $L(a\&b) = \{ab, ba\}$
- $L((a \cdot b) \& c) = \{abc, cab, acb\}$ $cba \notin L((a \cdot b) \& c)$

Interleaving and counting

$T ::= \varepsilon \mid a \mid T + T \mid T \cdot T \mid T^* \mid T\&T \mid T[n..m]$

- $m \in N \cup \{*\}$
- $a = a[1..1]$ $a? = a + \varepsilon$ $a^* = a[1..*]$
- $L(b[1..4]) = \{b, bb, bbb, bbbb\}$
- $L(a\&b) = \{ab, ba\}$
- $L((a \cdot b) \& c) = \{abc, cab, acb\}$ $cba \notin L((a \cdot b) \& c)$

Interleaving

- Interleaving is used in XML type languages
- RELAX-NG <interleave> … </interleave>
- The all group of XSD:

```
<xsd:complexType name="PurchaseOrderType">
    <xsd:all>
        <xsd:element name="billTo" type="USAddress"/>
        <xsd:element ref="comment" minOccurs="0"/>
        <xsd:element name="items" type="Items"/>
    </xsd:all>
</xsd:complexType>
```

The cost of Interleaving

- Membership
  - RE : PTime
  - RE with & : NP-complete
- Inclusion
  - RE: PSPACE (EXPTIME for EDTDs) complete
  - RE with & : EXPSPACE complete
- Our conflict-free expressions:
  - Inclusion: quadratic [IS09]
  - Membership: linear [CIKM08]

Our conflict-free REs

$$T ::= \varepsilon \mid a[m..n] \mid T + T \mid T \cdot T \mid T \& T$$

Our conflict-free REs

$$T ::= \varepsilon \mid a[m..n] \mid T + T \mid T \cdot T \mid T\&T$$

- Two restrictions:

1. repetition T* restricted to a* (denoting a[1..*]+ε)
2. single occurrence:

(a+b·a+a·c) : no

(a·b?) : ok

- Are these restrictions acceptable ?

[BexNevenSchwentickTuyls-VLDB06]: “An examination of 819 DTDs and XSDs … more than 99% of the REs occurring in practical schema’s are CHAREs”

Types as constraints

$T = ((a[1..3] \cdot b[2..2]) + c[1..*])$ and w in $L(T)$

Types as constraints

$$T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T)$$

- **lower-bound** (*nillability*): at least one of {*a*,*b*,*c*}=S(T) is in *w*;

Types as constraints

$T = ((a[1..3] \cdot b[2..2]) + c[1..*])$ and w in $L(T)$

- **lower-bound** (*nillability*): at least one of {*a*,*b*,*c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a*, *b*, *c*} is in w;

Types as constraints

$$T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T)$$

- **lower-bound** (*nillability*): at least one of {*a,b,c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a, b, c*} is in w;
- **cardinality**: if *a* is in w, it appears 1, 2 or 3 times; if *b* is there, it appears twice…

Types as constraints

$T = ((a[1..3] \cdot b[2..2]) + c[1..*])$ and w in $L(T)$

- **lower-bound** (*nillability*): at least one of {*a,b,c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a, b, c*} is in w;
- **cardinality**: if *a* is in w, it appears 1, 2 or 3 times; if *b* is there, it appears twice…
- **exclusion**: any of {*a,b*} excludes *c*
  c excludes any of {*a,b*}

Types as constraints

T = ((a[1..3]·b[2..2]) + c[1..*])   and w in L(T)

- **lower-bound** (*nillability*): at least one of {*a,b,c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a, b, c*} is in w;
- **cardinality**: if *a* is in w, it appears 1, 2 or 3 times; if *b* is there, it appears twice…
- **exclusion**: any of {*a,b*} excludes *c*
  c excludes any of {*a,b*}
- **co-occurrence**: *a* requires *b*; *b* requires *a*

Types as constraints

T = ((a[1..3]·b[2..2]) + c[1..*])   and w in L(T)

- **lower-bound** (*nillability*): at least one of {*a,b,c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a, b, c*} is in w;
- **cardinality**: if *a* is in w, it appears 1, 2 or 3 times; if *b* is there, it appears twice…
- **exclusion**: any of {*a,b*} excludes *c*
  c excludes any of {*a,b*}
- **co-occurrence**: *a* requires *b*; *b* requires *a*
- **order:** any a comes before any b

Types as constraints

$T = ((a[1..3] \cdot b[2..2]) + c[1..*])$ and w in $L(T)$

- **lower-bound** (*nillability*): at least one of {*a,b,c*}=S(T) is in *w*;
- **upper-bound**: no symbol out of {*a, b, c*} is in w;
- **cardinality**: if *a* is in w, it appears 1, 2 or 3 times; if *b* is there, it appears twice…
- **exclusion**: any of {*a,b*} excludes *c*
  c excludes any of {*a,b*}
- **co-occurrence**: *a* requires *b*; *b* requires *a*
- **order:** any a comes before any b

This is a complete characterization of T !

Types as constraints

$$T = ((a[1..3] \cdot b[2..2]) + c[1..*]) \quad \text{and} \quad w \text{ in } L(T)$$

- **lower-bound** S(T)
- **upper-bound**: Upper(S(T))
- **cardinality**: $a?[1..3] \wedge b?[2..2] \wedge c?[1..*]$
- **exclusion**: $\{a,b\} <> \{c\}$
- **co-occurrence**: $a \Rightarrow b \wedge b \Rightarrow a$ (abbreviated as $a \Leftrightarrow b$)
- **order**: $a < b$

Constraints

F::= A | $A \Rightarrow B$ | a?[m..n] | upper(A) | $A<B$ | $F \wedge F'$

Constraints

F::= A | A⇒B | a?[m..n] | upper(A) | A<B | F ∧ F'

$w \vDash A$: $w{\downarrow}A \neq \emptyset$

$w \vDash A{=}{>}B$: if $w \vDash A$ then $w \vDash B$

$w \vDash A<B$: any *A* is before any *B*

$w \vDash a?[m..n]$: if a in w, then $m \leq |w{\downarrow}a| \leq n$

$w \vDash \mathrm{upper}(A)$: $S(w) \subseteq A$

Derived operators

- Double co-occurrence:
  - $A \Leftrightarrow B \Leftrightarrow_{def} A \Rightarrow B$ and $B \Rightarrow A$
- Mutual exclusion
  - $A <> B \Leftrightarrow_{def} A < B$ and $B < A$
  - corresponds to c-f union types $T_A + T_B$
- Negation
  - $\neg A \quad \Leftrightarrow_{def} \quad A \Rightarrow \varnothing$
  - False $\Leftrightarrow_{def} \varnothing$
  - True $\Leftrightarrow_{def} \varnothing \Rightarrow \varnothing$

Flat, order and co-occurrence constraints

- Each c-f type can be associated to a conjunction

$$F(T)=Flat(T) \wedge OC(T) \wedge CC(T)$$

- Theorem [IS09]: $w \in T \Leftrightarrow w \models F(T)$

- Theorem (subtyping)

$$T \subseteq U \Leftrightarrow T \models Ffat(U),\ T \models OC(T),\ T \models CC(T)$$

- Each of the 3 above entailements can be checked: independently and in O(n^2) time [IS09].

- In a recent work [ICDT09]: quadratic algorithm when only U is c-f

- Good news for result analysis: $Q : S \dashrightarrow S'$ via $S_{out} \subseteq S'$

Constraints for efficient validation

Main points

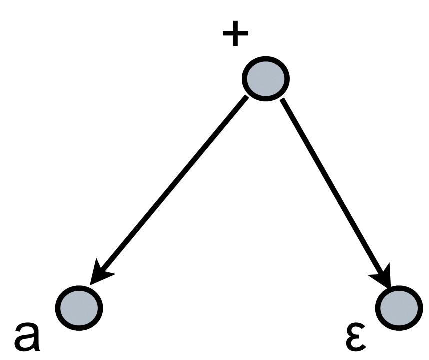
- XML schema validation ≃ RE membership
- Membership for RE+{interleaving, counting} is NP-complete
- Most of REs defined in real-life schemas are **conflict-free**
- Semantics of c-f REs can be captured by logical constraints (previous work at DBPL'07)
- Streaming RE membership checking via streaming constraint **residuation**
- Linear complexity!
- Extension to XML schema validation: immediate (see [CIKM08])

Constraints construction

$T = ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$

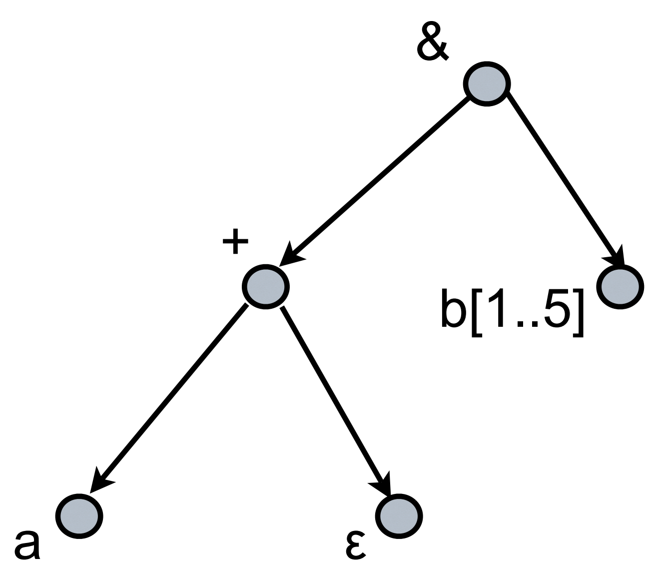
Constraints construction

$T = (\underline{(a+\varepsilon)}\&b[1..5]) + (c \cdot d[1..*])$



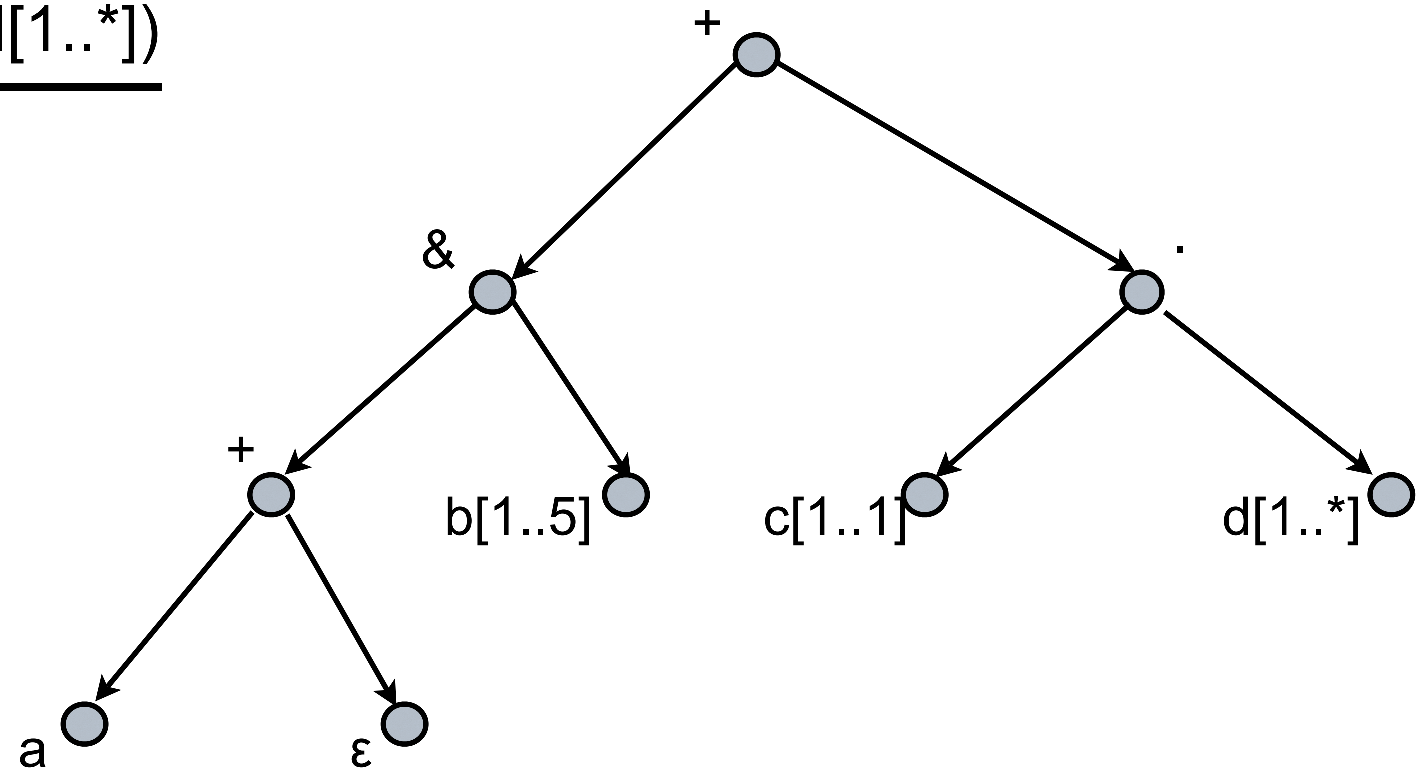
Constraints construction

T = ((a+ε)&b[1..5]) + (c·d[1..*])



Constraints construction

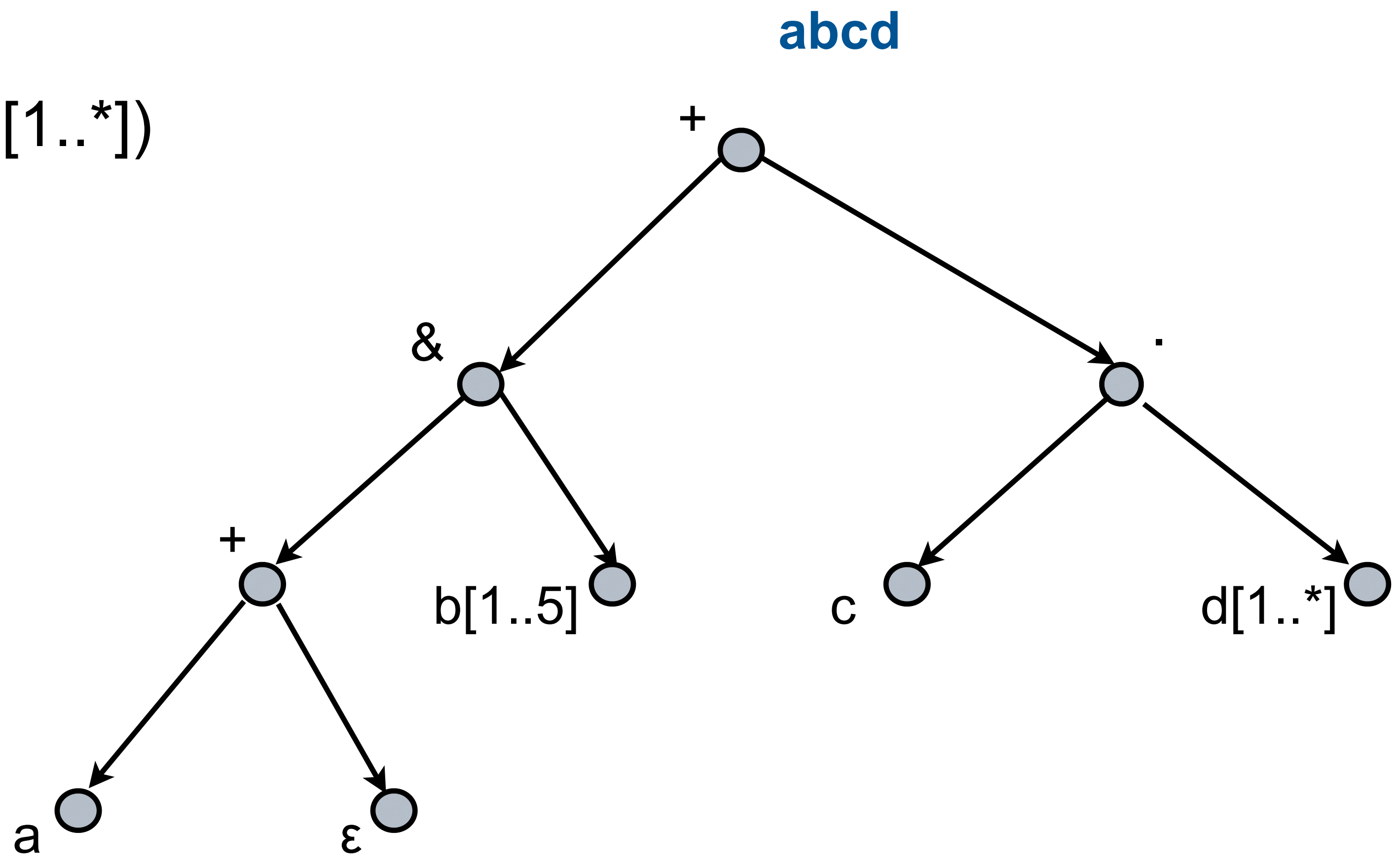
$T = \underline{((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])}$



Constraints construction

$T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$

T is not nillable: $\varepsilon \notin T$

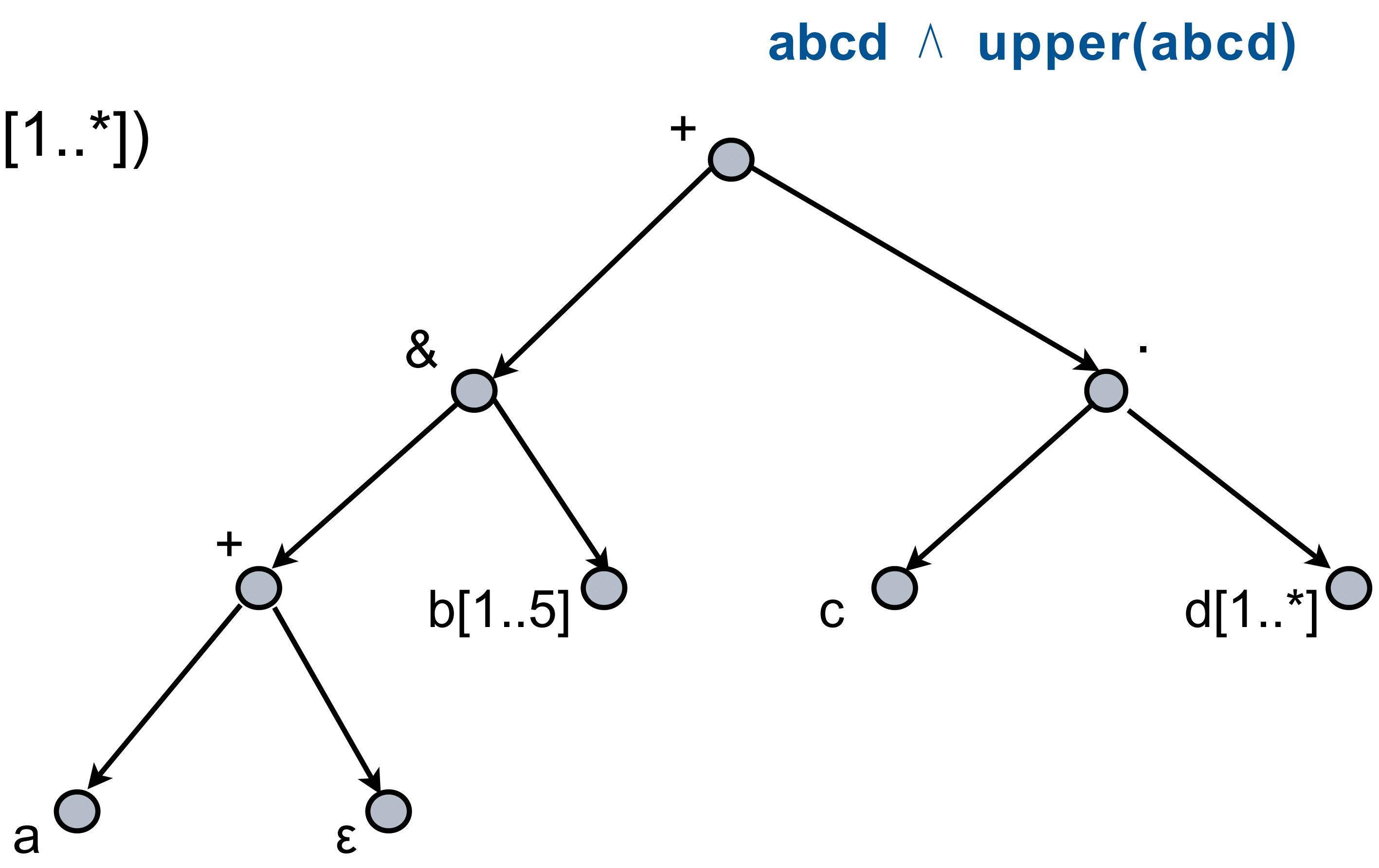


C(T) = **abcd** $\wedge$ **......**

Constraints construction

$T = ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$

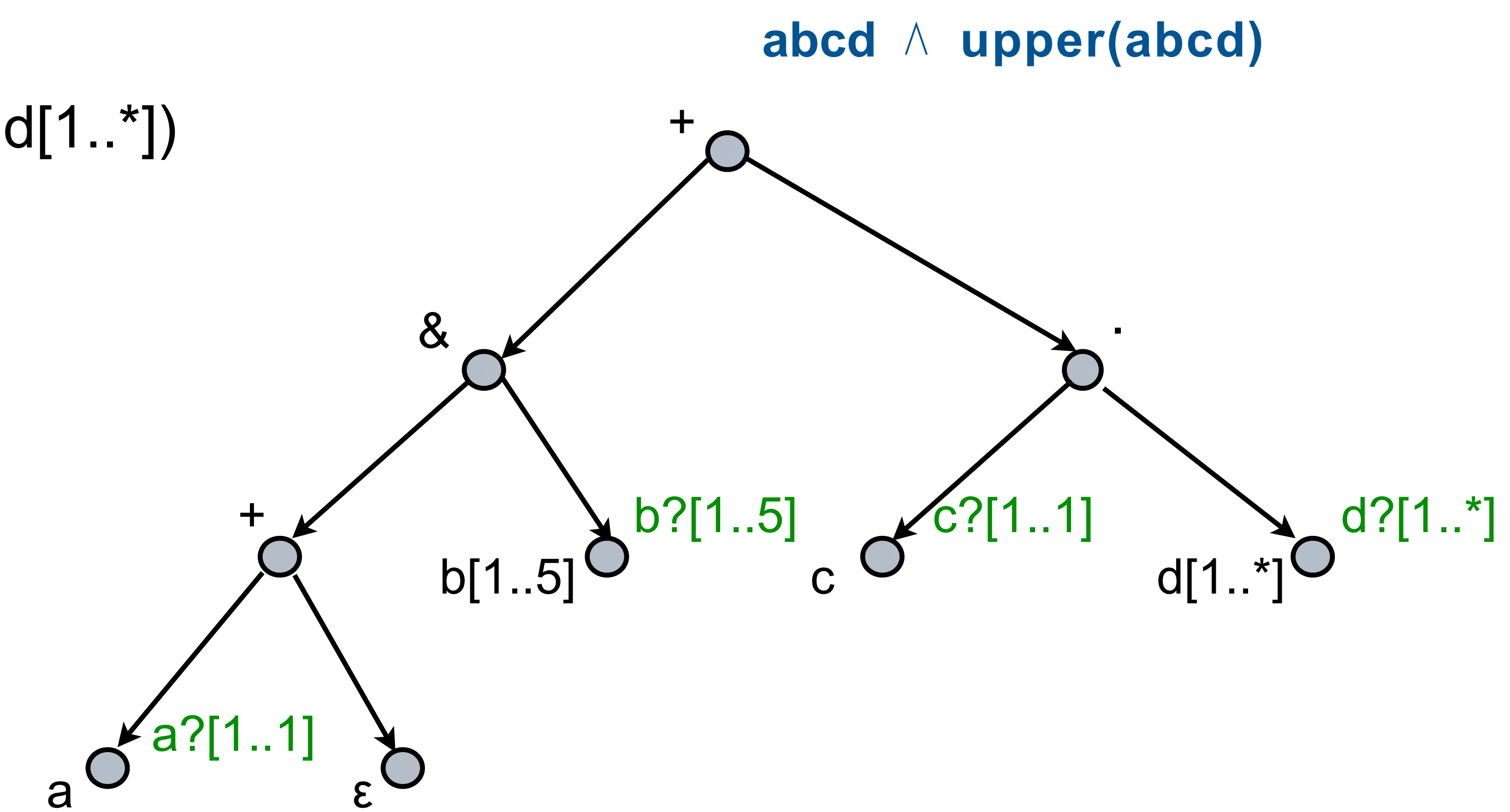
T is not nillable: $\varepsilon \notin T$



C(T) = **abcd ∧ upper(abcd) ......**

Constraints construction

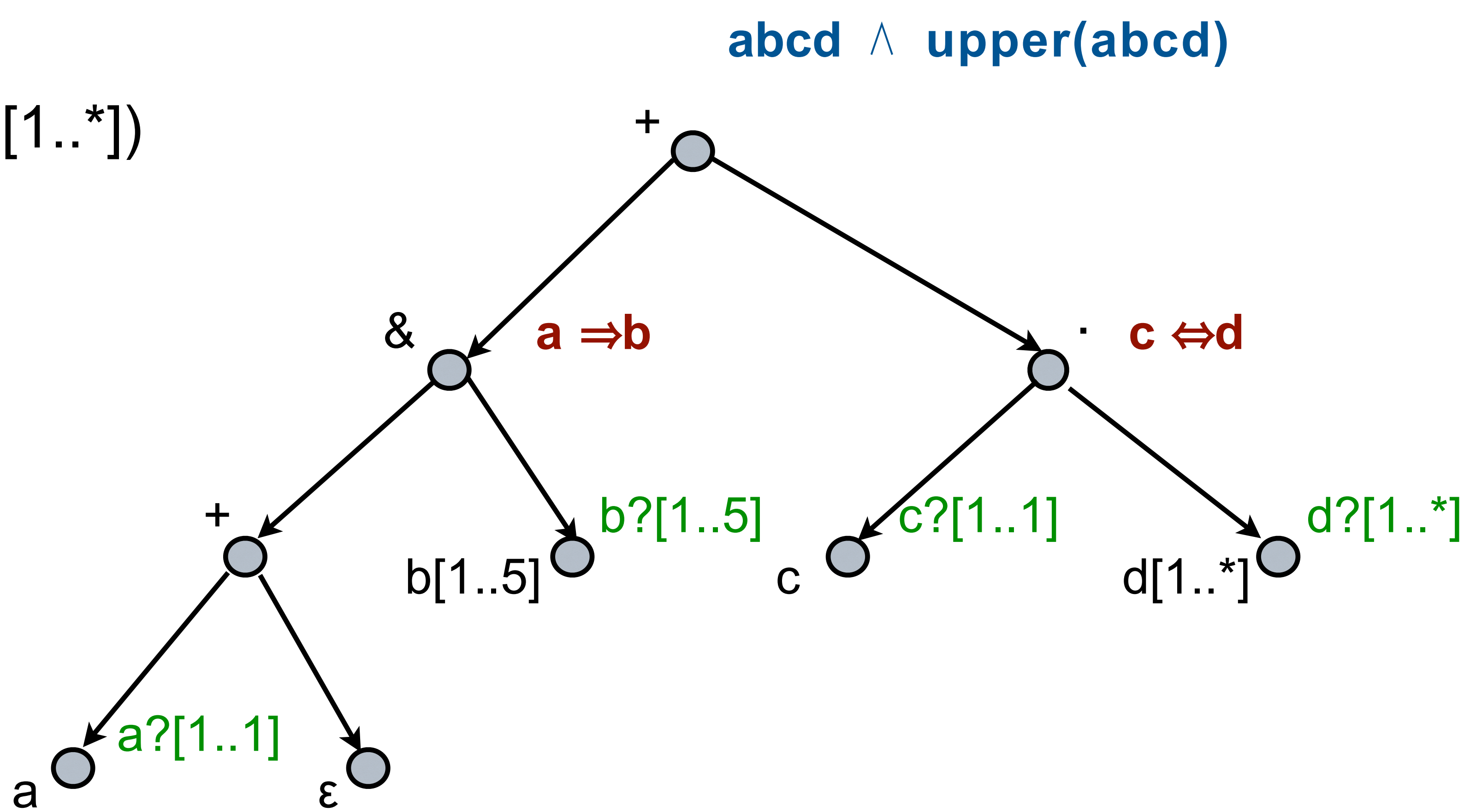
$T = ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$



C(T) = **abcd ∧ upper(abcd) ∧** a?[1..1] **∧ ....∧** d?[1..*]

Constraints construction

T = ((a+ε)&b[1..5]) + (c·d[1..*])

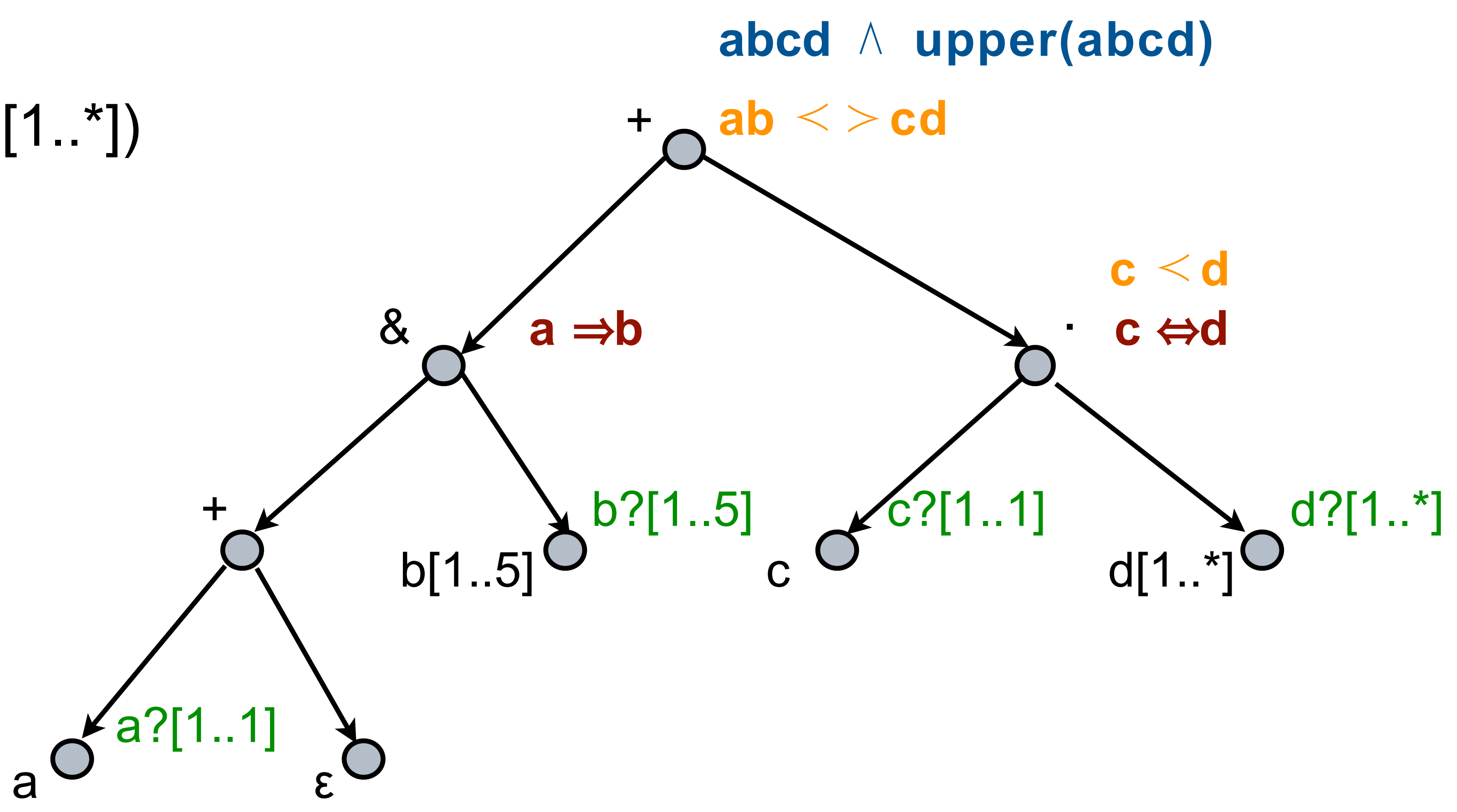


C(T) = **abcd ∧ upper(abcd) ∧** a?[1..1] **∧ ....∧** d?[1..*]

∧ a ⇒b ∧ c ⇔d

Constraints construction

$T = ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$



C(T) = **abcd** ∧ **upper(abcd)** ∧ a?[1..1] ∧ ....∧ d?[1..*]

ab < > cd ∧ c < d ∧ a ⇒b ∧ c ⇔d

Theorem : $w \in T \Leftrightarrow w \vDash C(T)$

Constraint membership

- We consider F=C(T) instead of T

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot \ldots\ldots \cdot a_n \models C(T)$ in a streaming fashion

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot ..... \cdot a_n \vDash C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1$$

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot \ldots\ldots \cdot a_n \vDash C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2$$

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot ..... \cdot a_n \models C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2 \xrightarrow{a_3} ..... \xrightarrow{a_n} Fn$$

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot ..... \cdot a_n \models C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2 \xrightarrow{a_3} ..... \xrightarrow{a_n} Fn$$

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot \ldots\ldots \cdot a_n \models C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2 \xrightarrow{a_3} \ldots\ldots \xrightarrow{a_n} Fn$$

residuals

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot \ldots\ldots \cdot a_n \models C(T)$ in a streaming fashion

$$F \xrightarrow{a_1} F1 \xrightarrow{a_2} F2 \xrightarrow{a_3} \ldots\ldots \xrightarrow{a_n} Fn$$

residuals

$$w \models C(T) \Leftrightarrow Fn \cap \{False, A\} = \varnothing$$

Constraint membership

- We consider F=C(T) instead of T
- We build a tree representation of C(T)
- We check $w=a_1 \cdot a_2 \cdot ..... \cdot a_n \models C(T)$ in a streaming fashion
- Important: flat constraints do not need residuation:
  - counting constraints a?[m..n] : keep some counters updated during the visit
  - lower and upper bound constraints: trivial.

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A < > B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A <> B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A <> B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A < > B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A < > B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A <> B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A < > B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

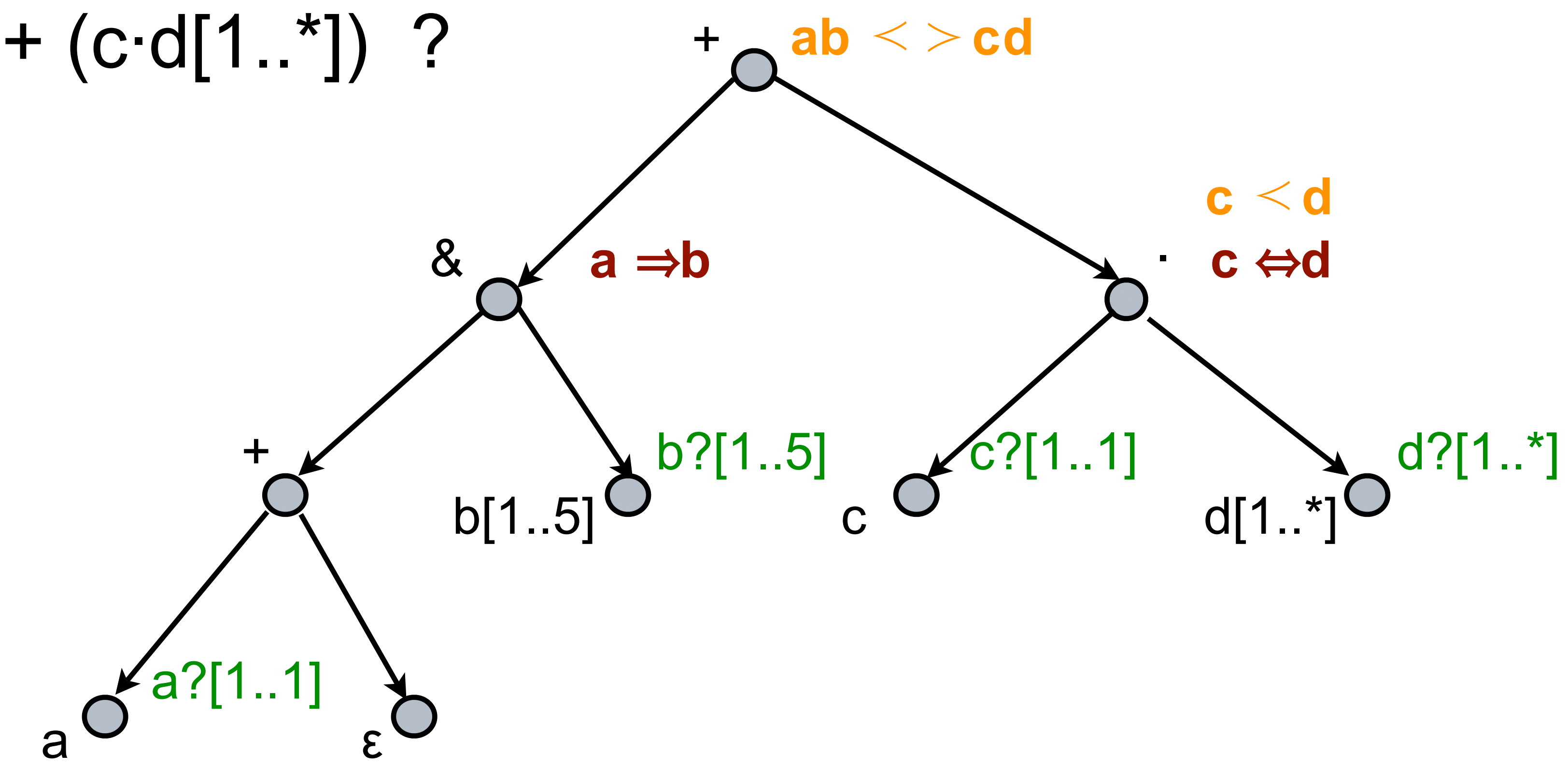
Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A <> B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

Residuation

Input	Constraint	Residual after a_i
$a_i \in A$	$A \Rightarrow B$	B
$a_i \in B$	$A \Rightarrow B$	true
$a_i \in A$	$A \Leftrightarrow B$	B
$a_i \in A$	A	true
$a_i \in A$	$A < > B$	$\neg B$
$a_i \in B$	$A < B$	$\neg A$
$a_i \in A$	$\neg B$	$\neg B$
$a_i \in A$	$\neg A$	false

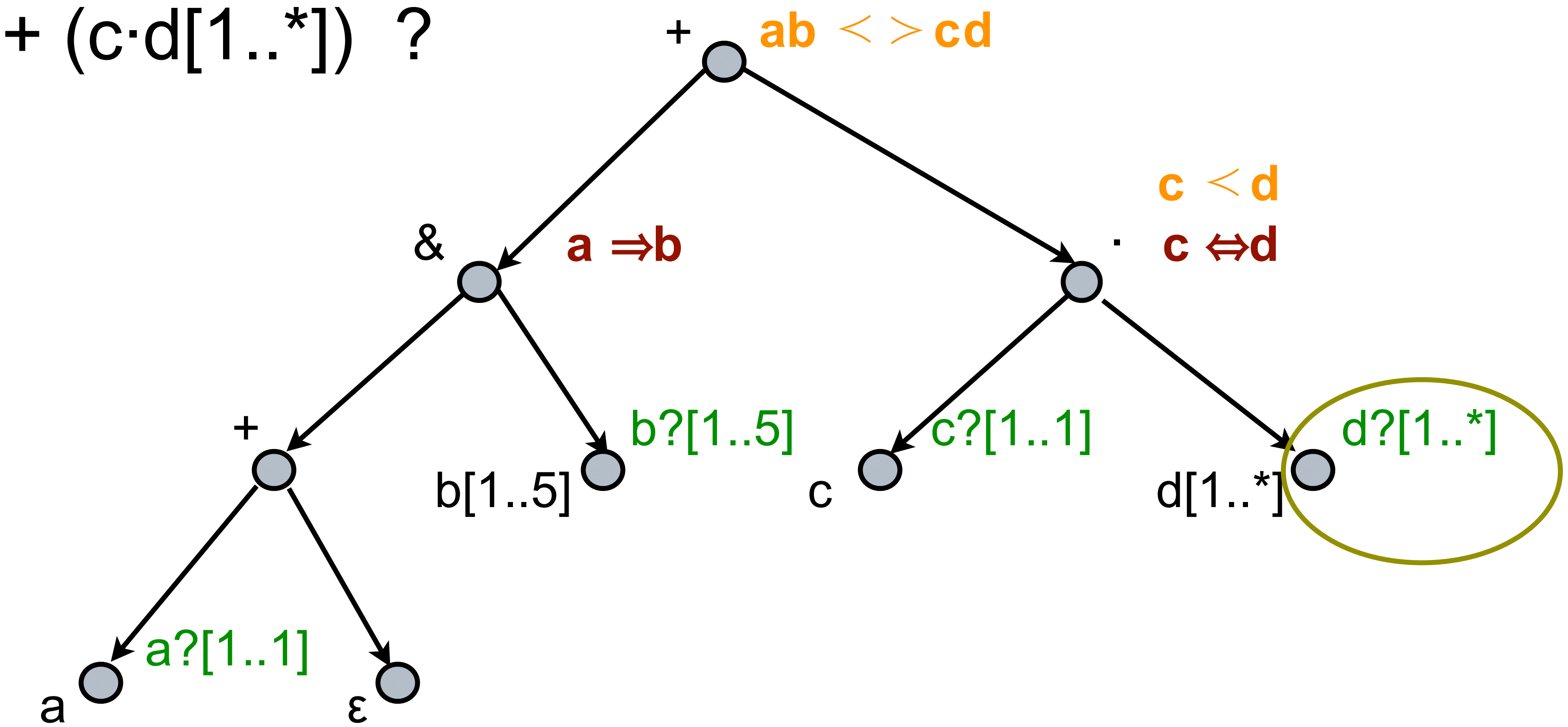
Linear residuation

$d \in ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$?



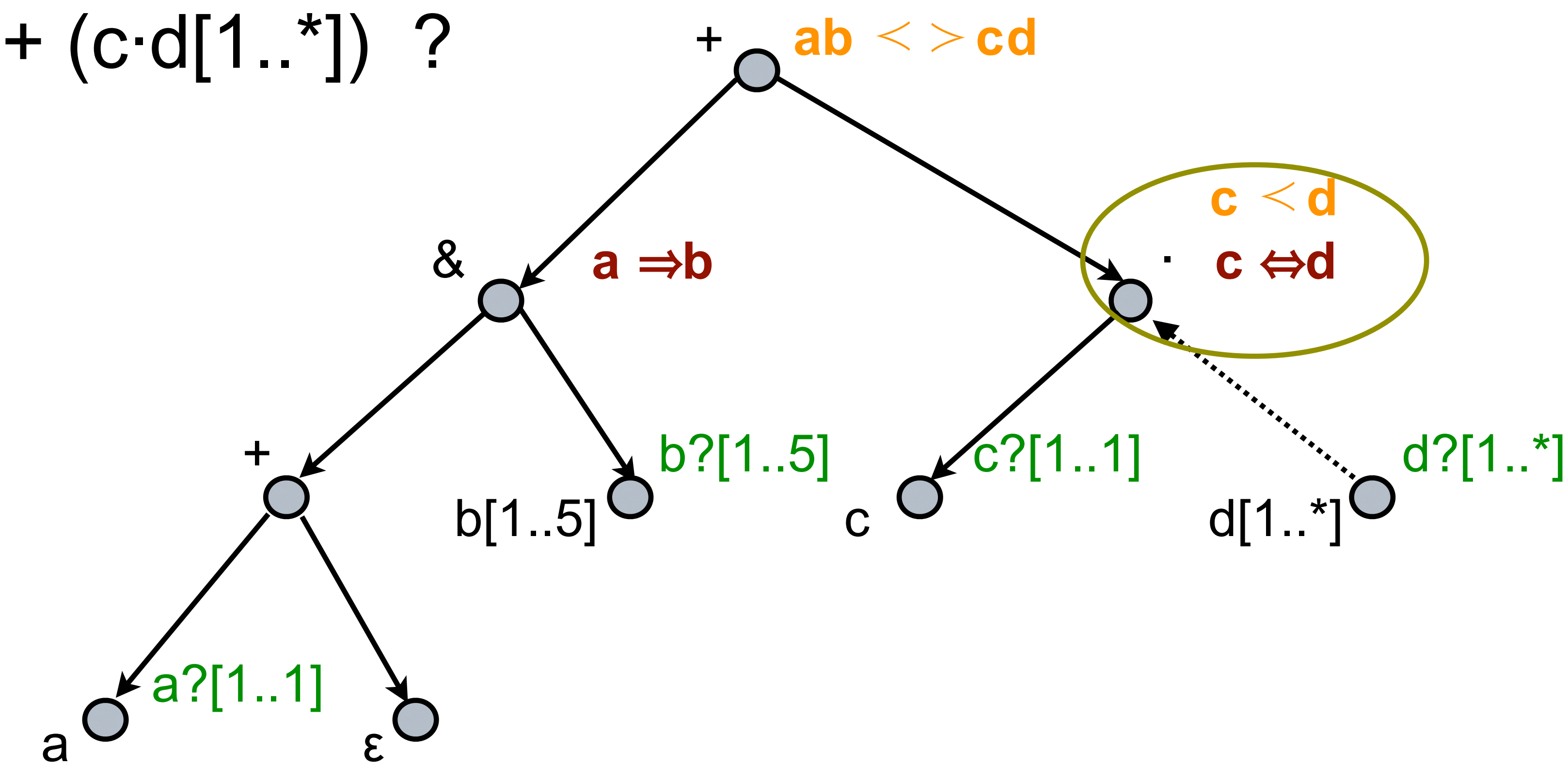
Linear residuation

$d \in ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$?



Linear residuation

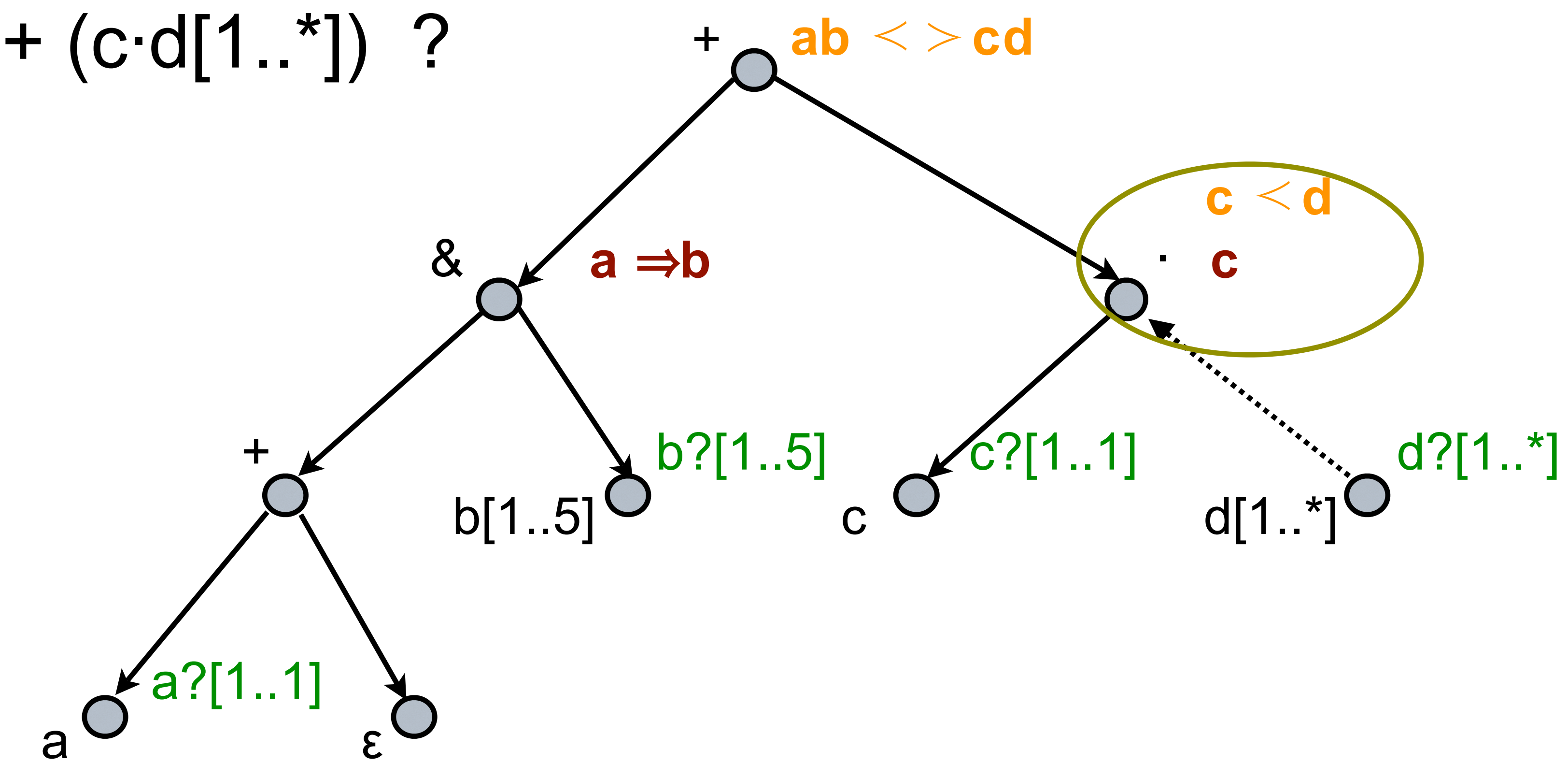
d ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



a_i∈B	A⇔B	A

Linear residuation

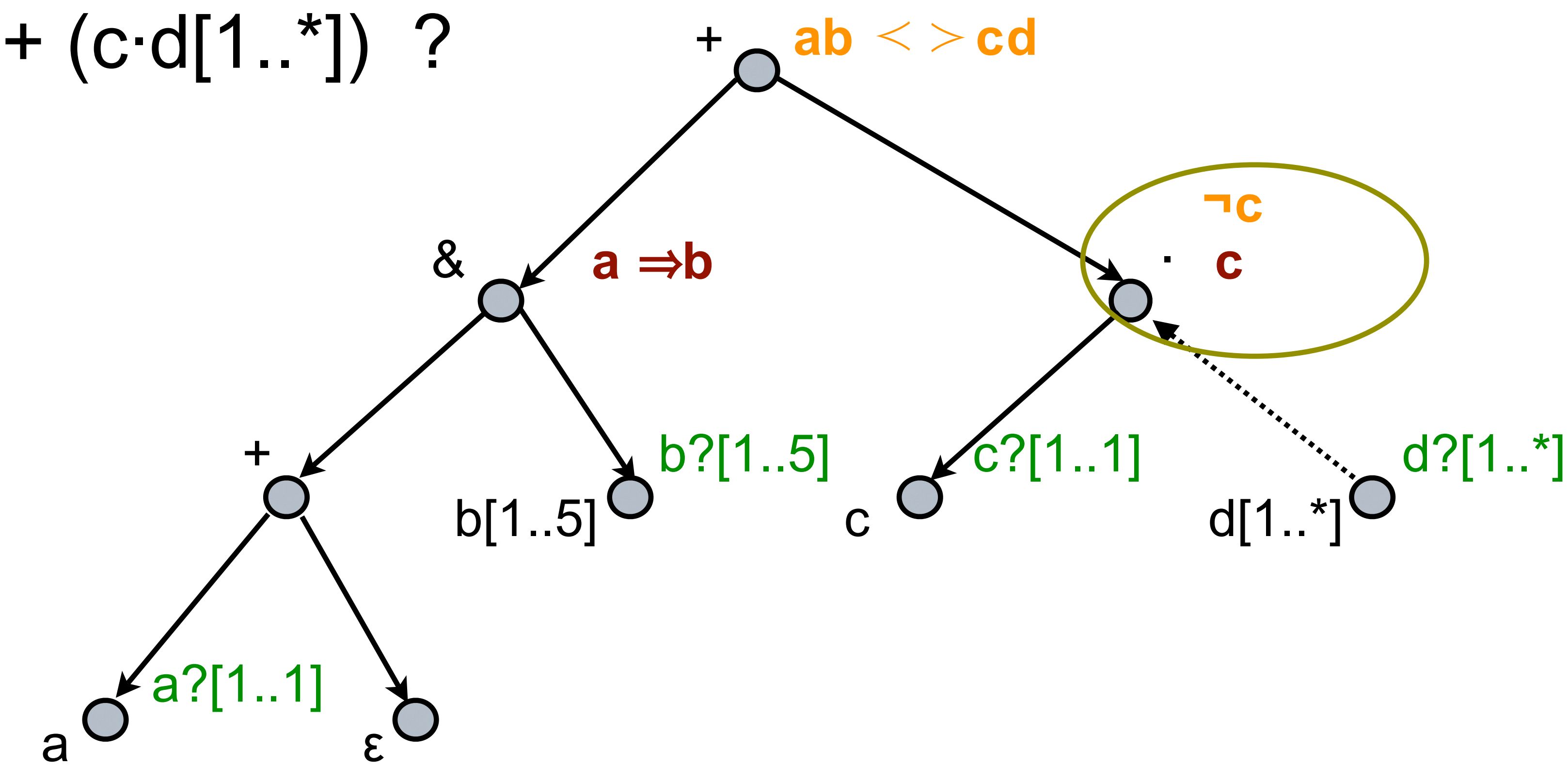
$d \in ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$?



$a_i \in B$	$A \Leftrightarrow B$	A

Linear residuation

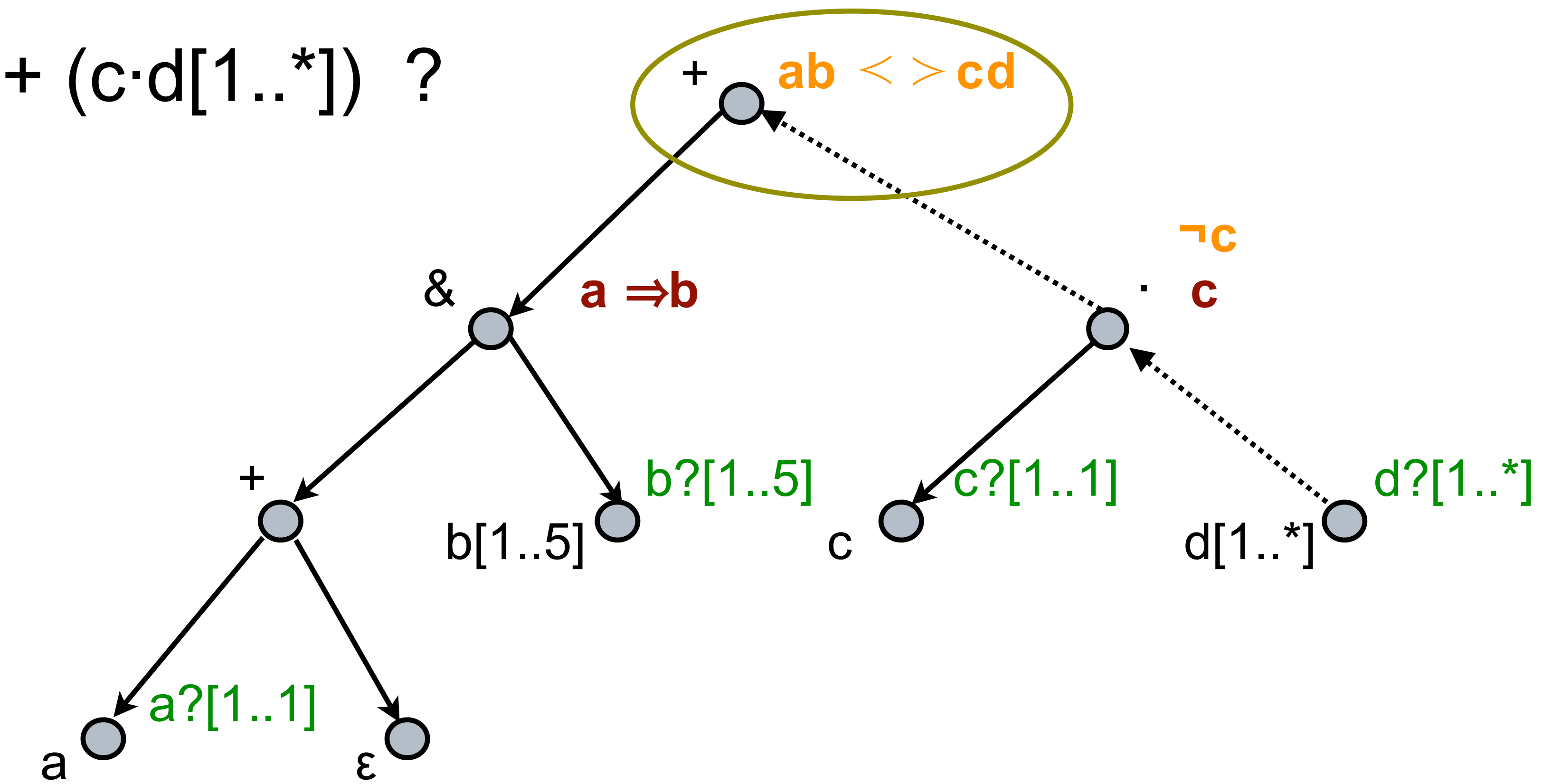
$d \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



$a_i \in A$	$A < B$	$\neg B$

Linear residuation

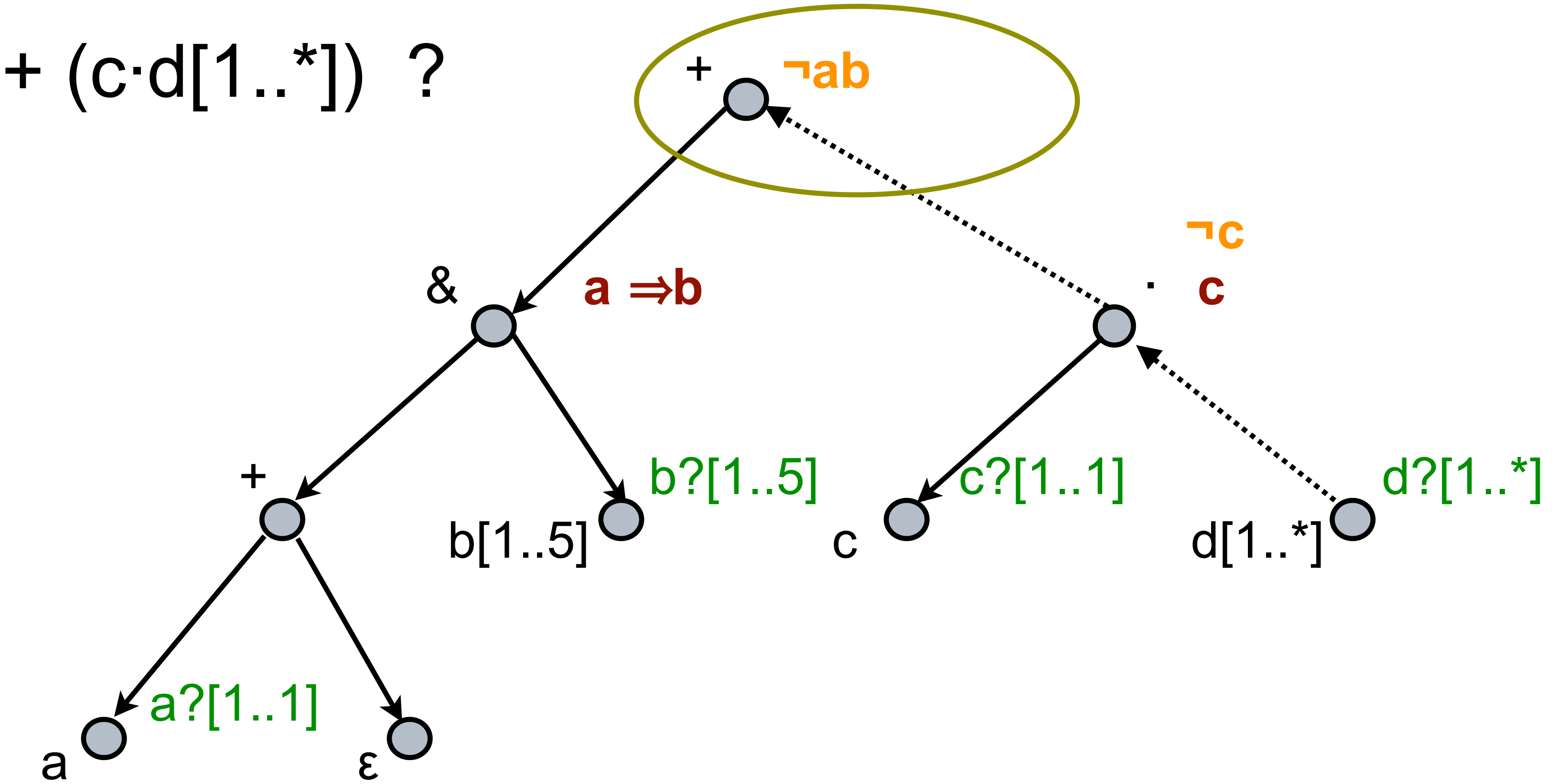
$d \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



$a_i \in A$	A < > B	¬B

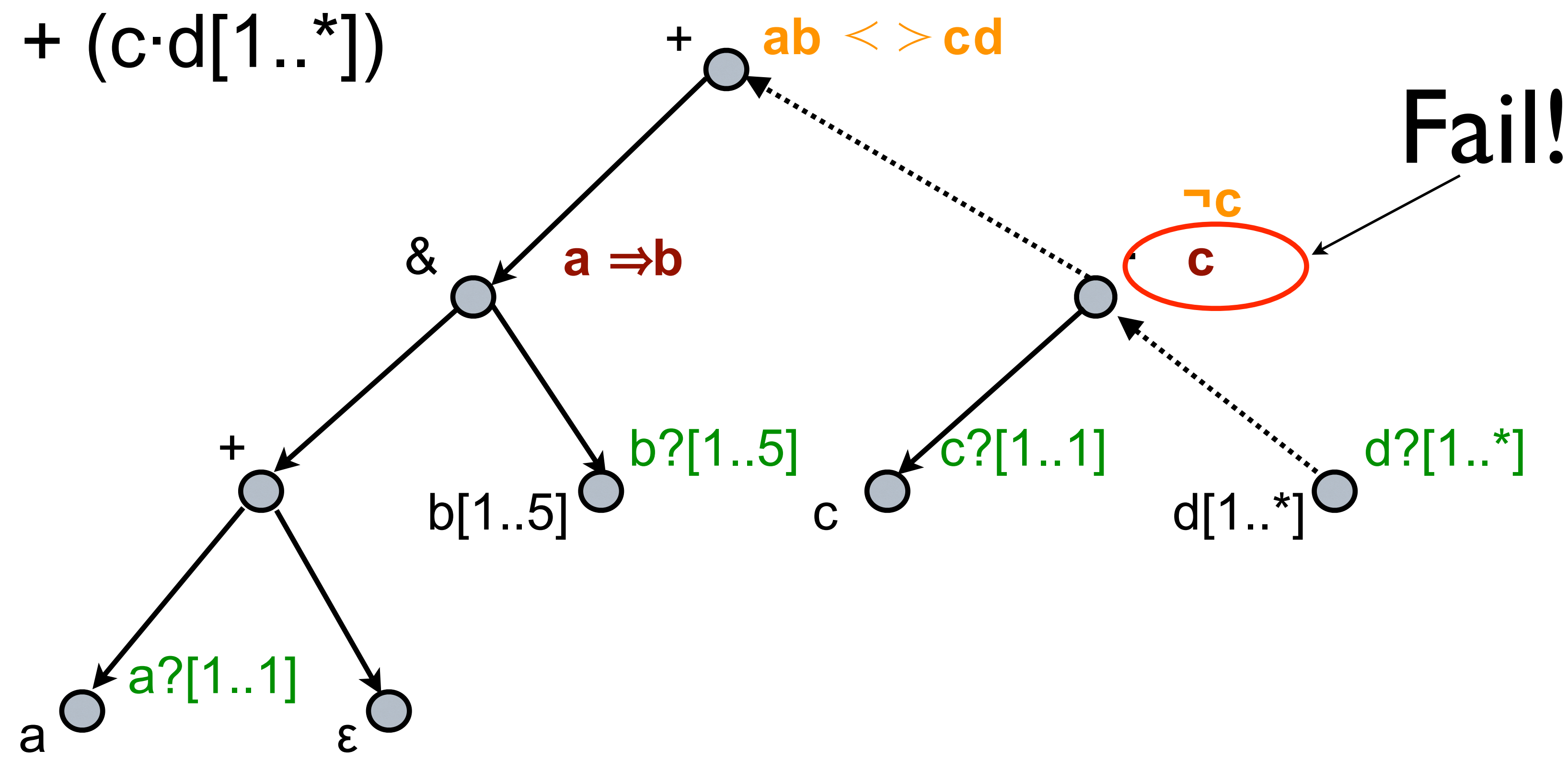
Linear residuation

$d \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



Linear residuation

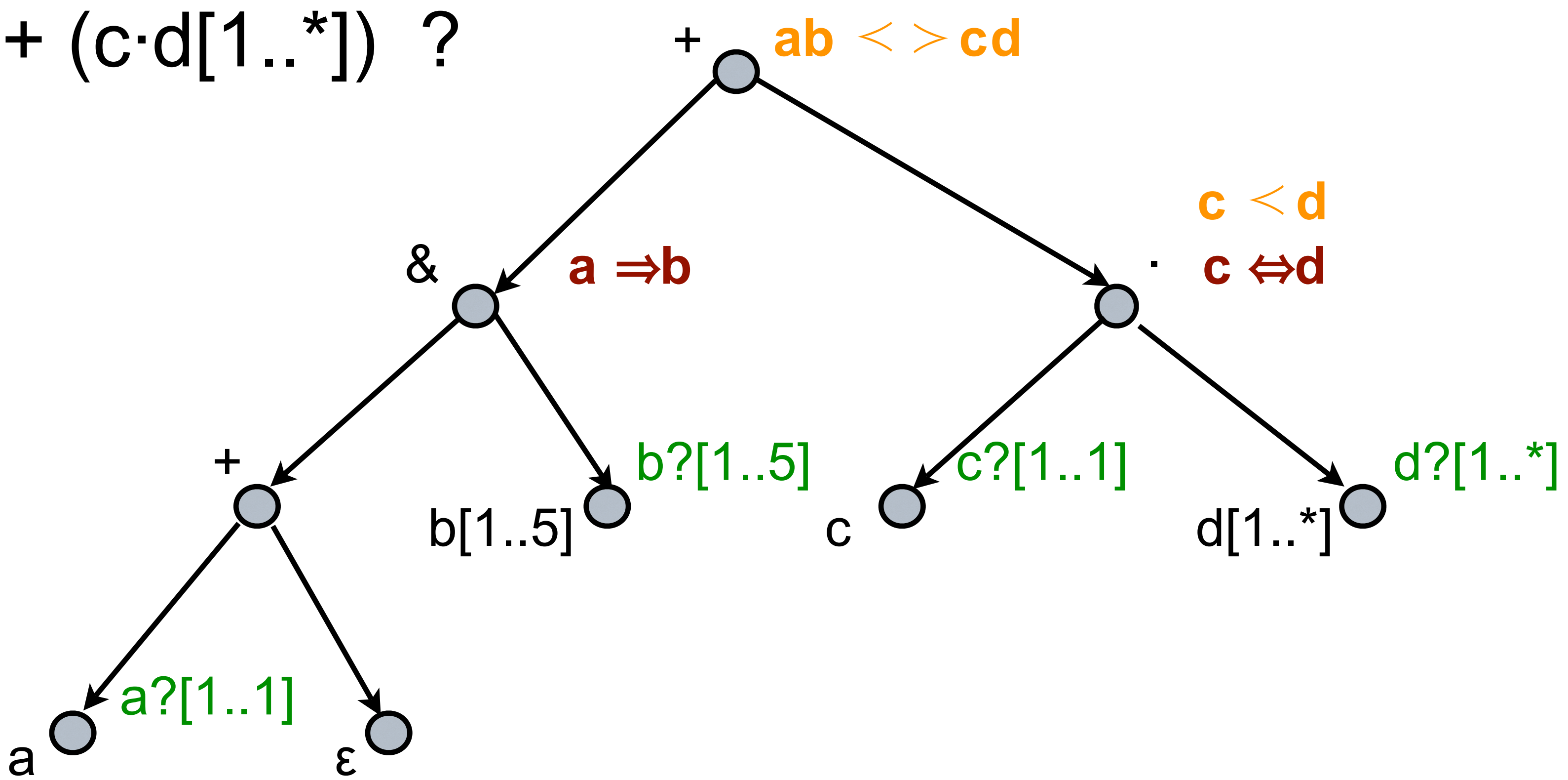
$d \notin ((a+\varepsilon)\&b[1..5]) + (c \cdot d[1..*])$



Failure : the final residual contains a formula A=c

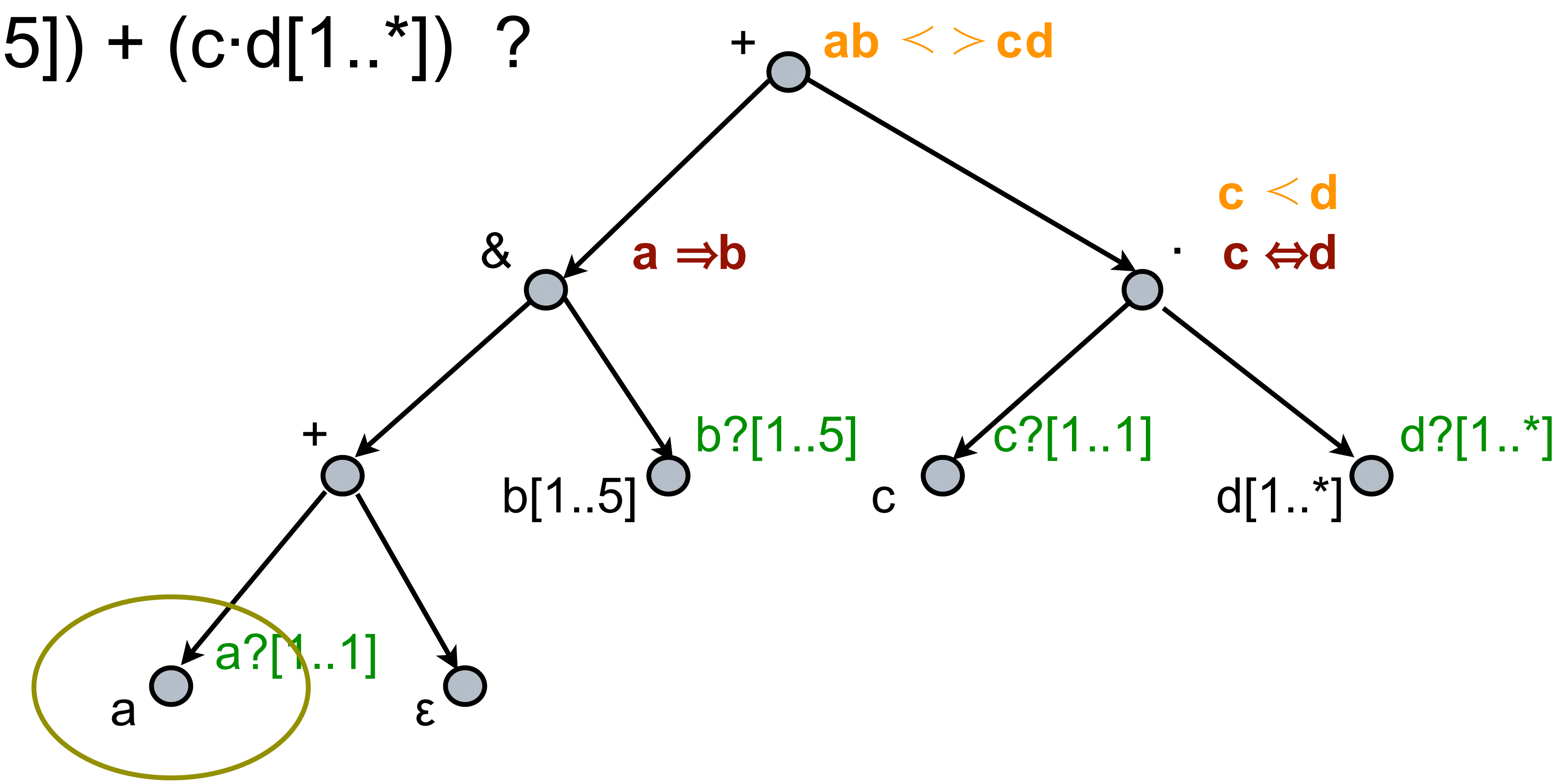
Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



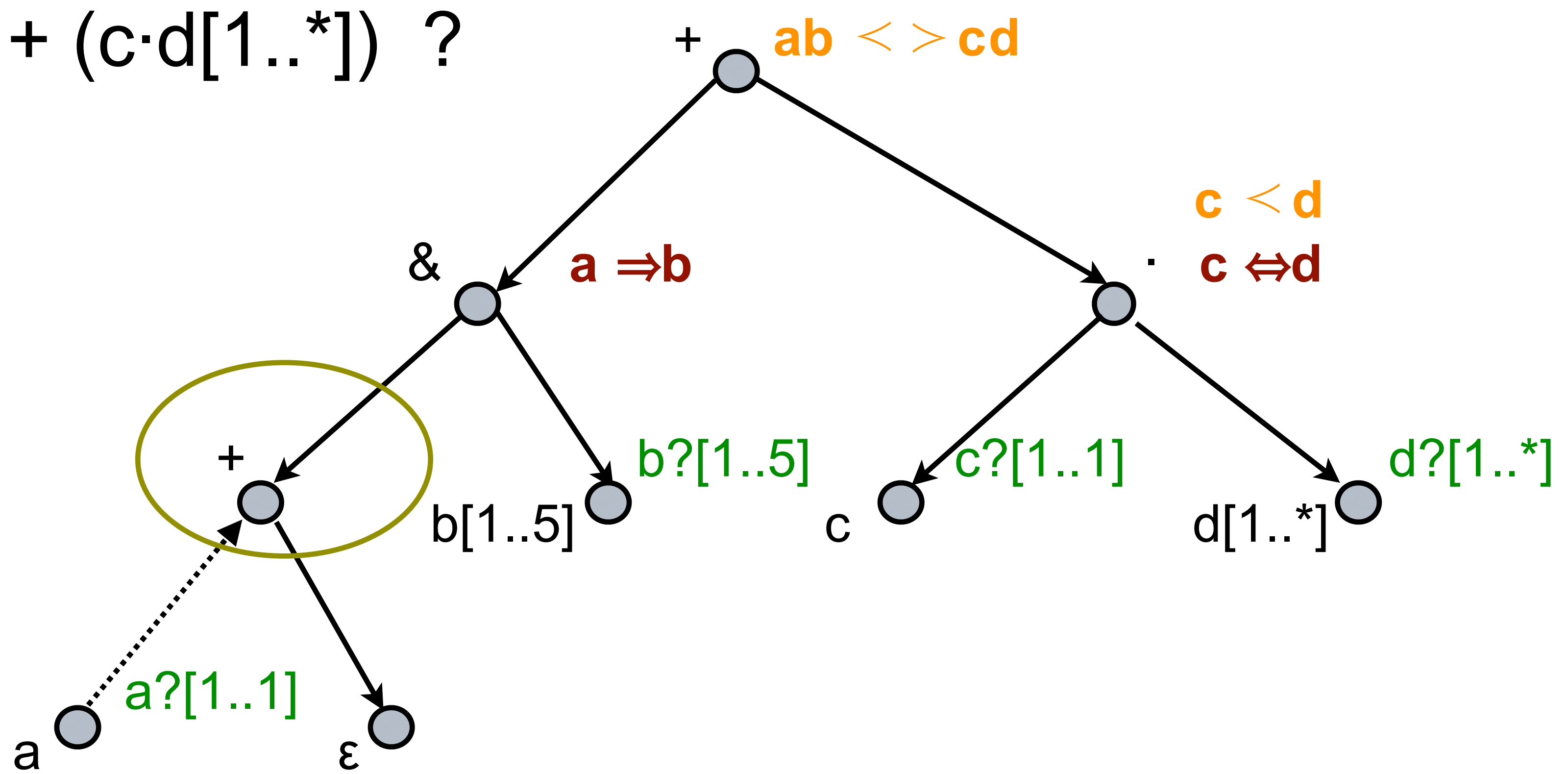
Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



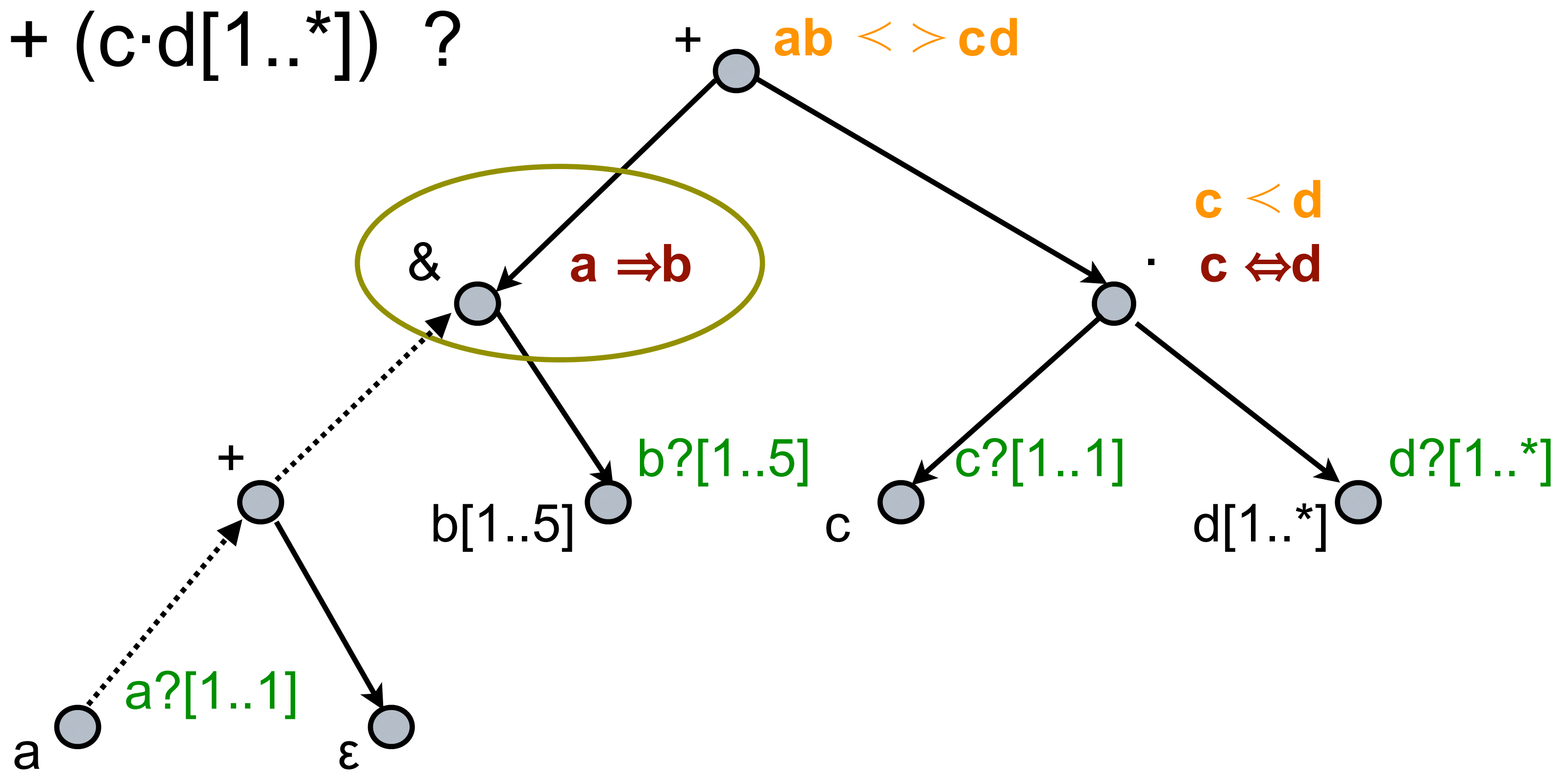
Linear residuation

$\boxed{a}bb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



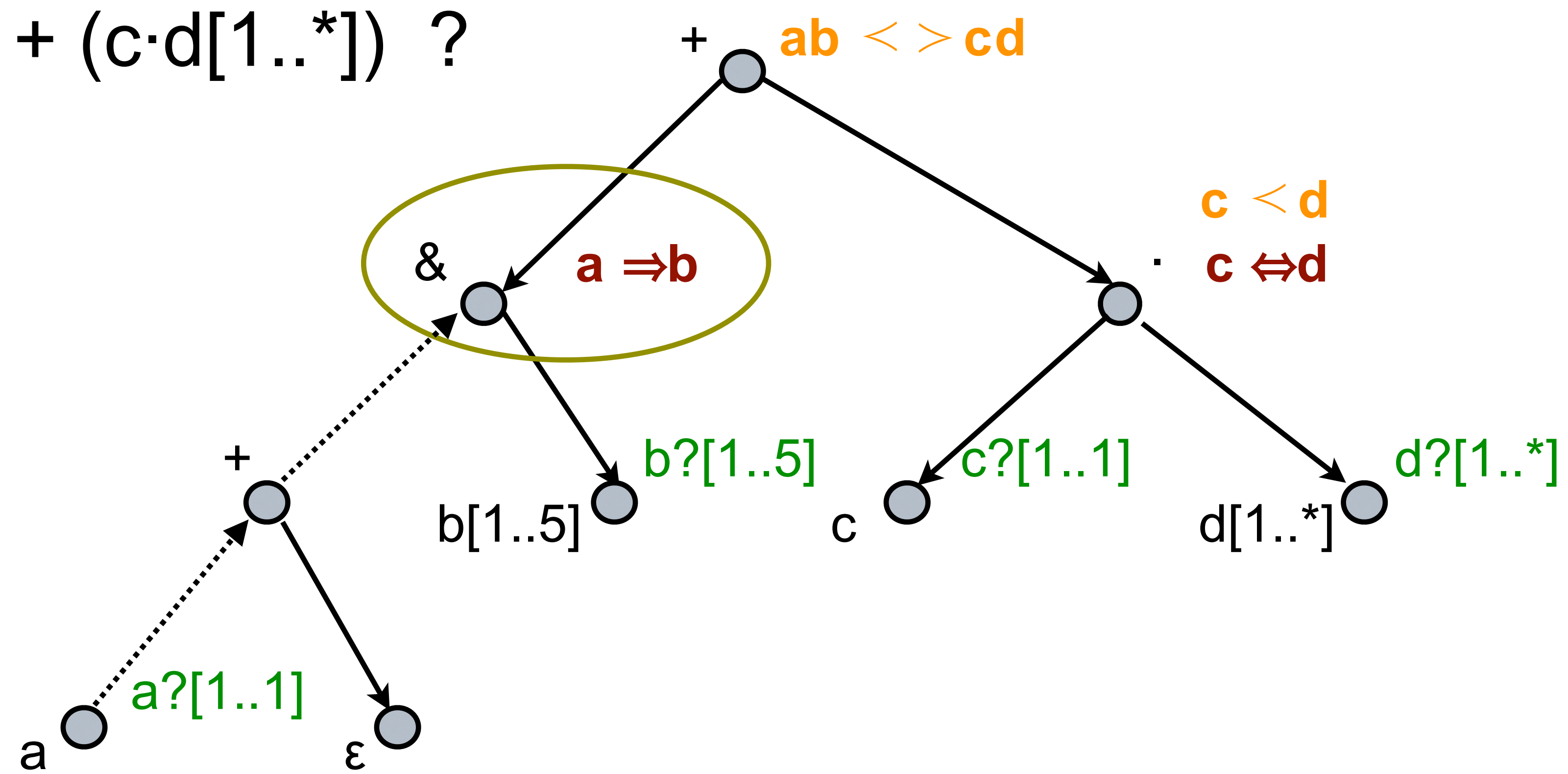
Linear residuation

$\boxed{a}bb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



Linear residuation

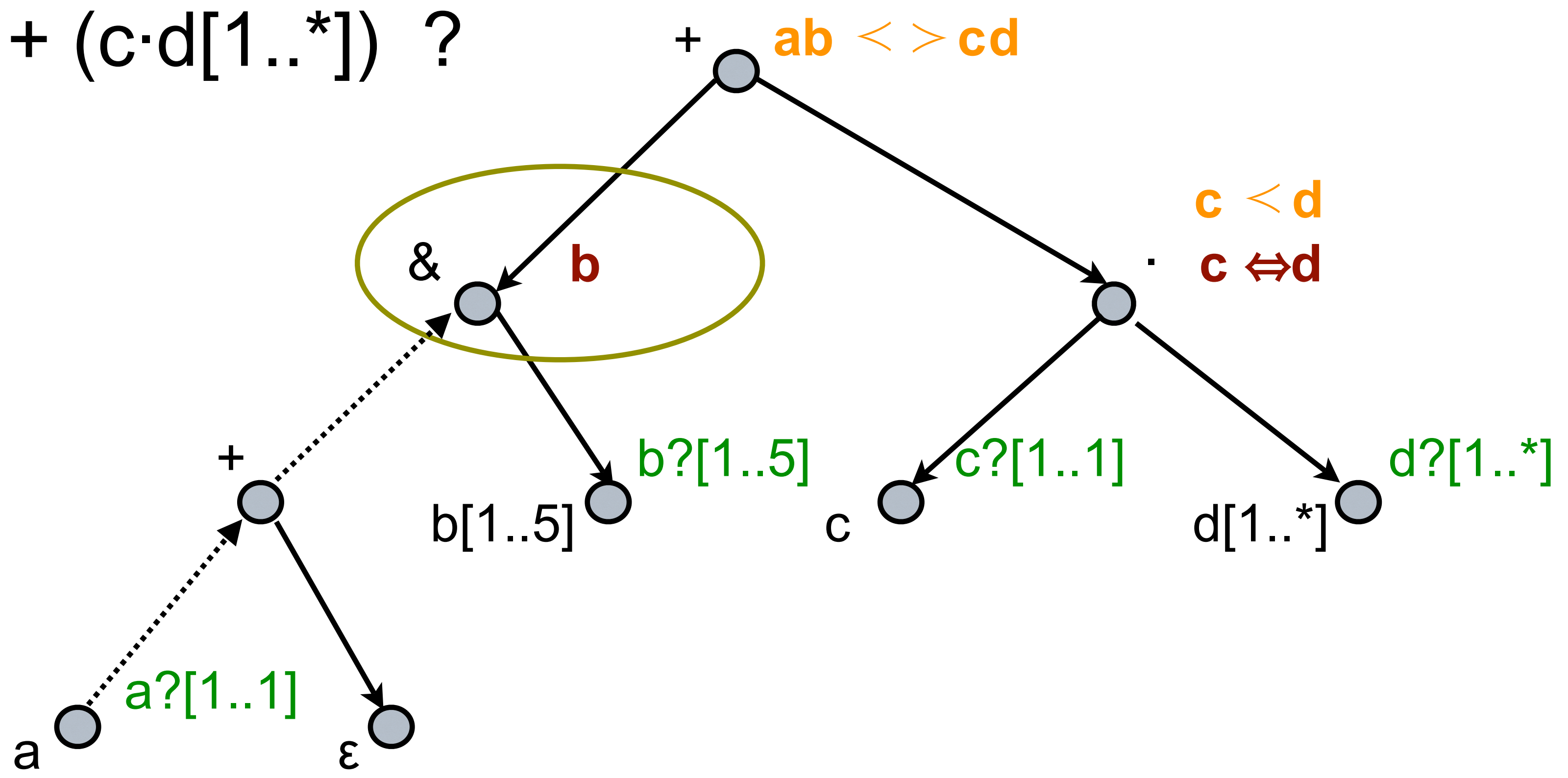
$\boxed{a}bb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



$a_i \in A$	$A \Rightarrow B$	B

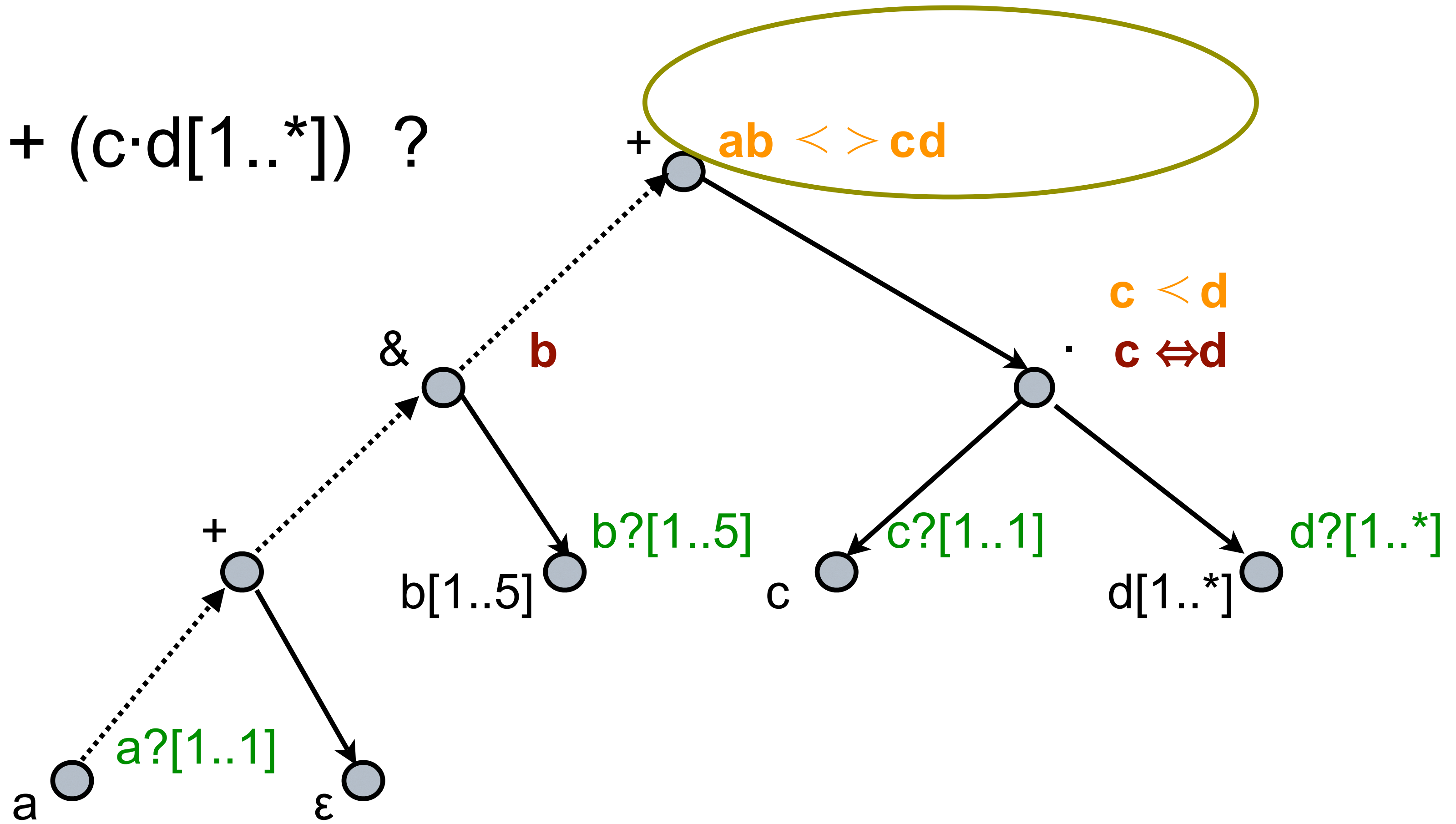
Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



Linear residuation

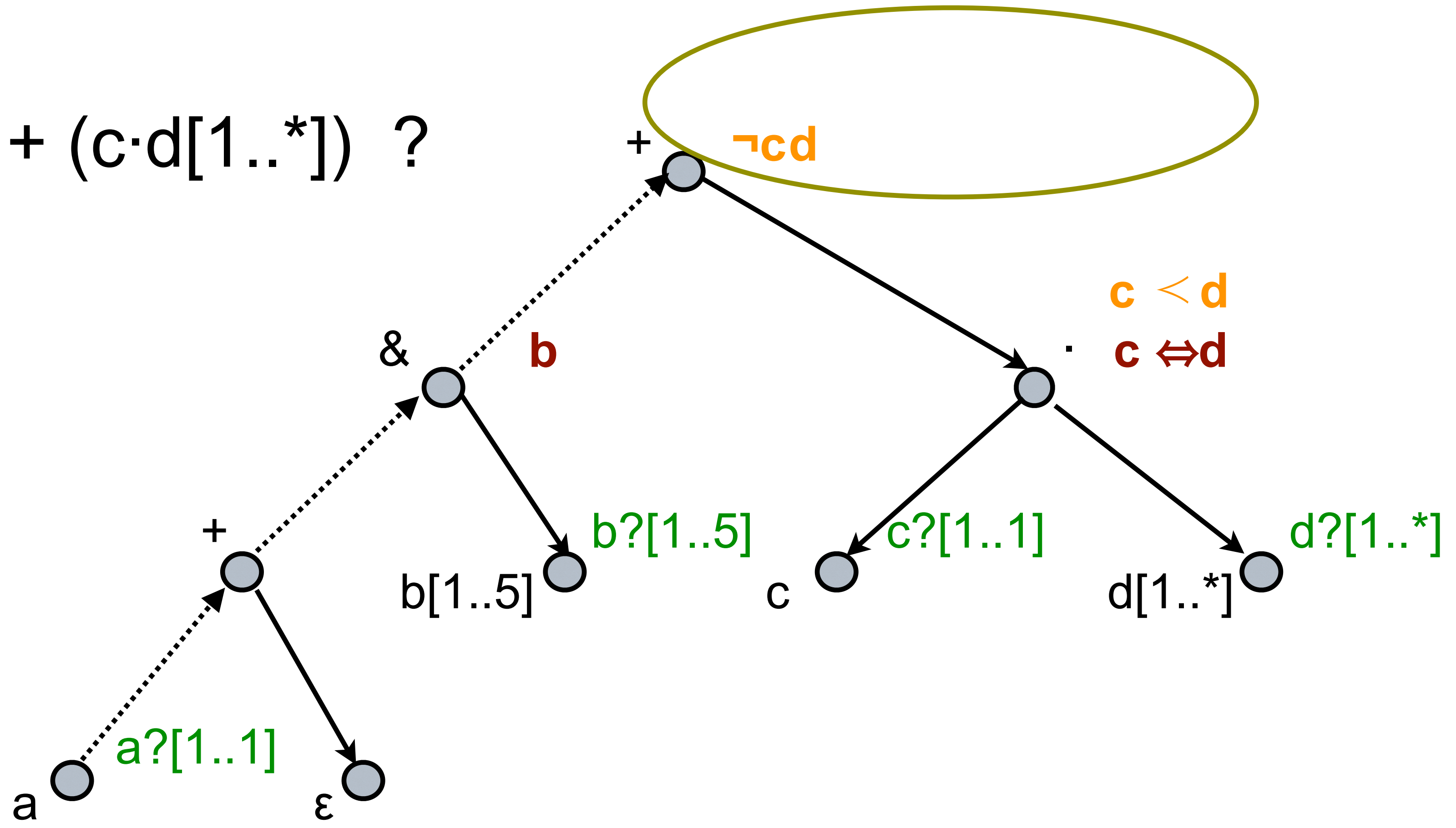
a̲bb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



a_i∈A	A < > B	¬B

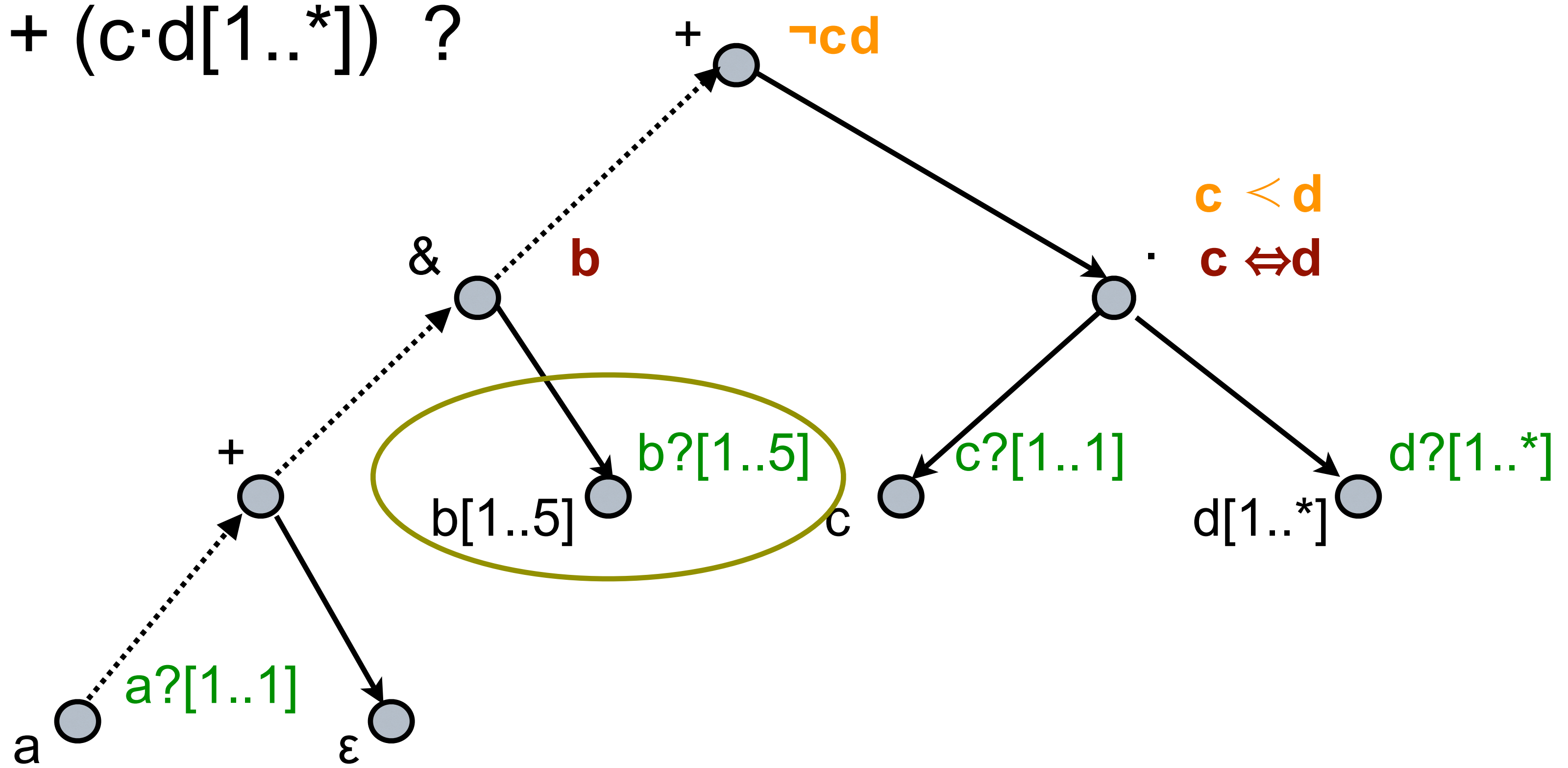
Linear residuation

$\boxed{a}bb \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



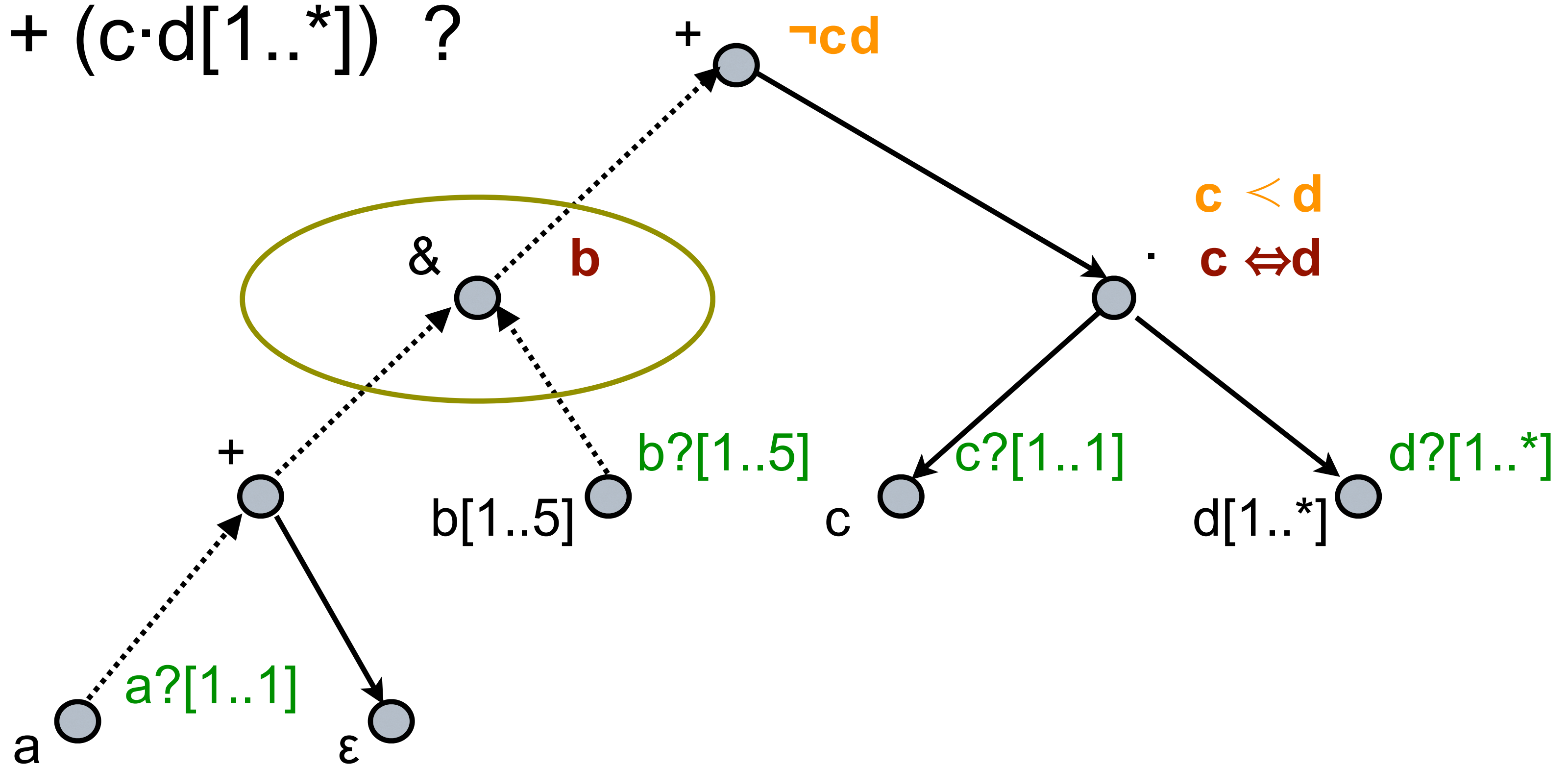
Linear residuation

a$\boxed{\text{b}}$b ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



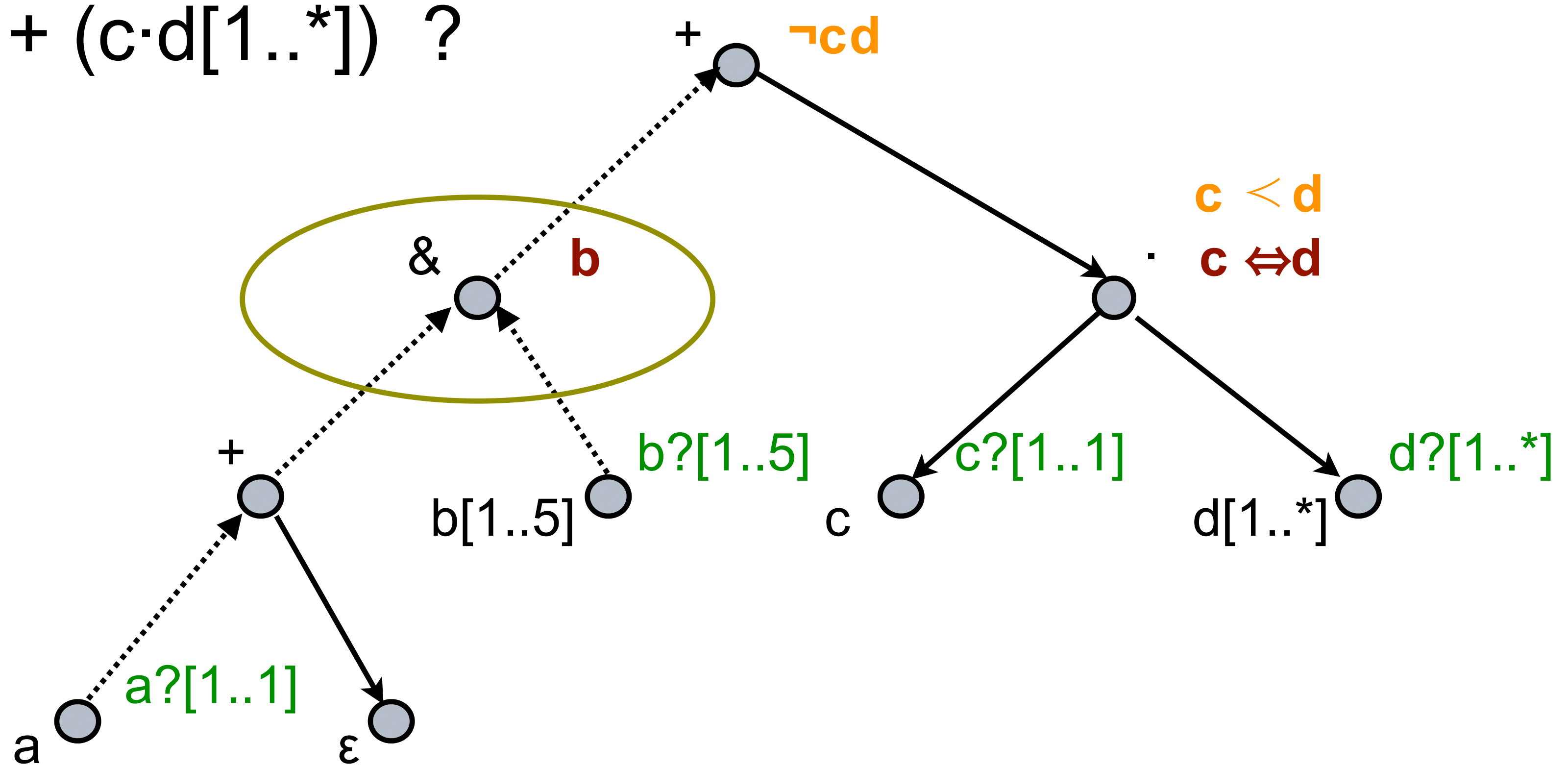
Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



Linear residuation

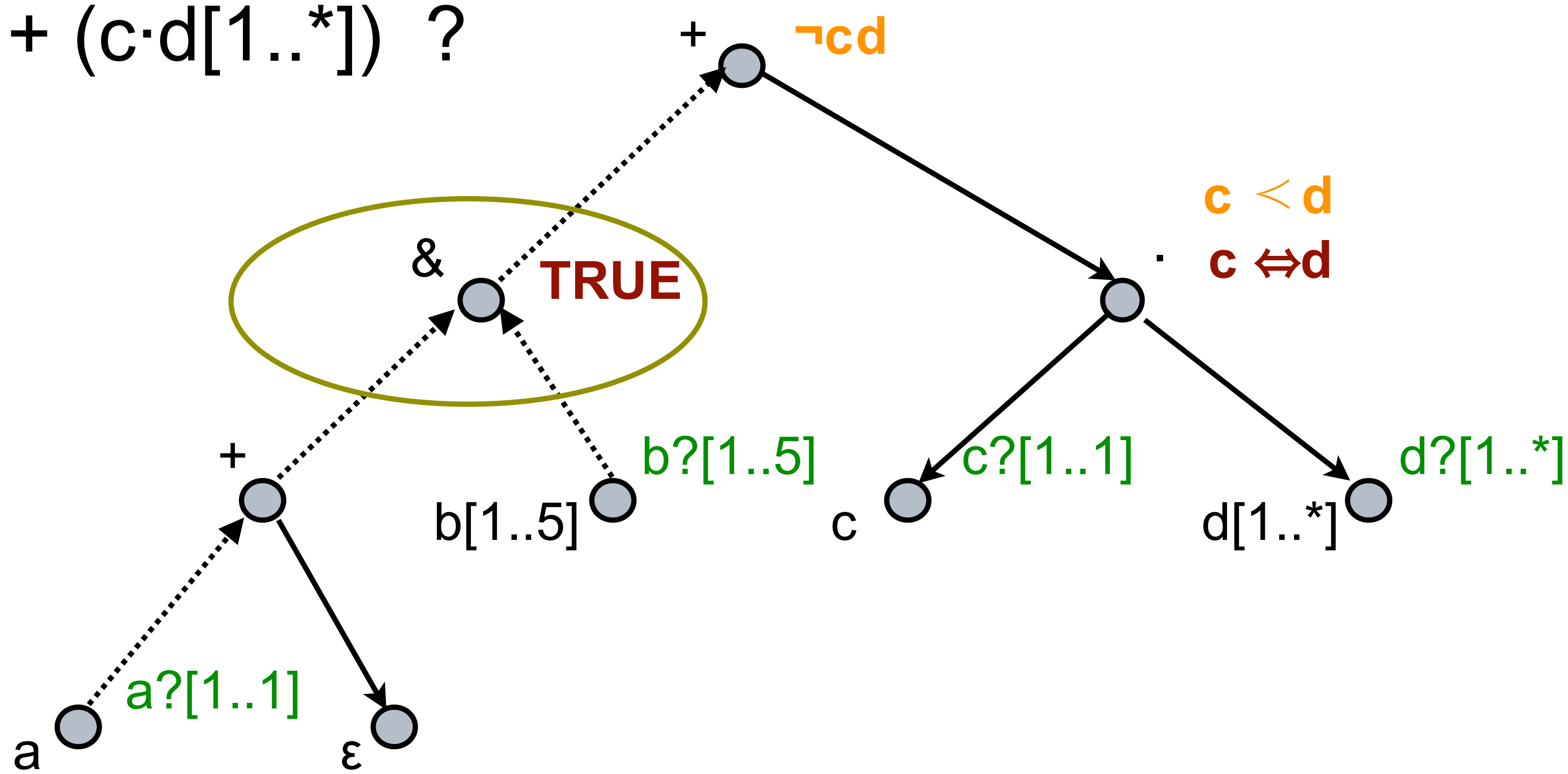
$\boxed{ab}b \in ((a+\varepsilon)\&b[1..5]) + (c\cdot d[1..*])$?



$a_i \in A$	A	true

Linear residuation

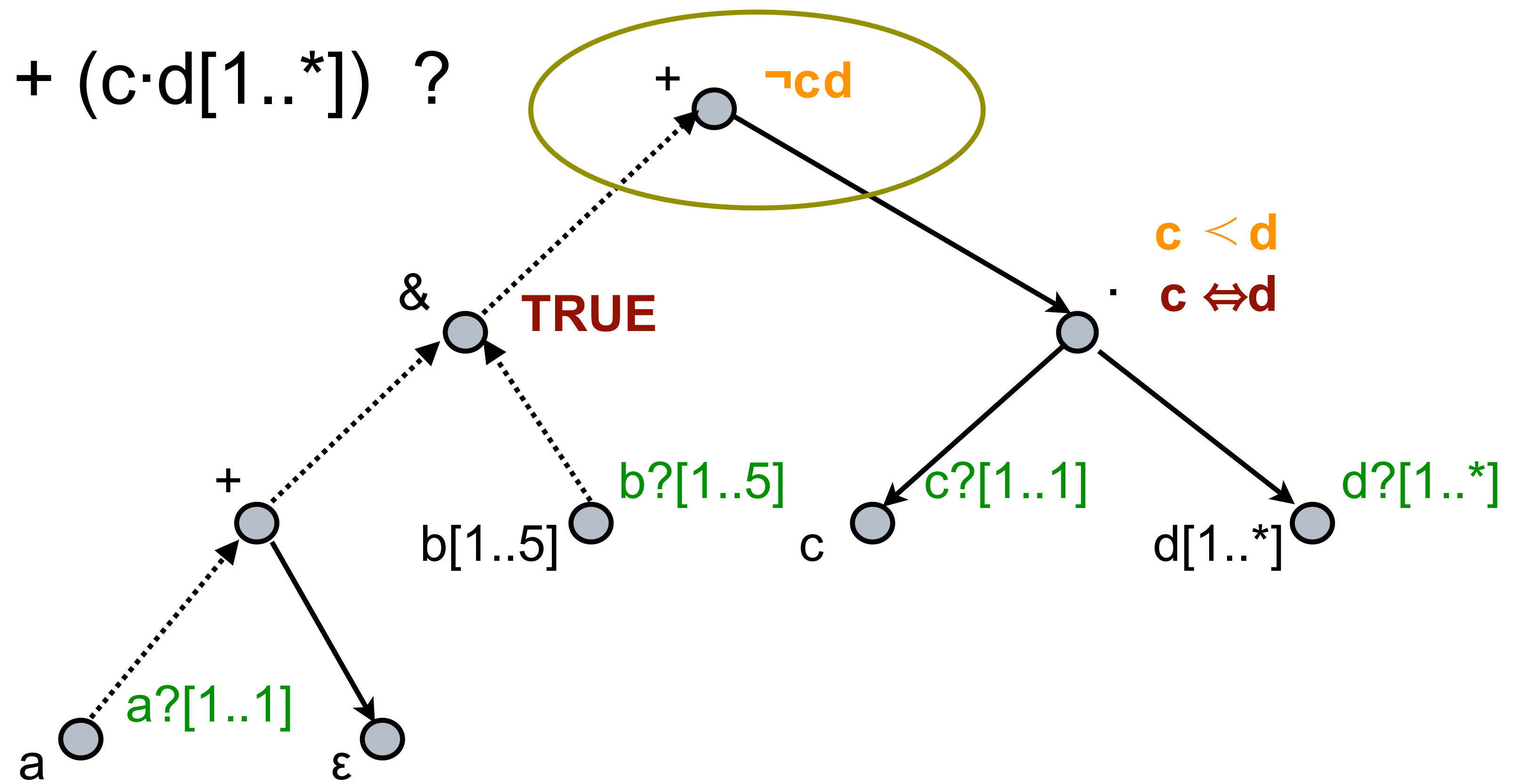
abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



$a_i \in A$	A	**true**

Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?

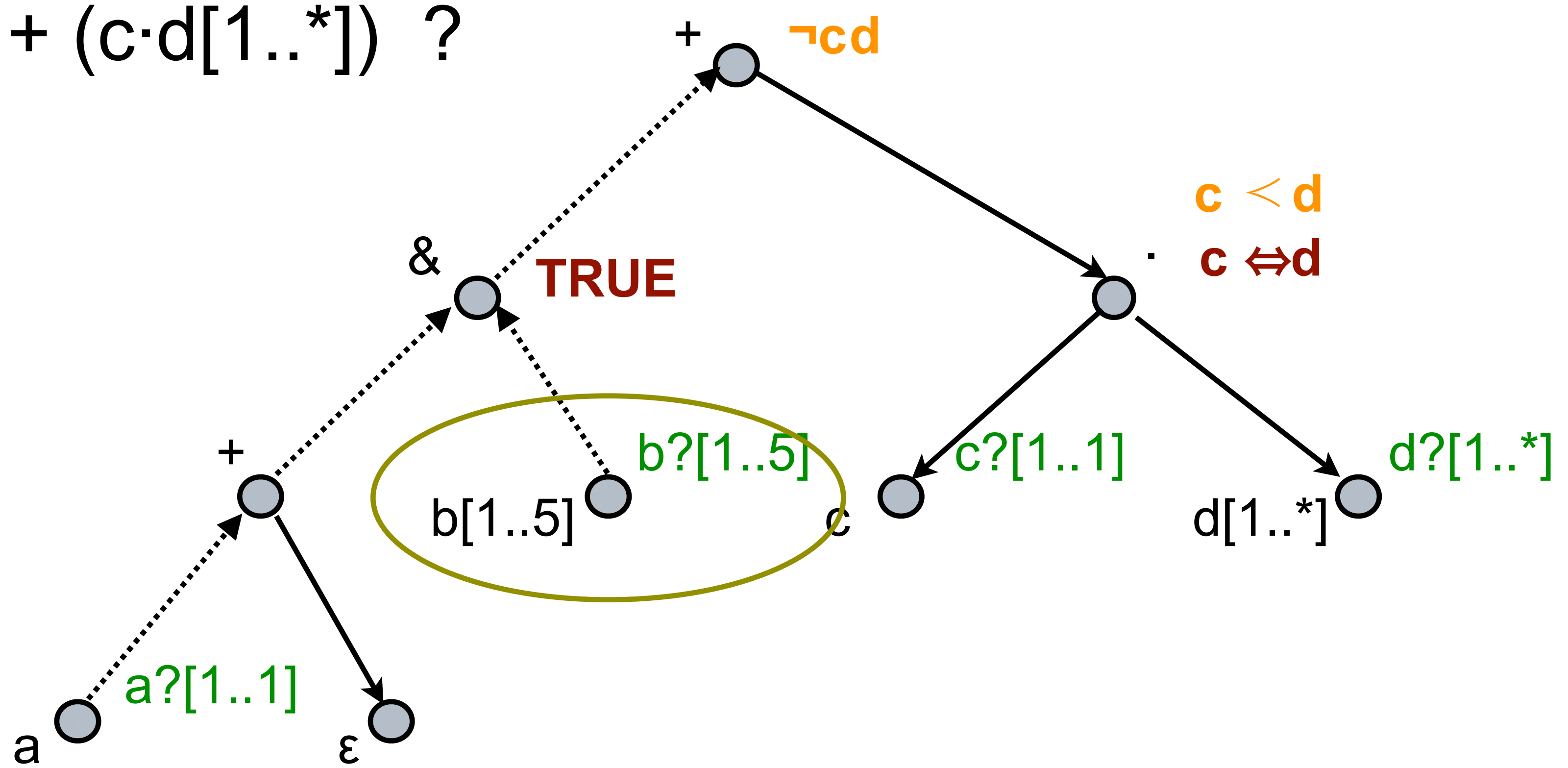


$a_i \in A$	¬B	**¬B**

Linear residuation

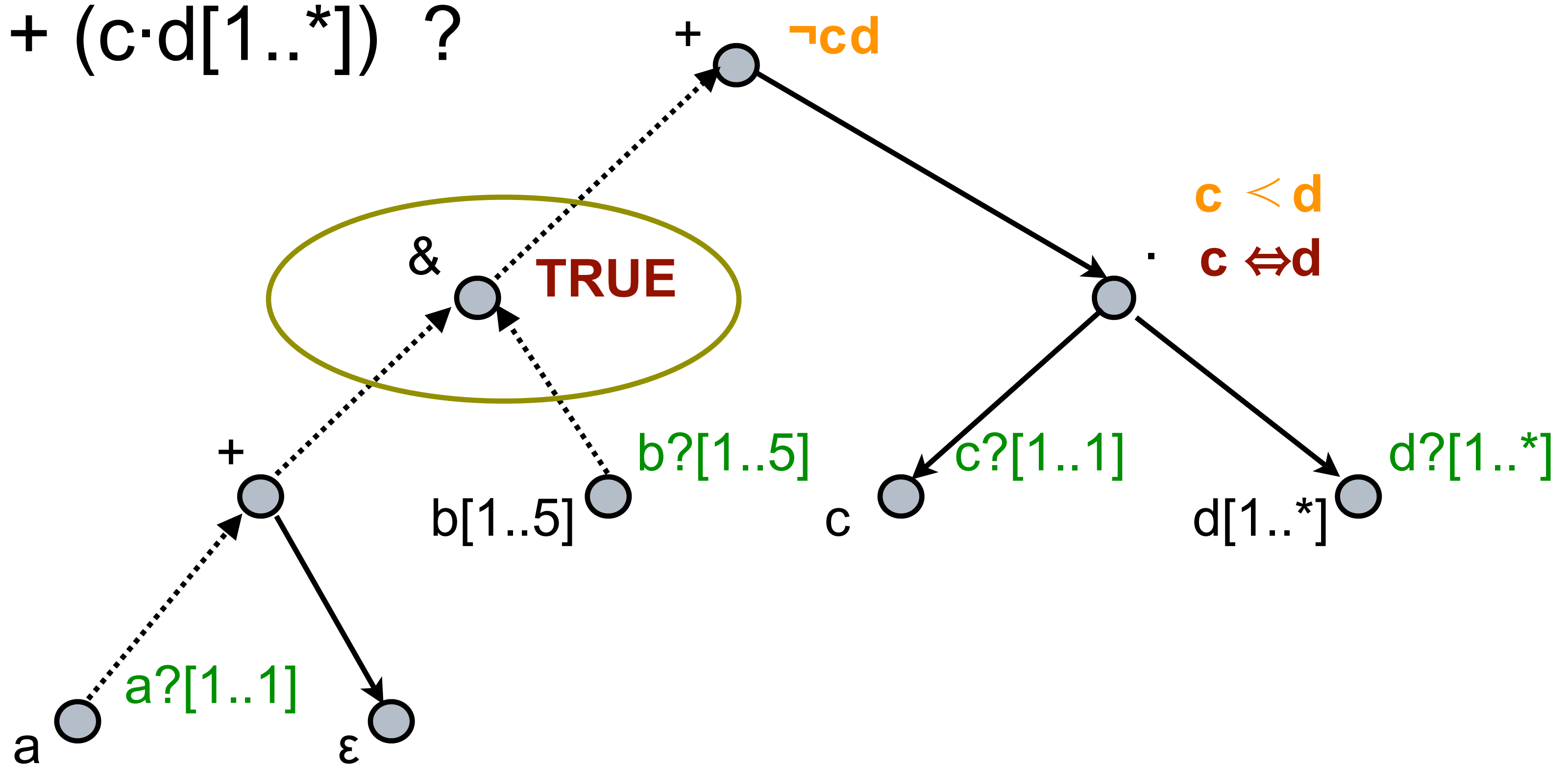
ab$\boxed{b}$ ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?

Nothing new!



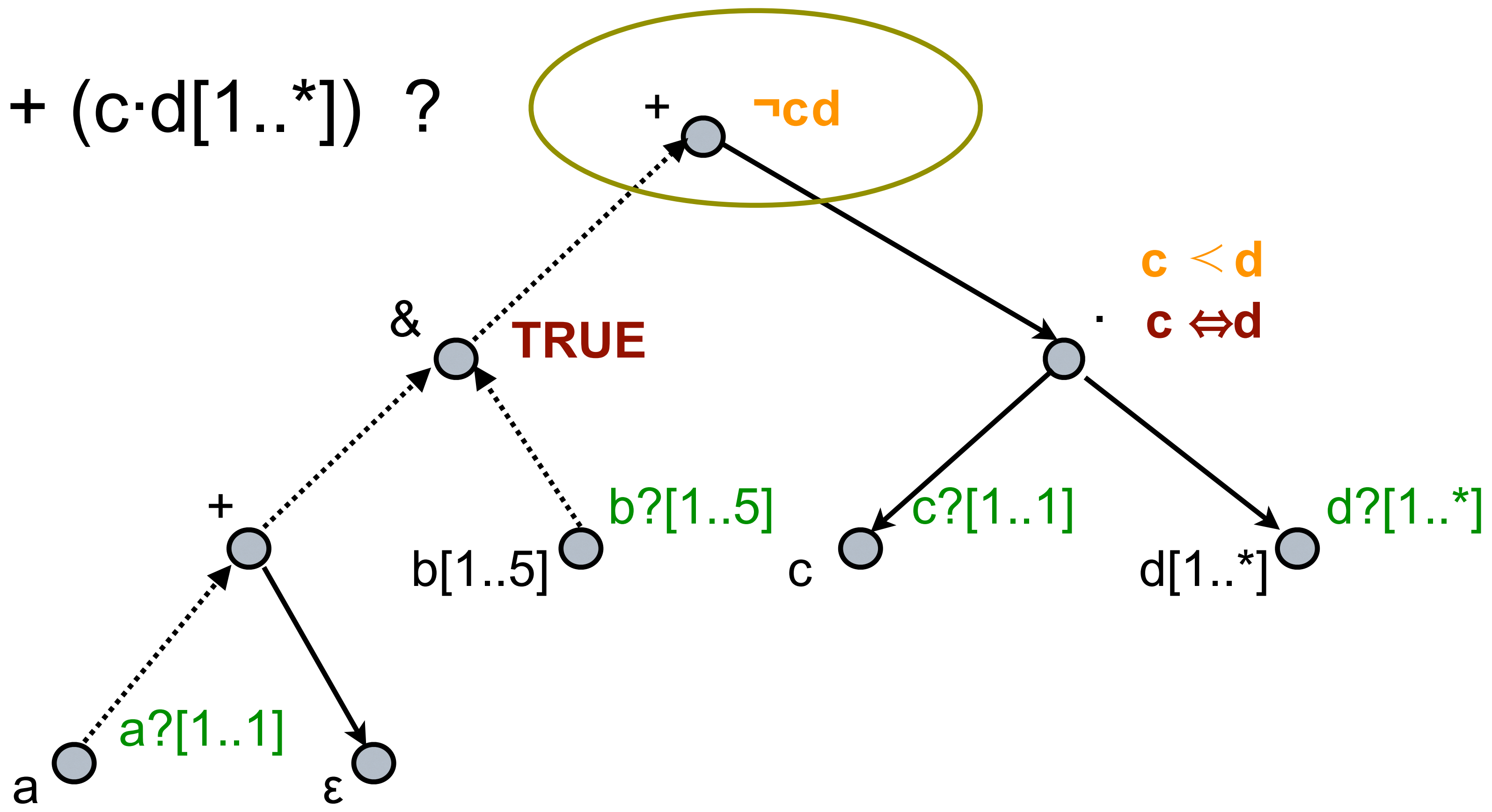
Linear residuation

ab$\boxed{b}$ $\in$ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



Linear residuation

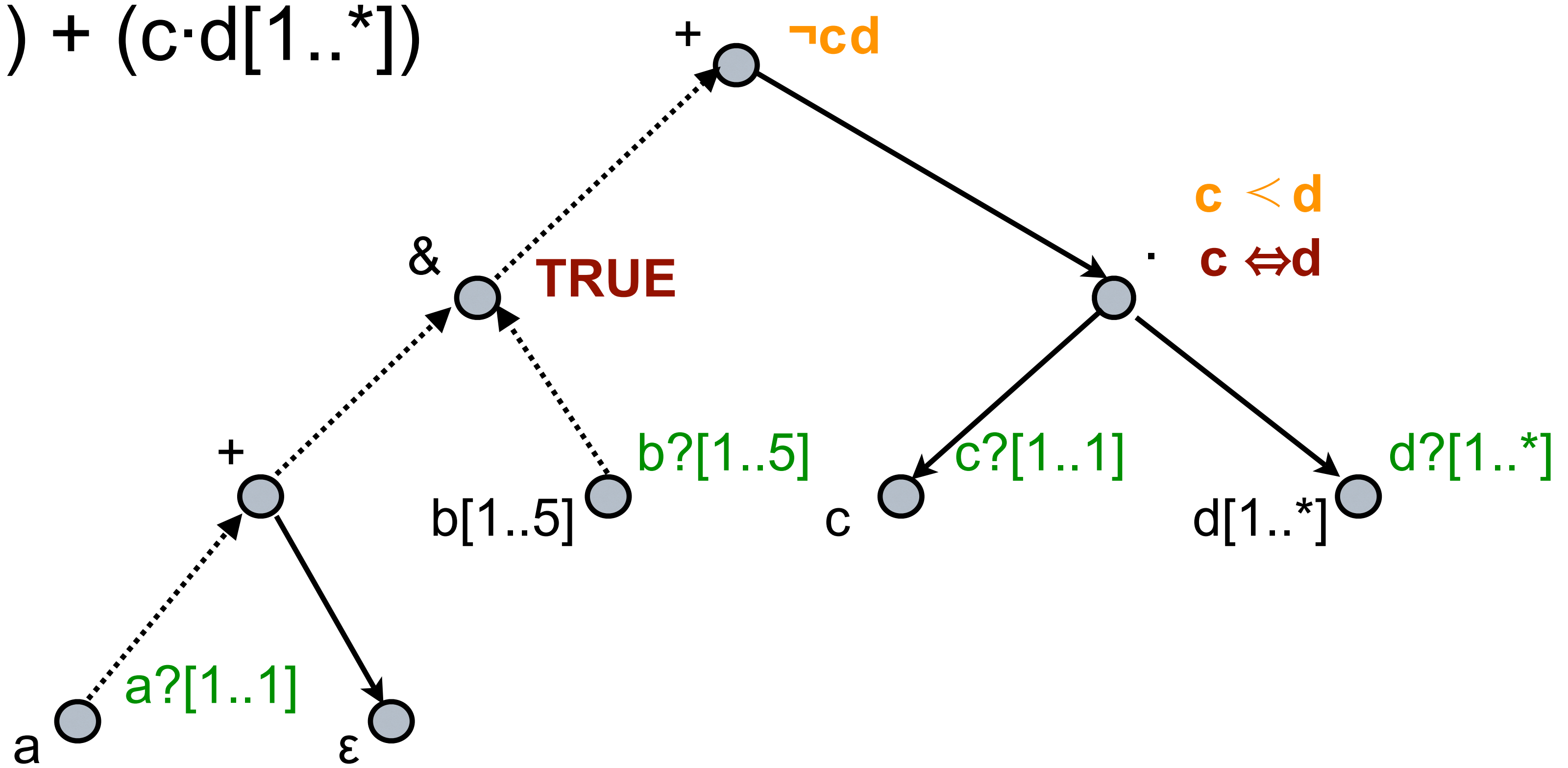
a$\boxed{b}$b $\in$ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



$a_i \in A$	¬B	¬B

Linear residuation

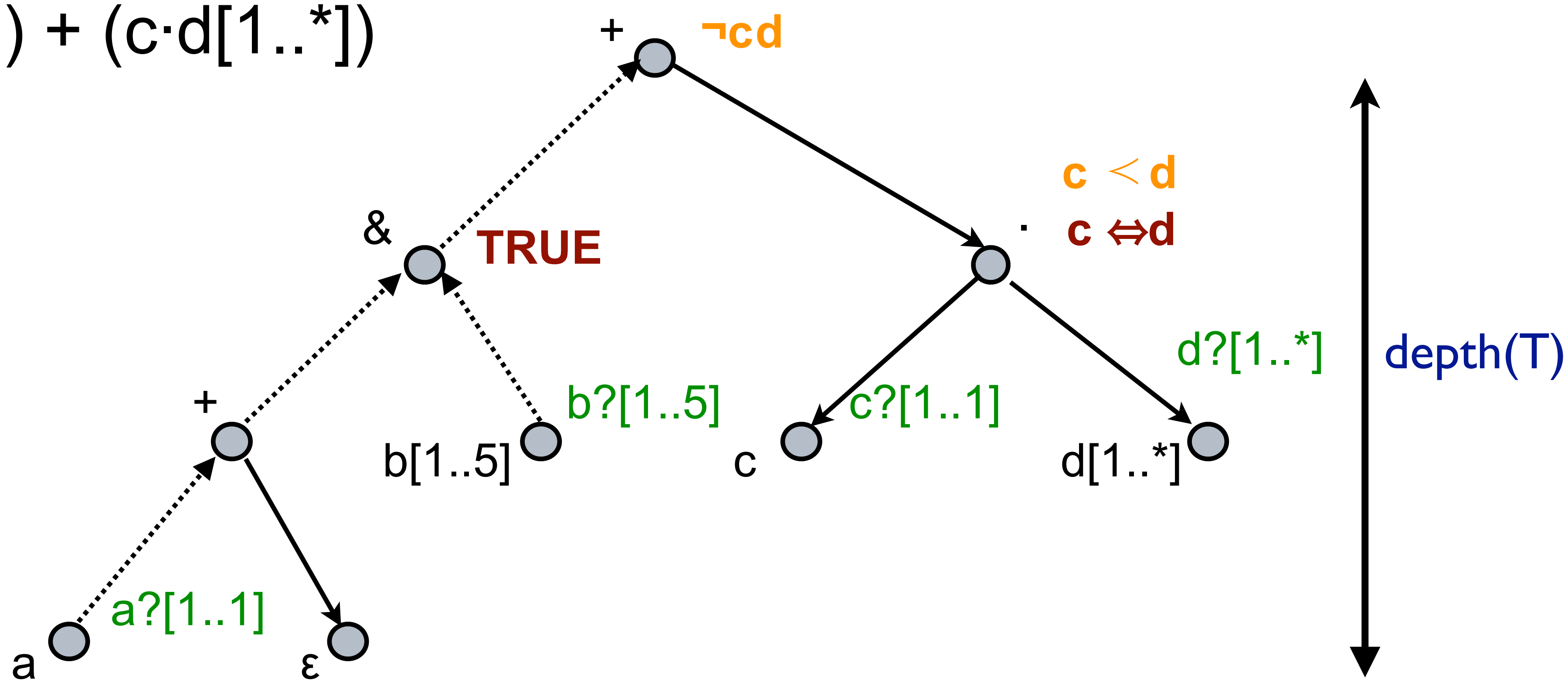
abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*])



No A or False in the final residual ⇒Success!

Linear residuation

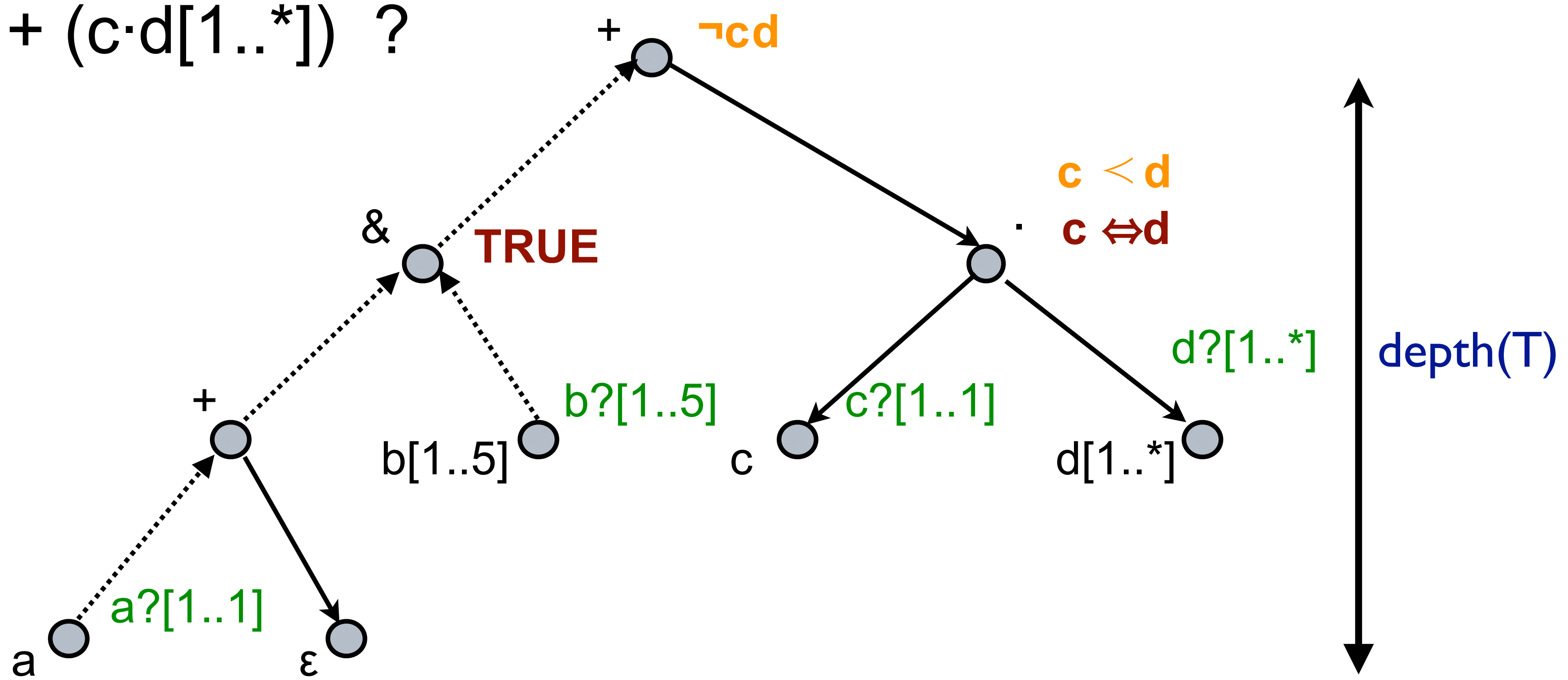
abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*])



Complexity : O(|T|+|w|·depth(T))

Linear residuation

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?

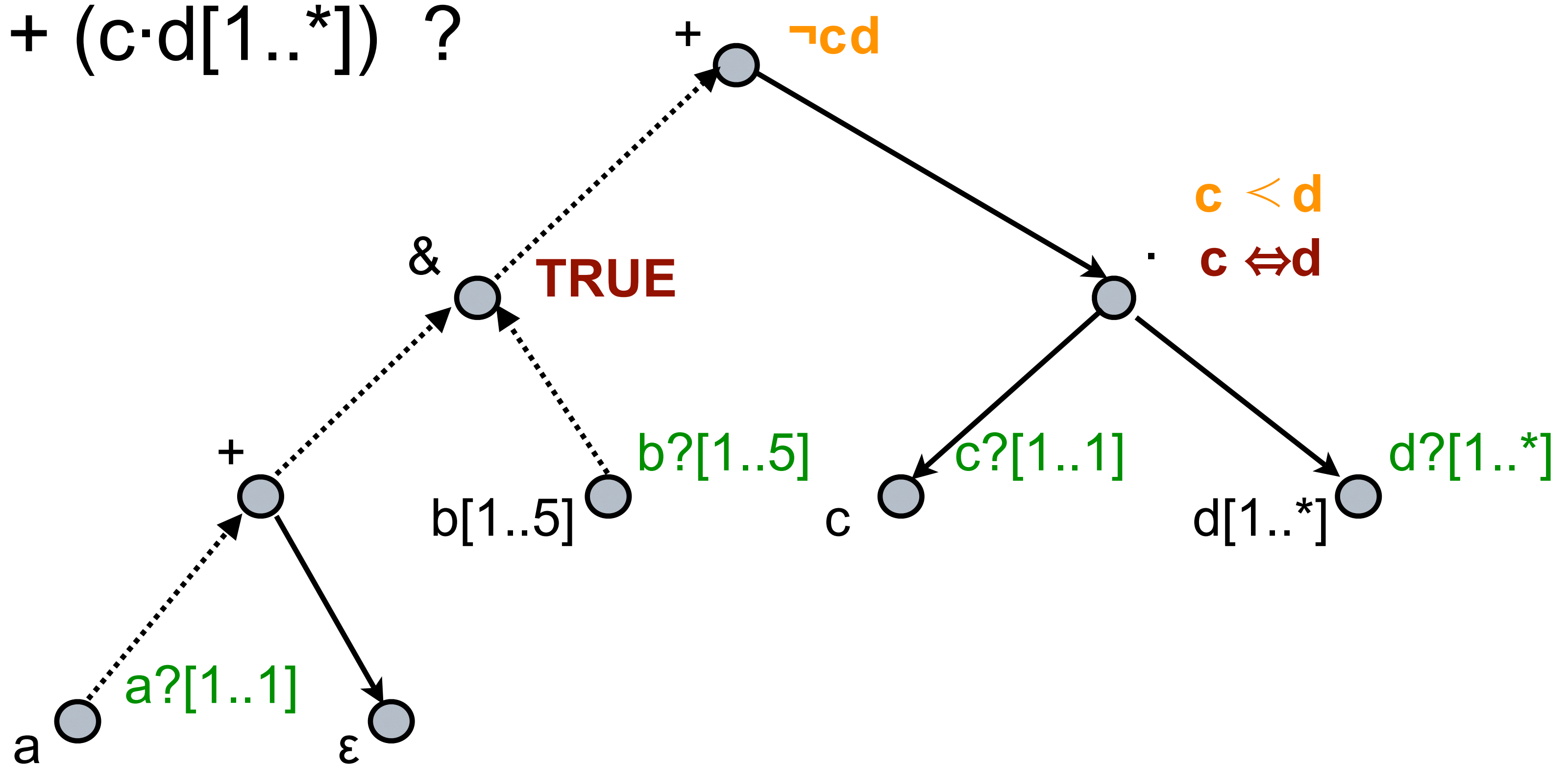


Complexity : O(|T|+|w|·depth(T))

We can do better : O(|T|+|w|) !

Avoiding redundant visits

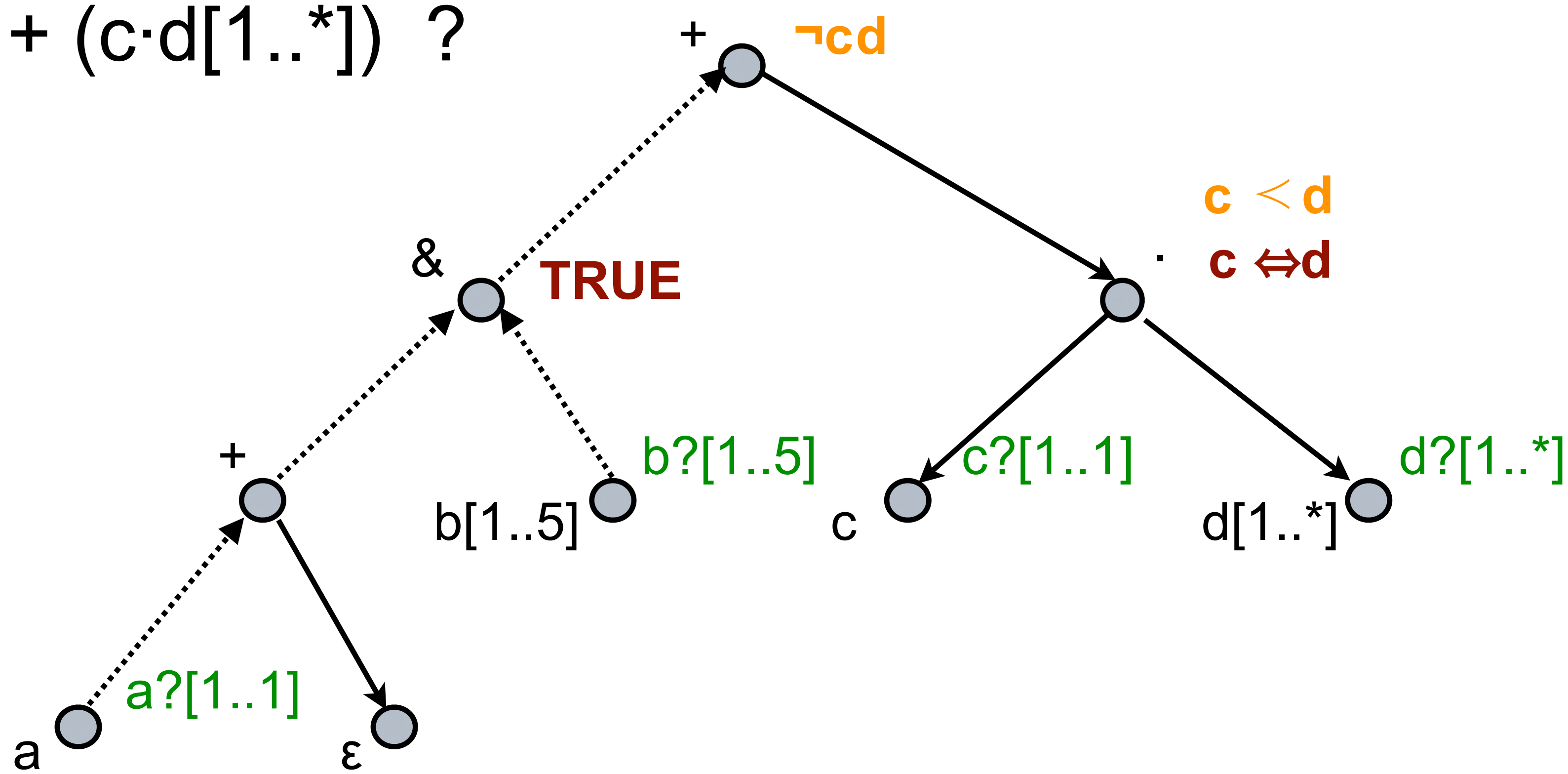
abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



Remark : the operations made for the second b were redundant !

Avoiding redundant visits

abb ∈ ((a+ε)&b[1..5]) + (c·d[1..*]) ?



Remark : the operations made for the second b were redundant !

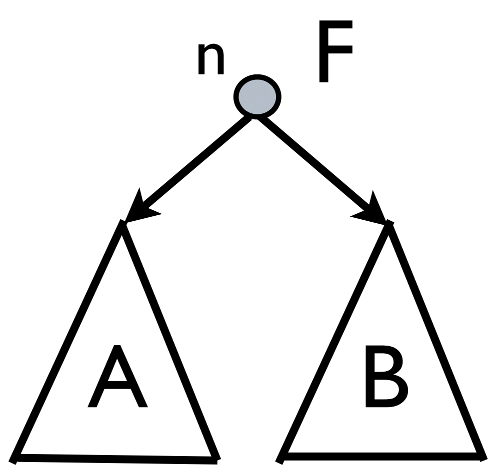
Once a symbol is met and processed, there is *almost* non reason to consider and process it if met again.

Stability

- A node **n** is visited each time a symbol in its left or right hand side sub-tree is met in w

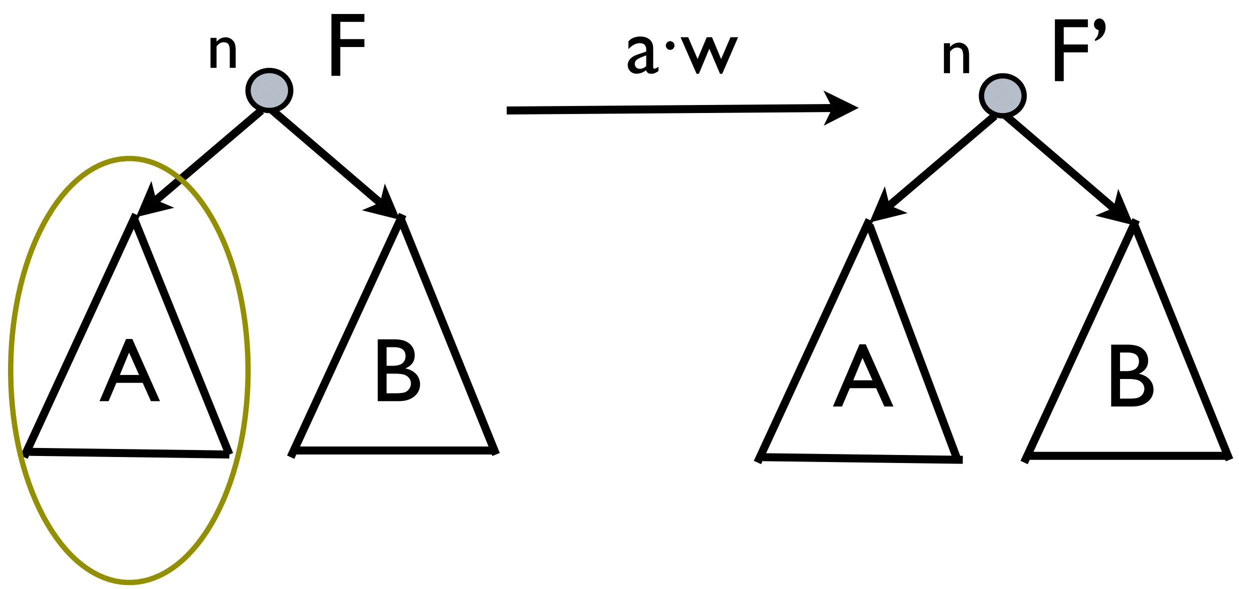
Stability

- A node **n** is visited each time a symbol in its left or right hand side sub-tree is met in w



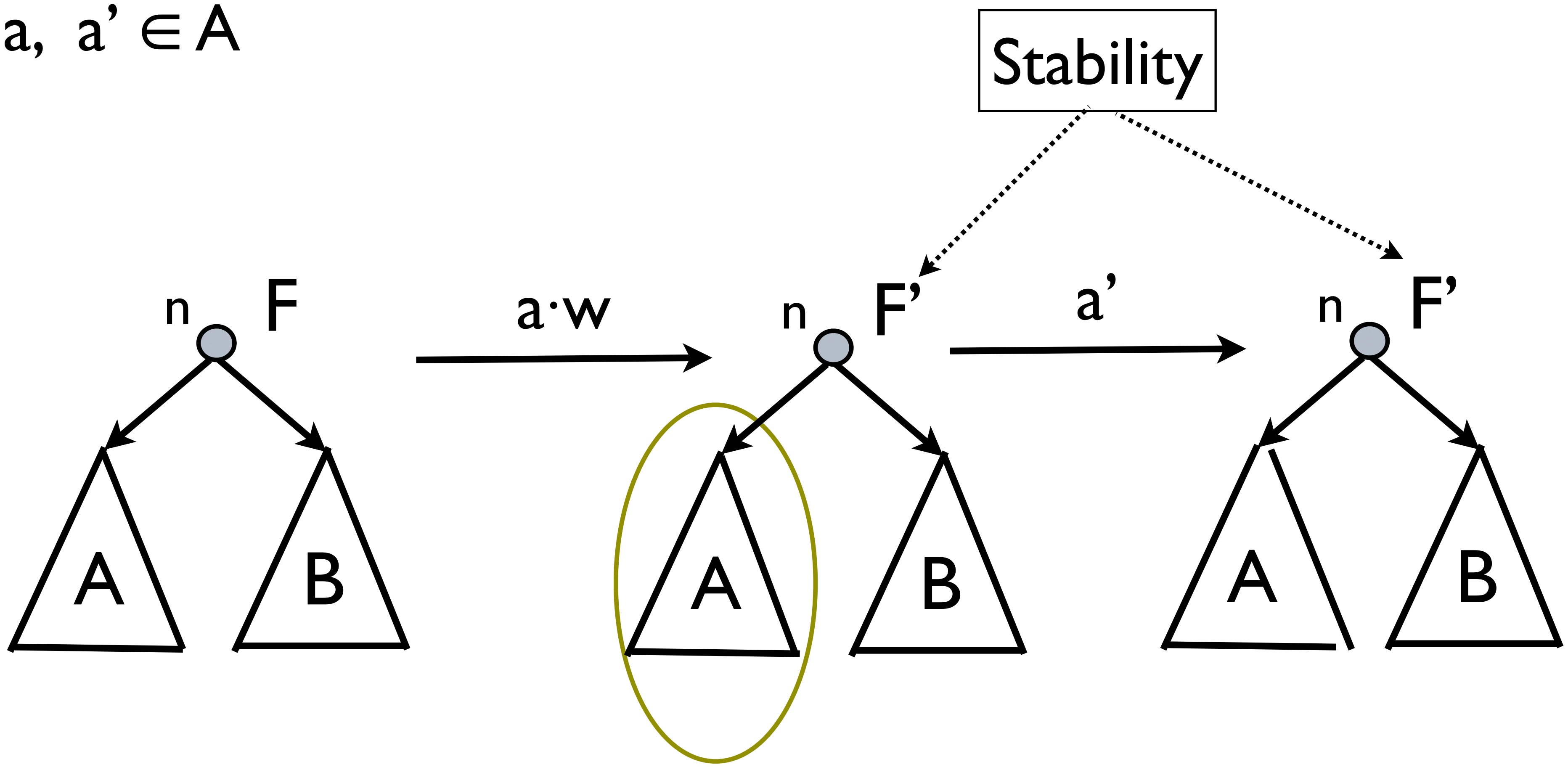
Stability

$a \in A$



Stability

a, a' ∈ A

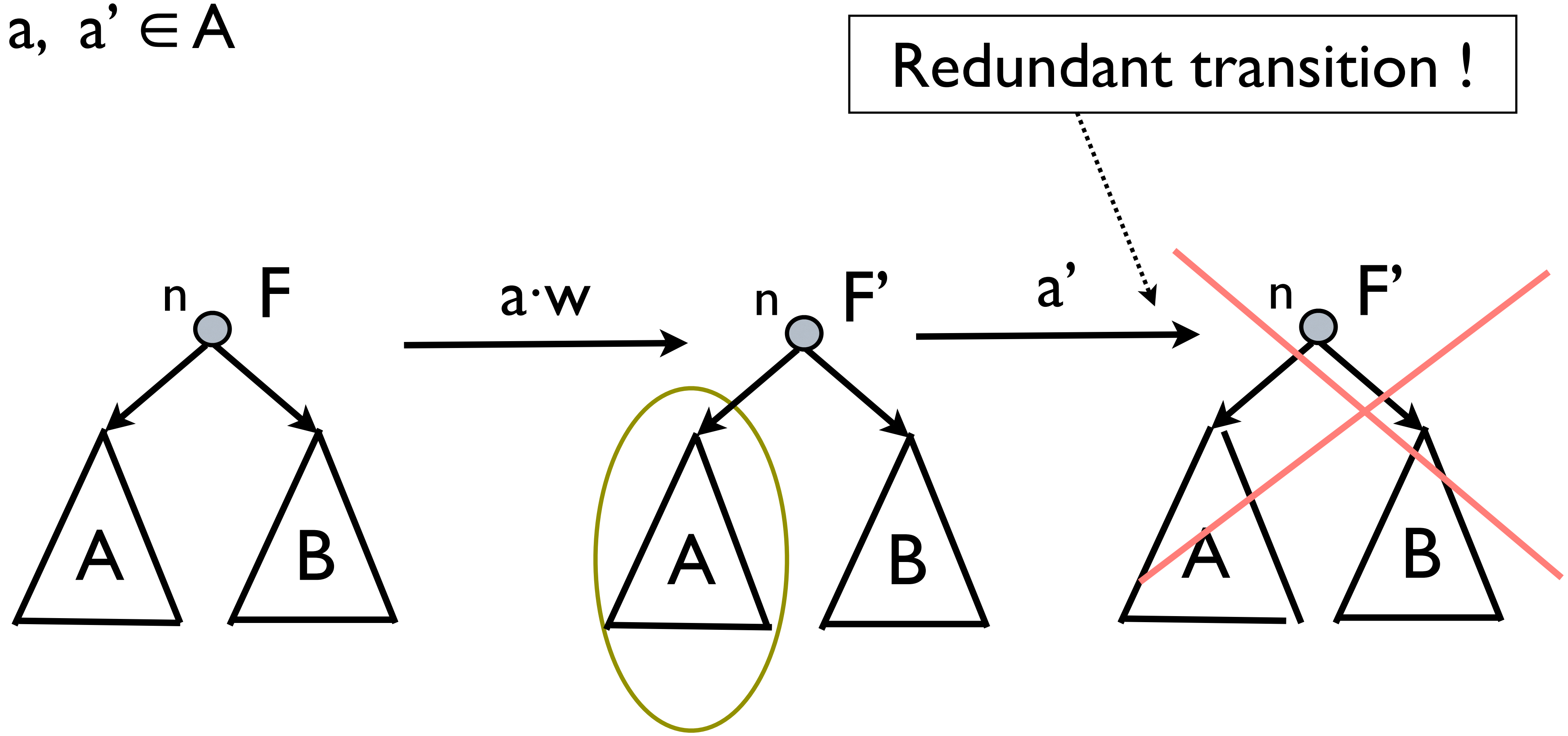


This is almost always true !

It is almost always true that n does not need to be visited more than once for symbols in A

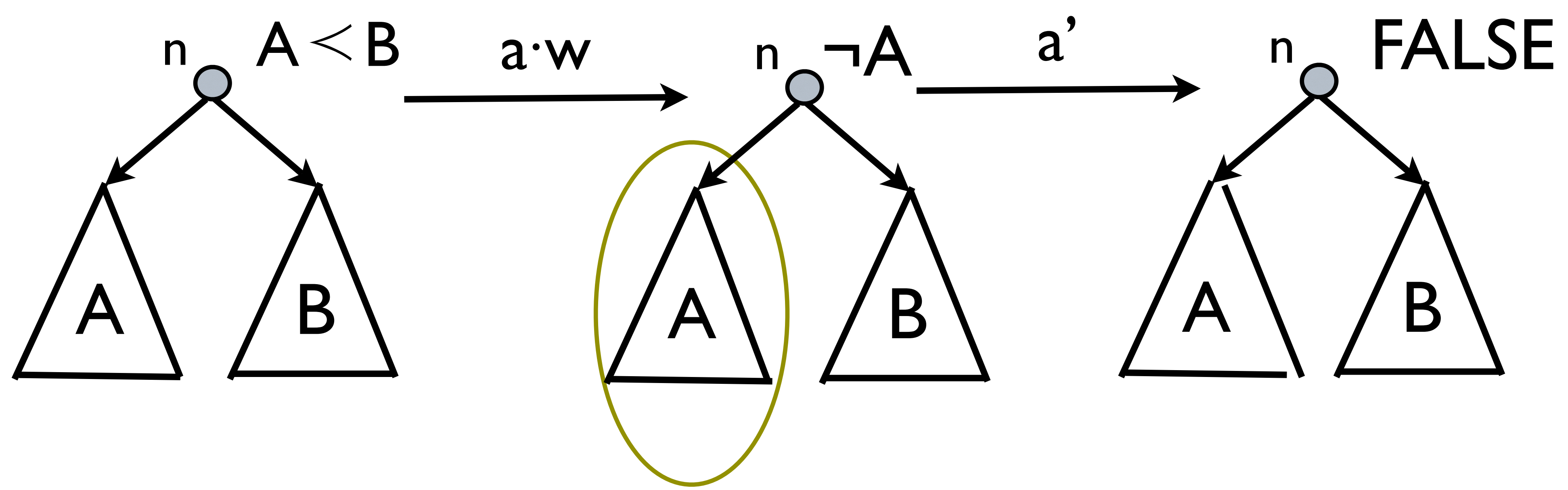
Stability

a, a' ∈ A



Stability - exception

a, a' ∈ A $w \cap B \neq \emptyset$



Residuation stops!

Exeption : F= A ≺ B and $w \cap B \neq \emptyset$

The linear algorithm

- B-stability always holds
- So during residuation each node needs to be processed/ visited at most three times.
  - For the pattern A-B-A with $F = A \prec B$
- Complexity $O(|T|+|w|)$

Some tests without residuation

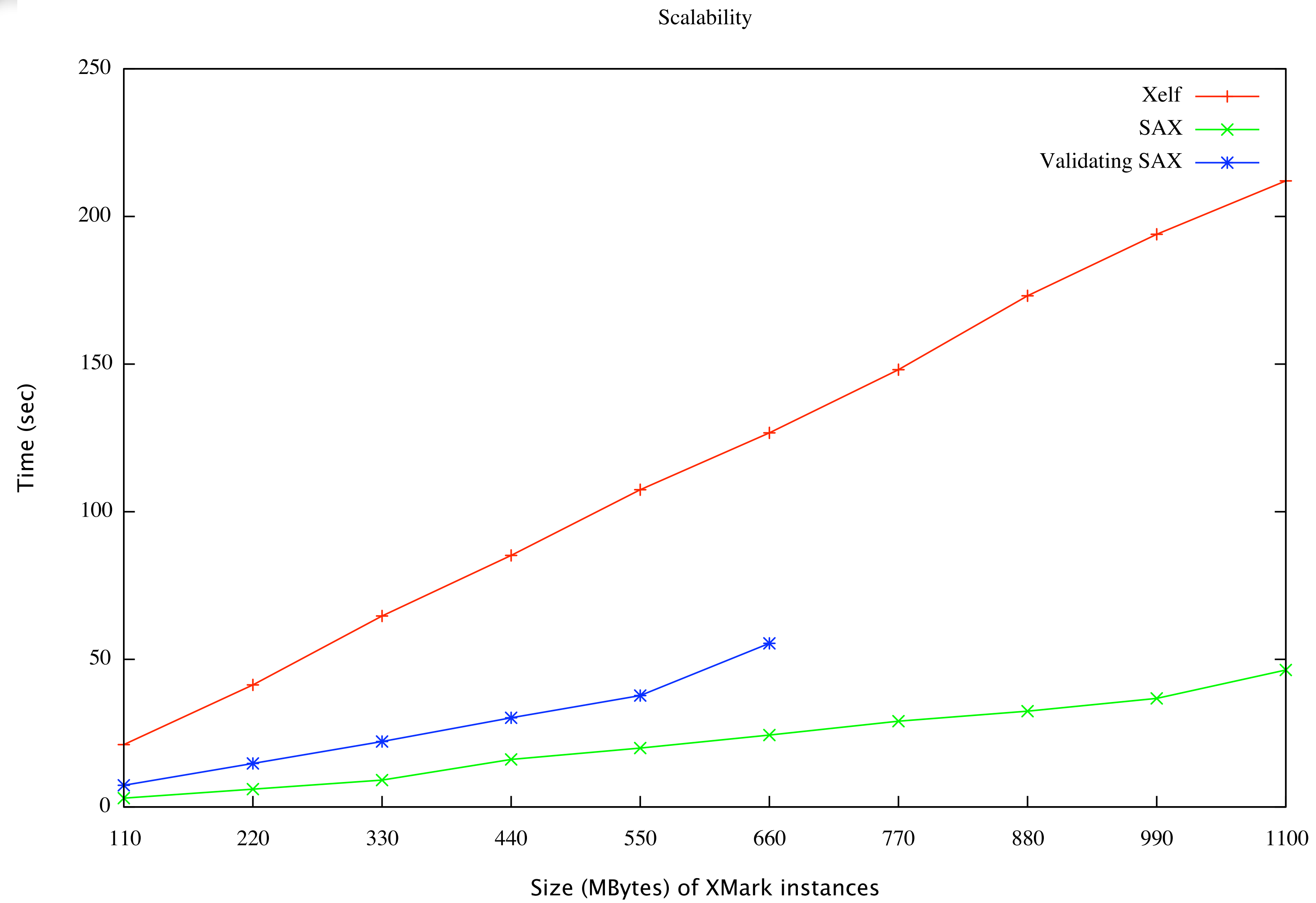


Figure 9: Scalability of Xelf.

Conclusions

- We have seen some main ideas behind the use of XML schema:
  - checking query correctness and result analysis
  - efficient document validation
  - query and update optimisation (time/space)
- Other interesting applications:
  - schema mapping maintenance in XML data integration (based on result analysis and a notion of type-projection) [DBPL05, PPDP06, TOIT09]
  - query-update independence: a technique derived from type-based projection can ensure highly precise analysis [work in progress, ask Federico Ulliana for details]

References

- D. Colazzo, G. Ghelli, P. Manghi, C. Sartiani. *Types for Path Correctness of XML queries.* ACM-SIGPLAN International Conference on Functional Programming (ICFP), 2004.
- D. Colazzo and C. Sartiani. *An Efficient Algorithm for XML Type Projection.* ACM-SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP), 2006.
- V. Benzaken, G. Castagna, D. Colazzo, and K. Nguyen. *Type-Based XML Projection*. International Conference on Very Large Databases (VLDB), 2006
- G. Ghelli, D. Colazzo and C. Sartiani. *Linear Time Membership for a Class of XML Types with Interleaving and Counting.* ACM Conference on Information and Knowledge Management (CIKM), 2008.

References

- D. Colazzo, G. Ghelli and C. Sartiani. *Efficient asymmetric inclusion between regular expression types.* International Conference on Database Theory (ICDT), 2009.
- D. Colazzo, G. Ghelli and C. Sartiani. *Efficient Inclusion for a Class of XML Types with Interleaving and Counting.* Information Systems. Volume 34, Issue 7, Pages 577-670, November, 2009.
- D. Colazzo, G. Ghelli, L. Pardini and C. Sartiani. *Linear Inclusion for XML Regular Expression Types*. ACM Conference on Information and Knowledge Management (CIKM), 2009.

- D. Colazzo, G. Ghelli and C. Sartiani. *Efficient asymmetric inclusion between regular expression types.* International Conference on Database Theory (ICDT), 2009.

- D. Colazzo and C. Sartiani. *Detection of Corrupted Schema Mappings in XML Data Integration Systems.* ACM Transactions on Internet Technology (TOIT), 2009.

Merci ! Any questions ?