Model Differences

How to specify and detect changes in graph-like models

Ulf Schwekendiek (ulf.schwekendiek@gmail.com)

26. March 2008

Outline

Outline

- Challenges

Outline

- Challenges
- Plain Text Differences - GNU Diff

Outline

- Challenges
- Plain Text Differences - GNU Diff
- XML-Differences Computation

Outline

- Challenges
- Plain Text Differences - GNU Diff
- XML-Differences Computation
- Project Idea

Challenges

Challenges

Challenges

- How to detect differences in models?

Challenges

- How to detect differences in models?
  - Idea 1: Take a textual representation of the model and run *diff*

Challenges

- How to detect differences in models?
  - Idea 1: Take a textual representation of the model and run *diff*
  - Idea 2: Since models can be represented in XML use the X-Diff algorithm in order to detect changes

Challenges

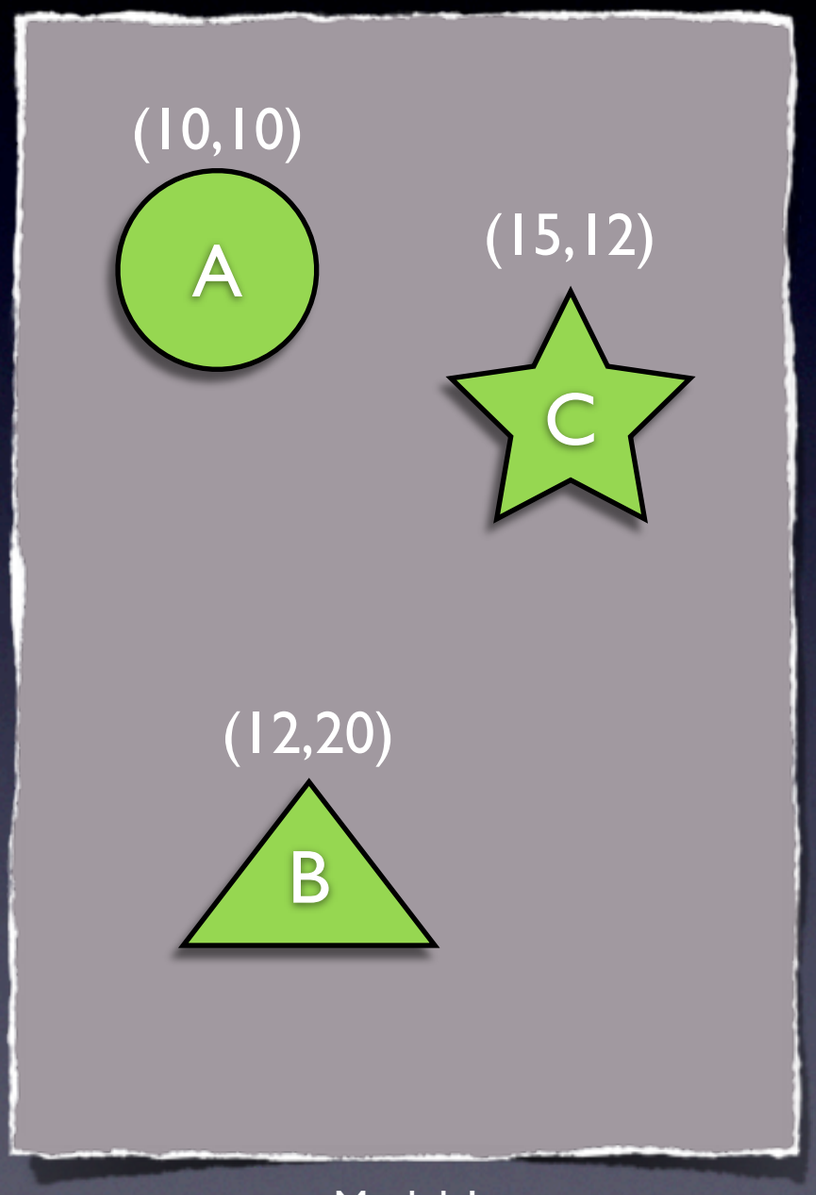
- How to detect differences in models?
  - Idea 1: Take a textual representation of the model and run *diff*
  - Idea 2: Since models can be represented in XML use the X-Diff algorithm in order to detect changes
  - Idea 3: MDE approach: Everything is a model! Use known difference algorithms and represent the difference as a model.

Challenges

Where is the “real”/important for MDE differences?

Challenges

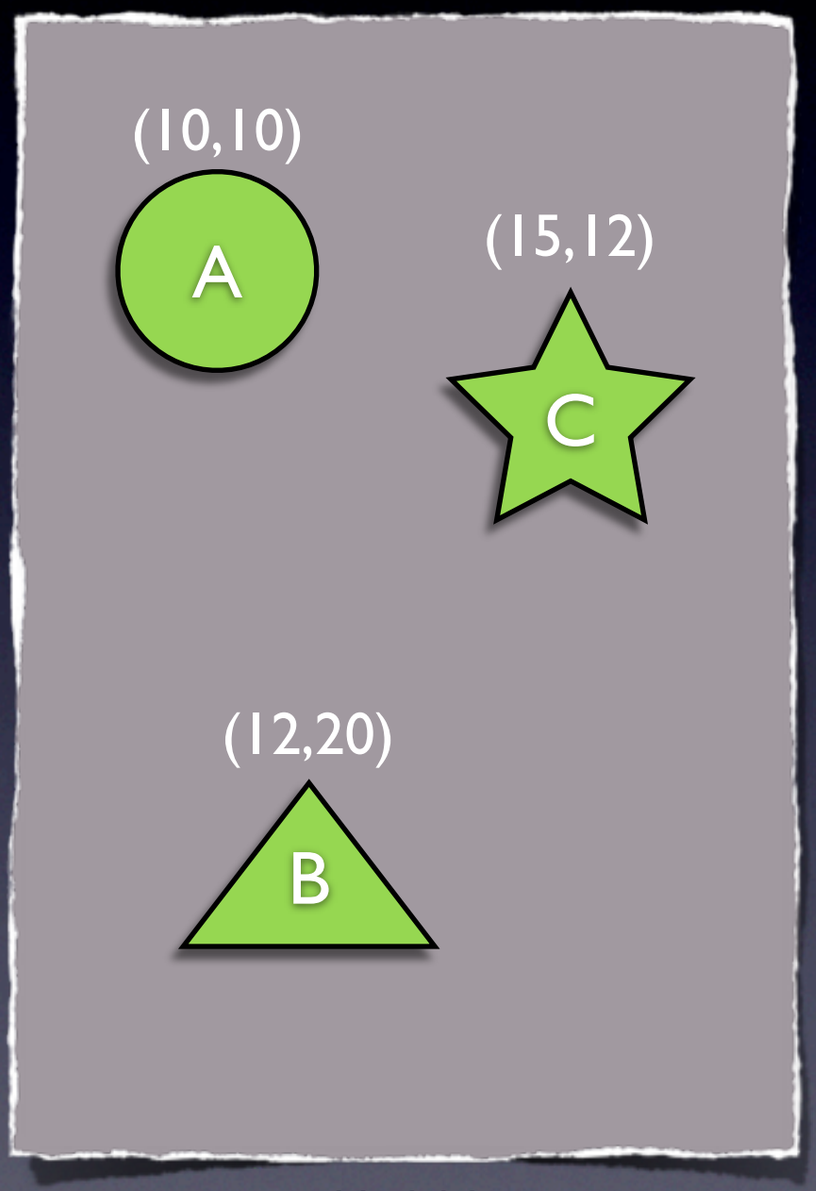
Where is the "real"/important for MDE differences?



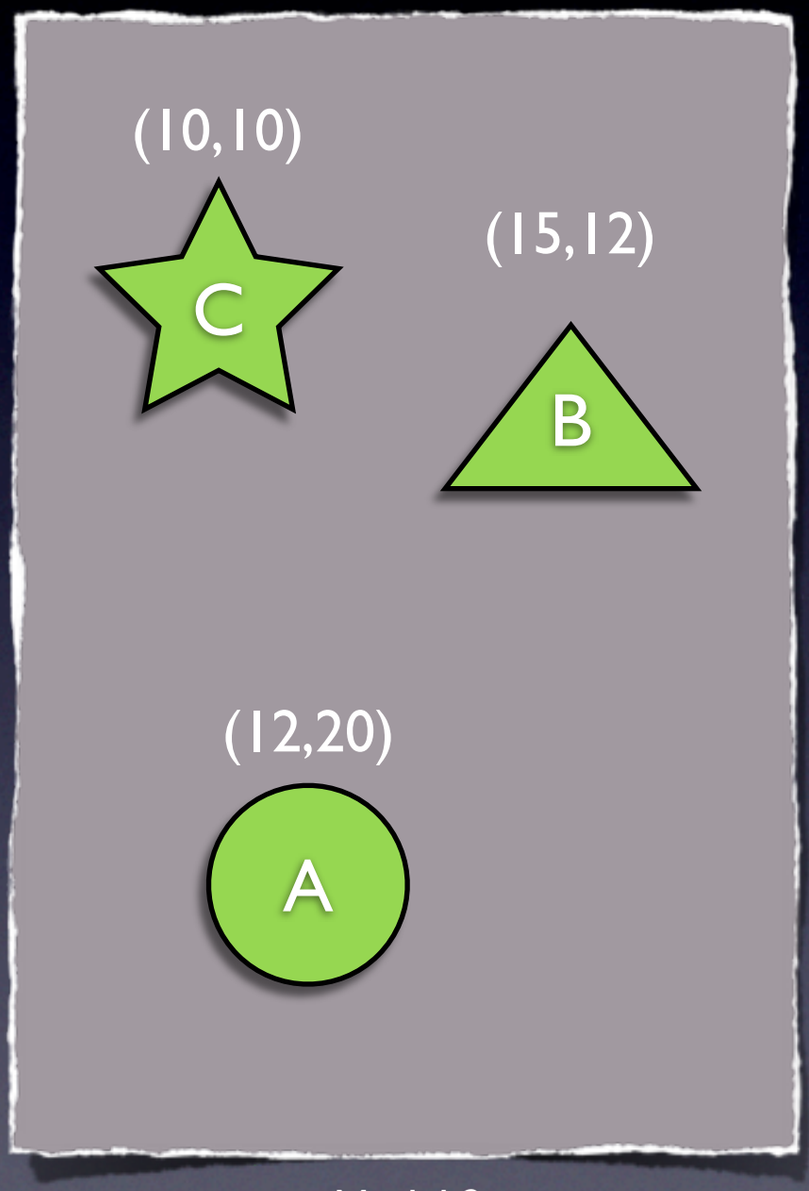
Model 1

Challenges

Where is the "real"/important for MDE differences?



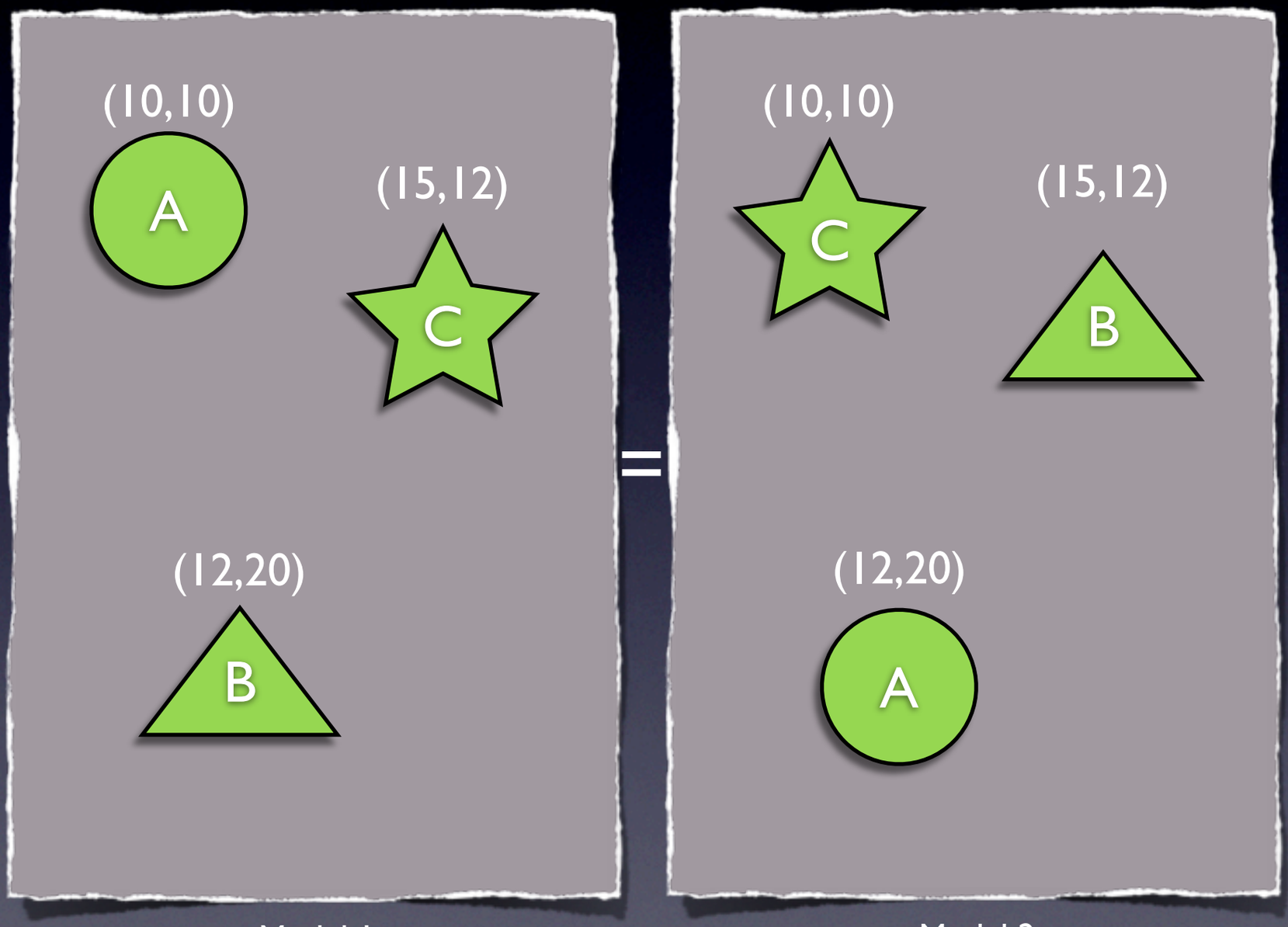
Model 1



Model 2

Challenges

Where is the "real"/important for MDE differences?



Model 1 Model 2

Challenges

Where is the "real"/important for MDE differences?

Model 1 Model 2 Model 3

Challenges

Where is the "real"/important for MDE differences?

Model 1 | Model 2 | Model 3

Plain Text Differences

GNU Diff

Challenges

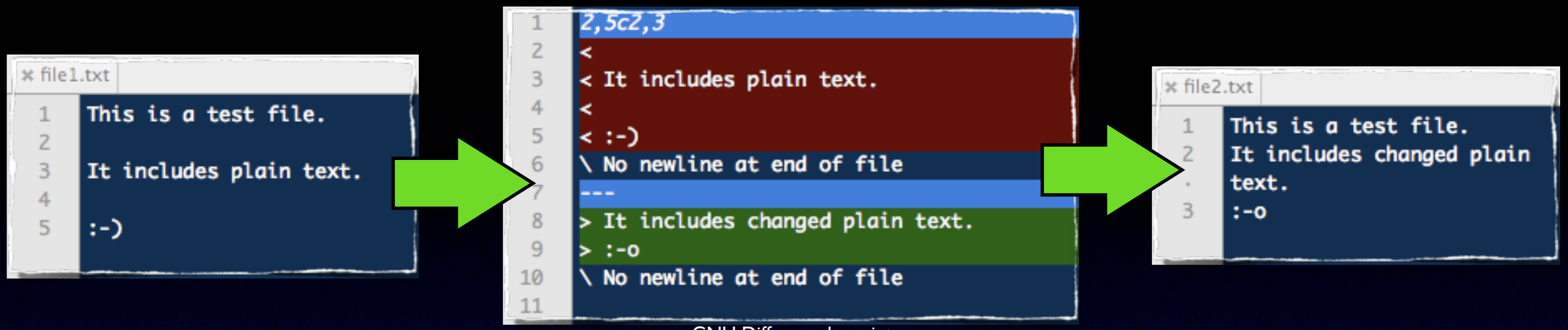
J. W. Hunt and M. D. McIlroy,

"An algorithm for differential file comparison"

Bell Telephone Laboratories CSTR #41 (1976)

- Detecting a difference in a plain text file is rather simple:
  - ***diff*** is used to compute the difference of two text strings (use of longest common subsequence algorithm) in *O(DN)*
  - Every single change in a text file will result a difference

What is a difference?



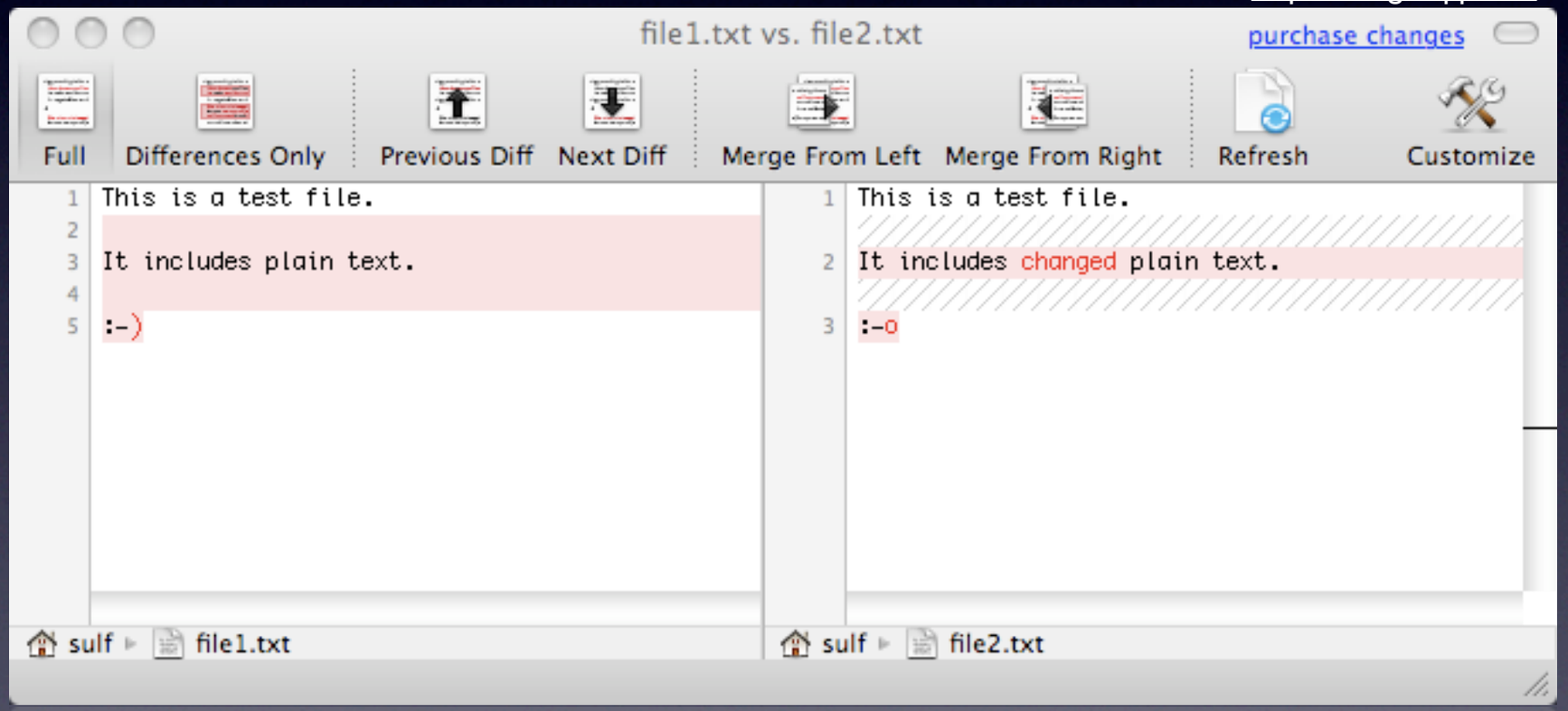
GNU Diff upgrade script

- We have two files/models/... and want to find the difference
- Difference is expressed as either
  - A script with add/delete/update directives describing the transformation from the source file/model/... to a target one
  - Graph coloring

Changes

- There are several tools that detect changes in a more graphical way:
- Example: Changes V.1.0

Source: http://changesapp.com



XML Differences

X-Diff

GNU diff on XML files

GNU Diff upgrade script

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```


Ulf Schwekendiek (ulf.schwekendiek@gmail.com)

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```


Ulf Schwekendiek (ulf.schwekendiek@gmail.com)

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```


Ulf Schwekendiek (ulf.schwekendiek@gmail.com)

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```

book1-book2.diff

```
1  3,4c3,4
2  <     <Title>Harry Potter and the Sorcerer's Stone</Title>
3  <   <Price>$10.00</Price>
4  ---
5  >     <Title>The adventures of Tom Sawyer</Title>
6  >   <Price>$29.00</Price>
7  7,8c7,8
8  <     <Title>The adventures of Tom Sawyer</Title>
9  <   <Price>$5.00</Price>
10 ---
11 >     <Title>Harry Potter and the Sorcerer's Stone</Title>
12 >   <Price>$10.00</Price>
13
```

GNU Diff upgrade script

GNU diff on XML files

book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```

GNU diff on XML files

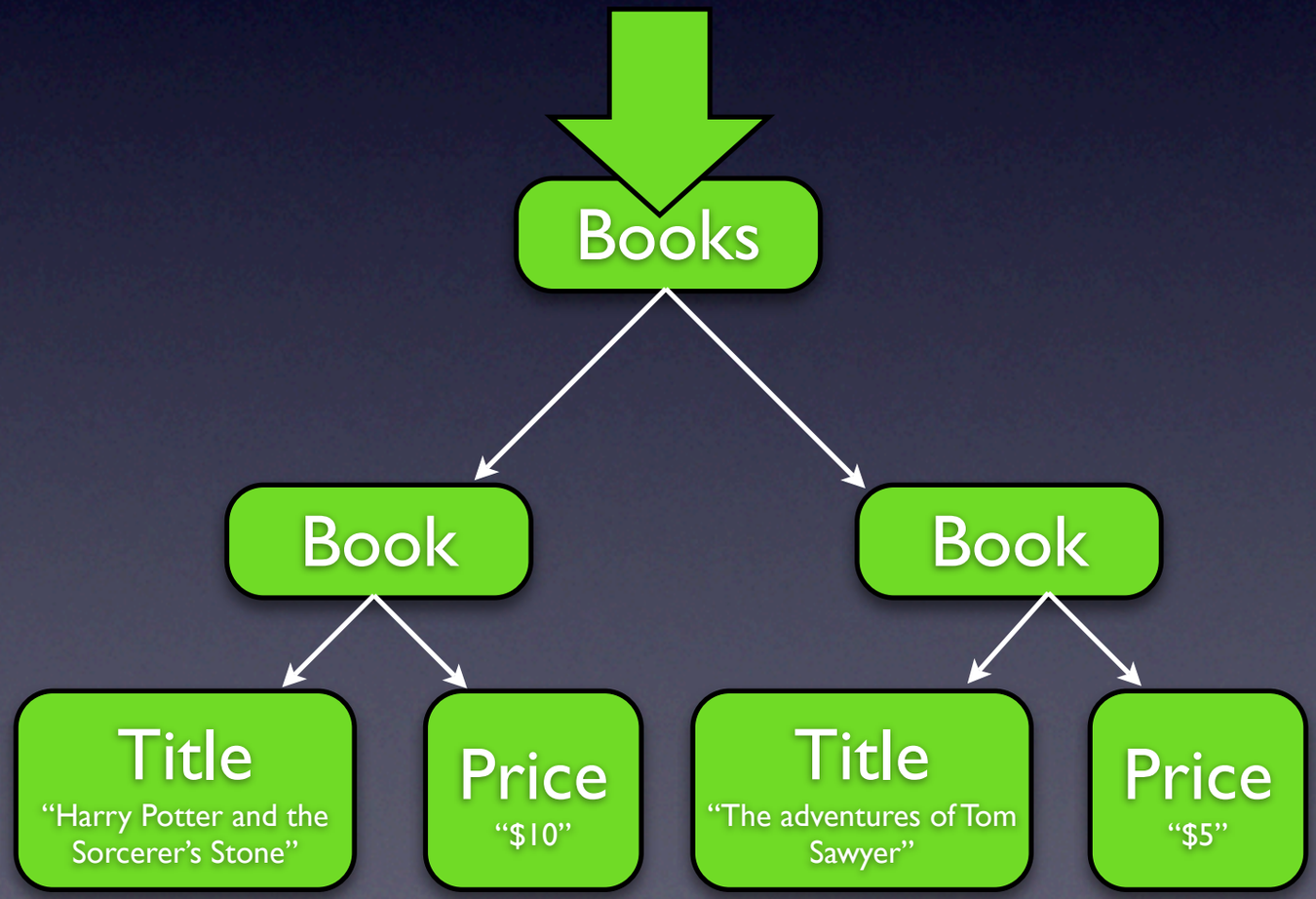
book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```



GNU diff on XML files

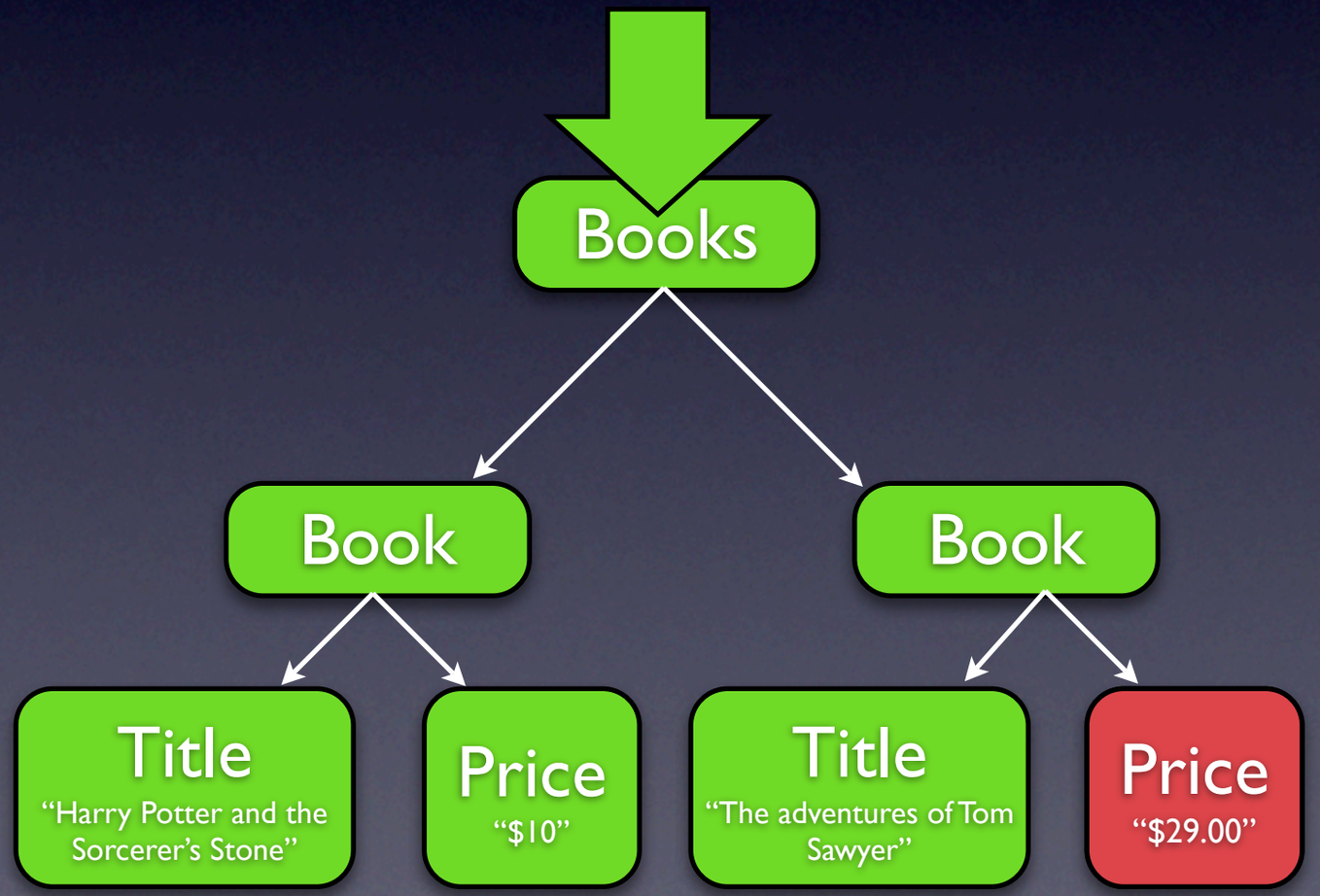
book1.xml

```
1  <Books>
2    <Book>
3      <Title>Harry Potter and the Sorcerer's Stone</Title>
4      <Price>$10.00</Price>
5    </Book>
6    <Book>
7      <Title>The adventures of Tom Sawyer</Title>
8      <Price>$5.00</Price>
9    </Book>
10 </Books>
```

vs.

book2.xml

```
1  <Books>
2    <Book>
3      <Title>The adventures of Tom Sawyer</Title>
4      <Price>$29.00</Price>
5    </Book>
6    <Book>
7      <Title>Harry Potter and the Sorcerer's Stone</Title>
8      <Price>$10.00</Price>
9    </Book>
10 </Books>
```



X-Diff

Y. Wang, D.J. DeWitt and J. Cai

"An Effective Change Detection Algorithm for XML Documents"

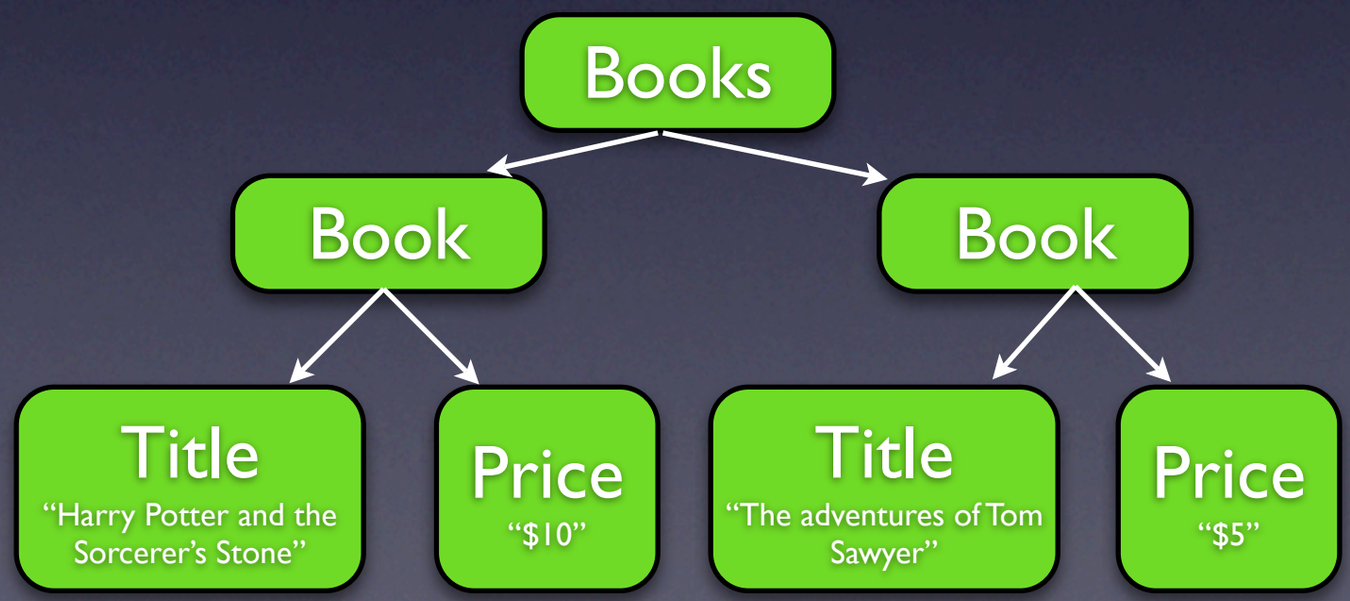
University of Wisconsin (2003)

- XML structure is hierarchical and can be represented as a tree
- Unordered trees
  - only ancestor relationship is important
  - optimal change detection of unordered trees is NP complete[1]
- Algorithm generates near optimal edit scripts
- Two trees are *isomorphic* if they are identical except for the ordering of siblings. X-Diff considers two trees as equivalent if they are *isomorphic*.

[1] K. Zhang, R. Statman, D. Shasha, "On the Editing Distance betweem Unordered Labeled Trees", *Information Processing Letters*, 42: 133-139, 1992

Tree Definition

- A tree has three kinds of nodes:
  - **Element** nodes - non-leaf nodes with one label, *name*
  - **Text** nodes - leaf nodes with one label, *value*
  - **Attribute** nodes - leaf nodes with two labels, *name* and *value*



Algorithm

Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.

Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model

Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node

Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched

Algorithm

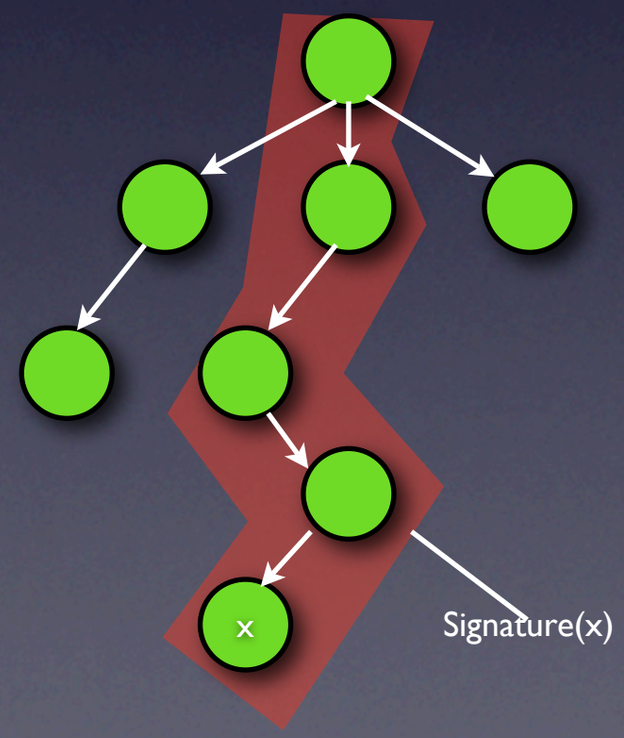
- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes

Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes
  - Signature is introduced

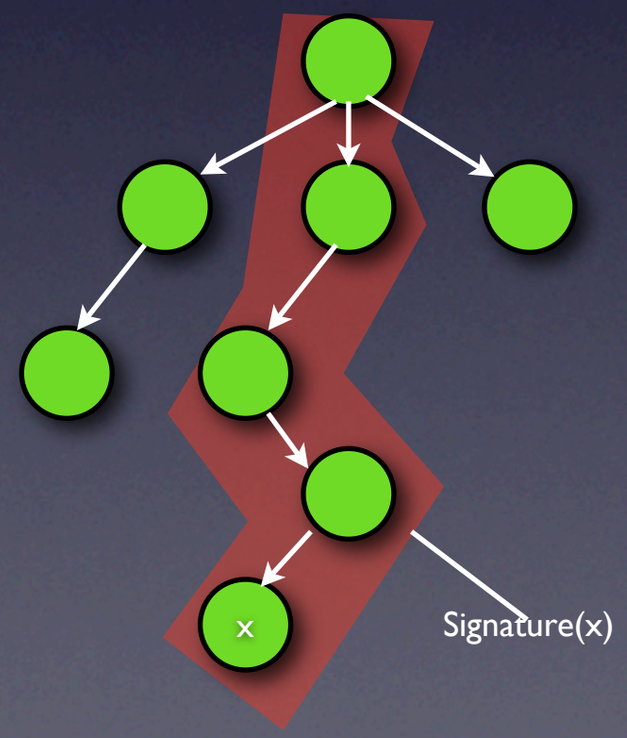
Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes
  - Signature is introduced



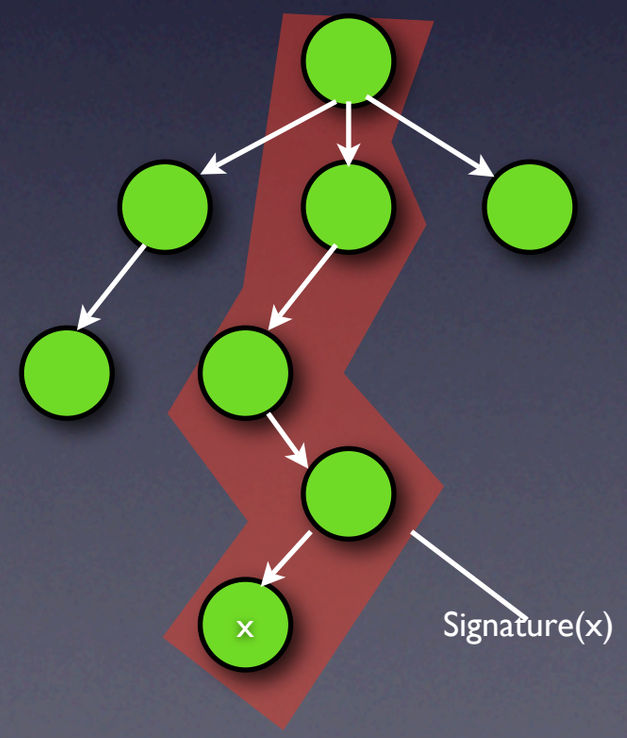
Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes
  - Signature is introduced
- Based on minimum-Cost Matching an upgrade script is generated



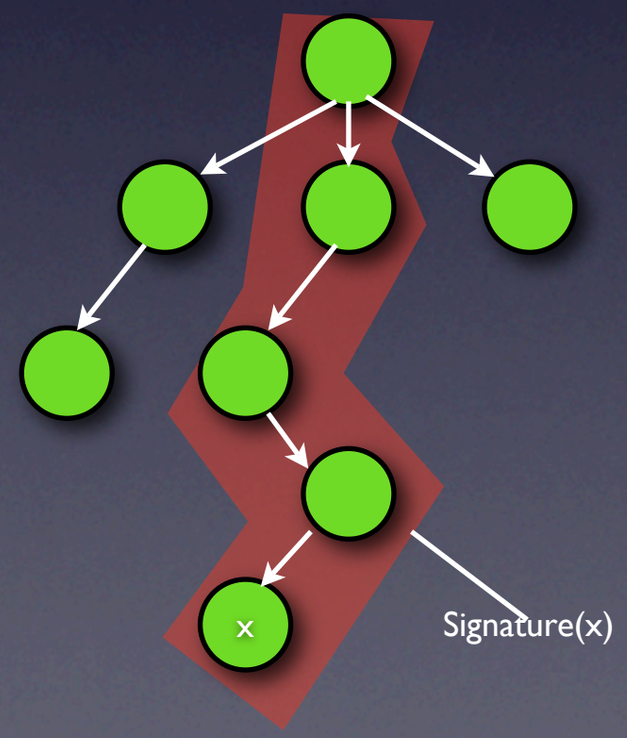
Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes
  - Signature is introduced
- Based on minimum-Cost Matching an upgrade script is generated
- Algorithm only computes editing distances between nodes that have the same signature

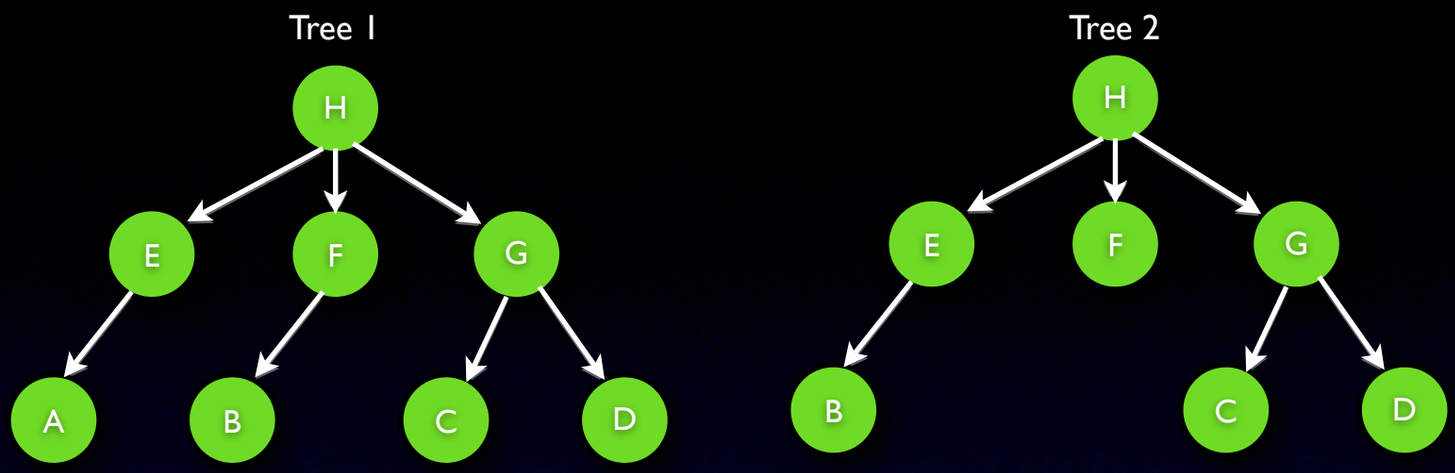


Algorithm

- In order to compare edit scripts a cost model is introduced, which also affect the algorithm design.
  - X-Diff uses a simple cost model
- Unnecessary to compare each source node with any other target node
  - only nodes of the same type are matched
  - text nodes should not be matched with element nodes or attribute nodes
  - Signature is introduced
- Based on minimum-Cost Matching an upgrade script is generated
- Algorithm only computes editing distances between nodes that have the same signature
  - Important to archive polynomial time complexity



Calculating minimum-Cost matching

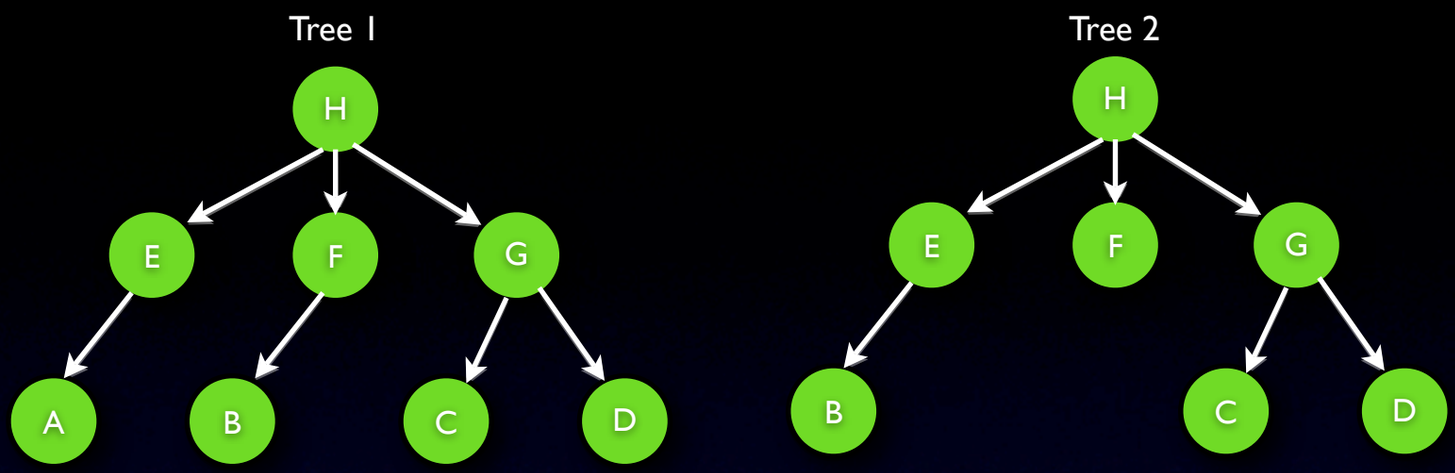


N1

N2

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

Calculating minimum-Cost matching

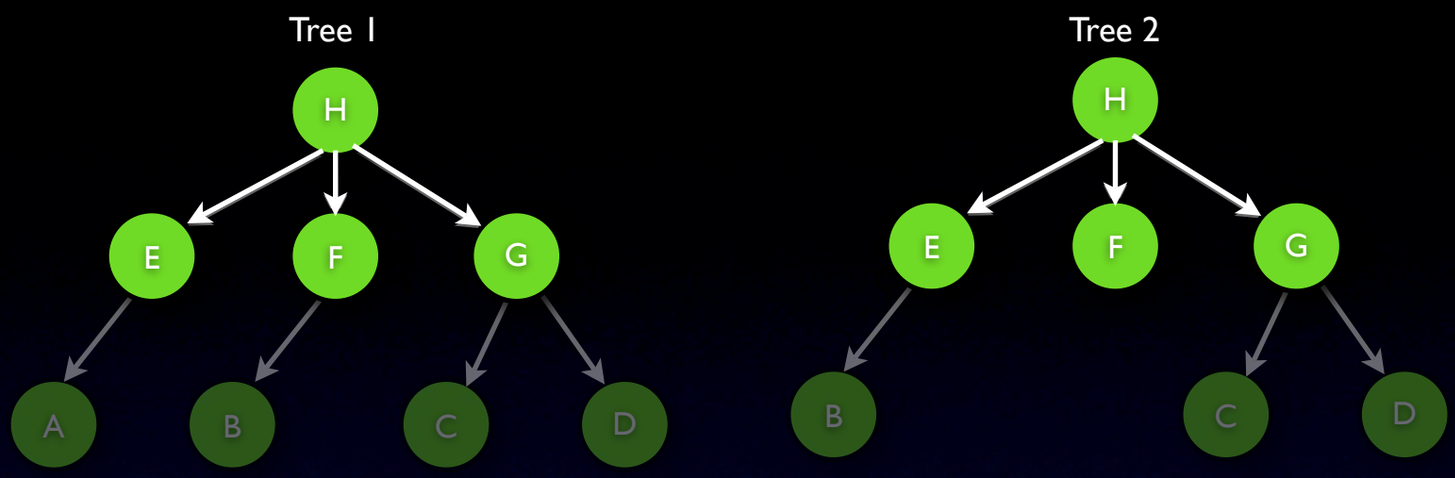


N1

N2

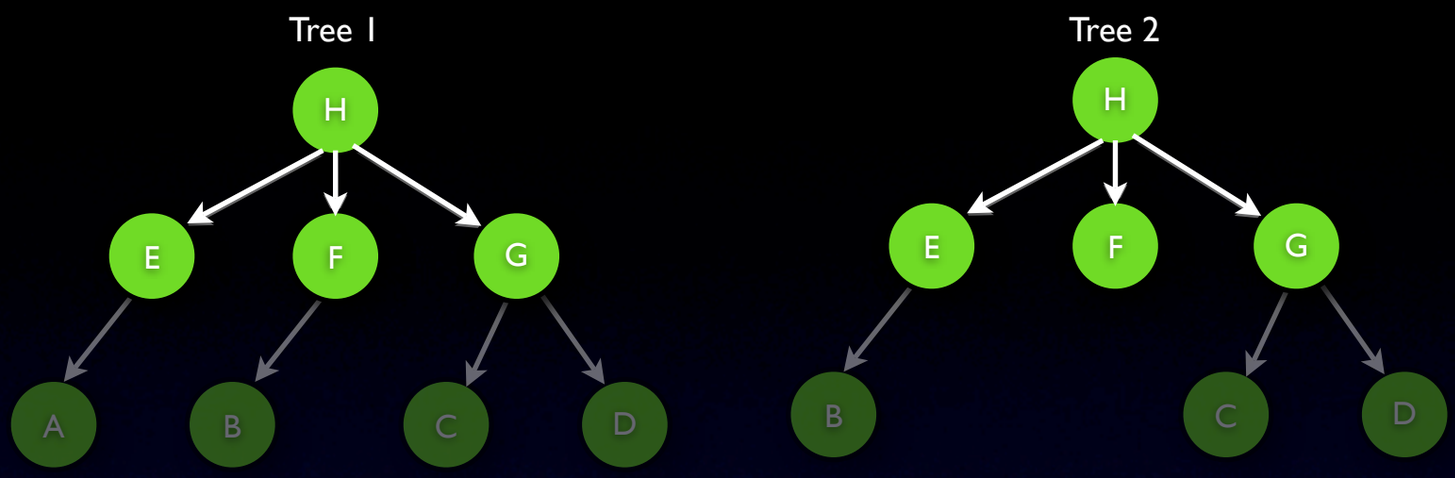
Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

Calculating minimum-Cost matching



Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

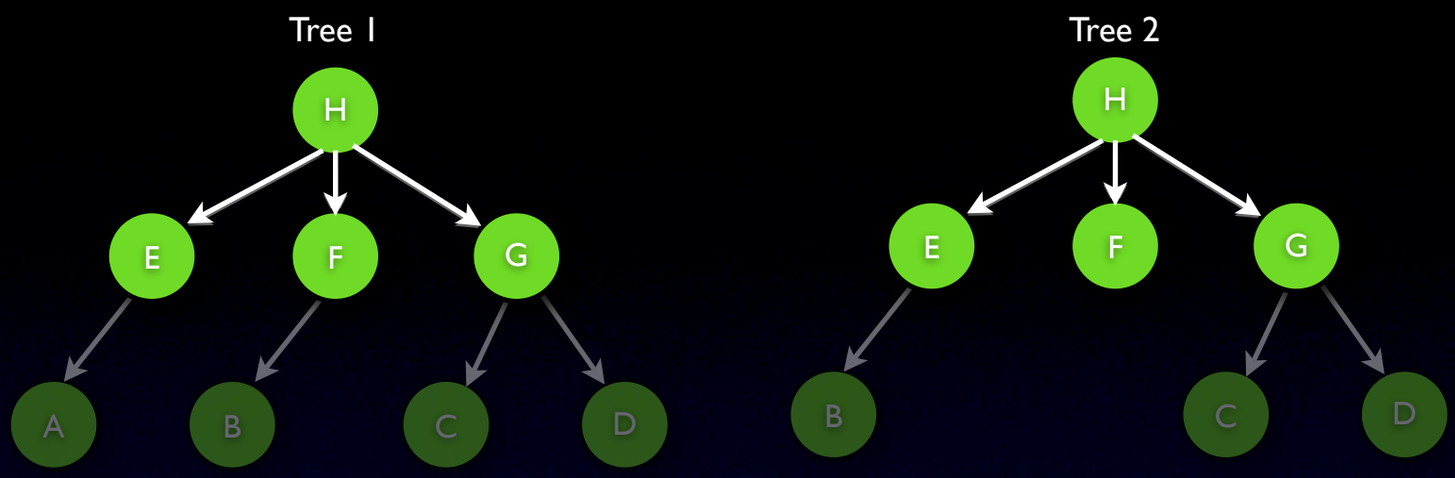
Calculating minimum-Cost matching



dist(A,B) = 1

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

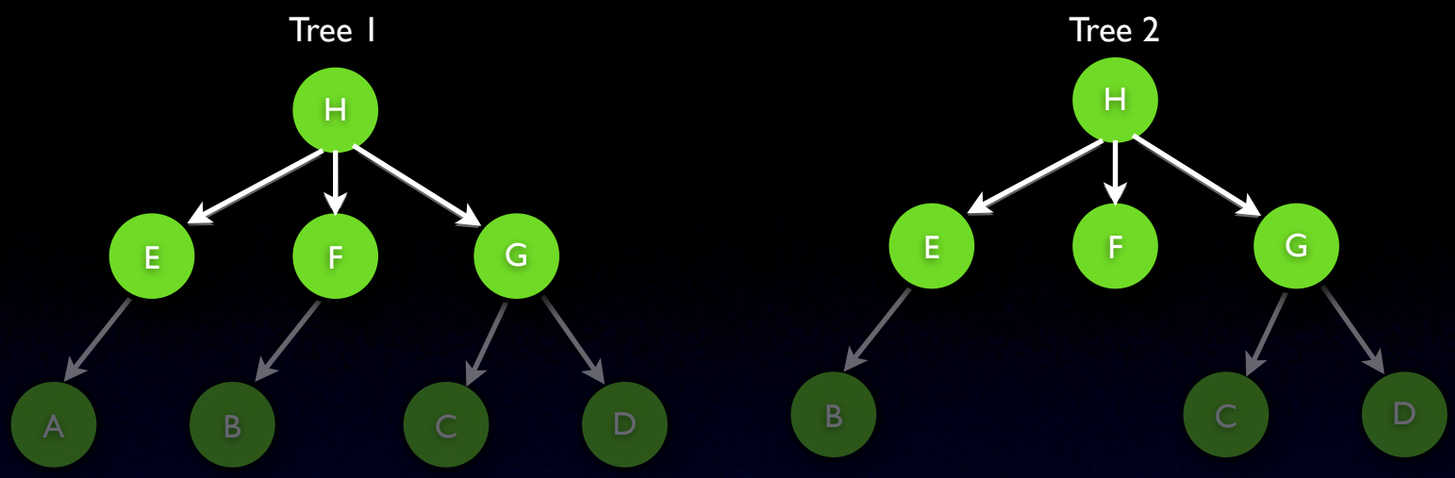
Calculating minimum-Cost matching



dist(A,B) = 1
dist(C,C) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node *x* in N_1
    For every node *y* in N_2
      If Signature(*x*) = Signature(*y*)
        Compute Dist(*x*, *y*);
        Save matching (*x*, *y*) with Dist(*x*, *y*) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping $(x, y) \in M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

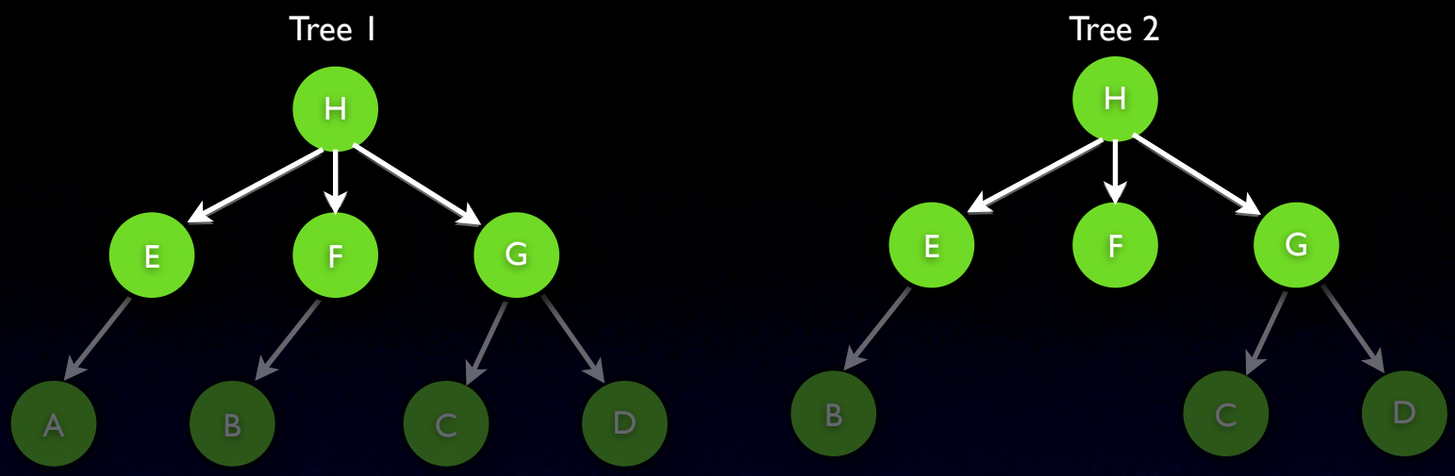
Calculating minimum-Cost matching



dist(A,B) = 1
dist(C,C) = 0
dist(D,D) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

Calculating minimum-Cost matching



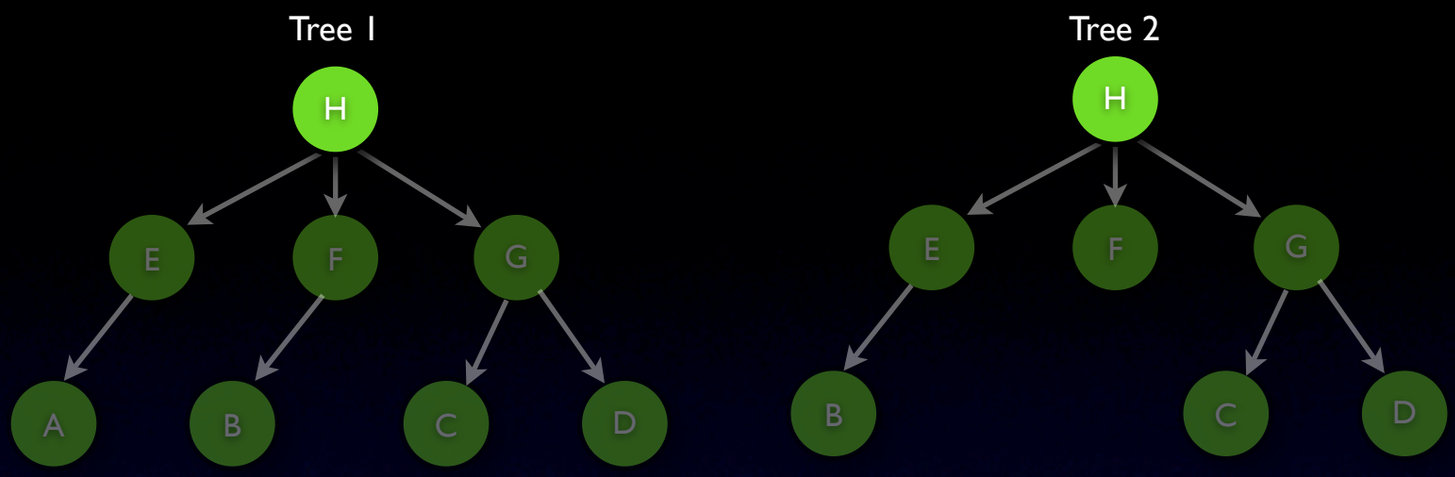
N1

N2

dist(A,B) = 1
dist(C,C) = 0
dist(D,D) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

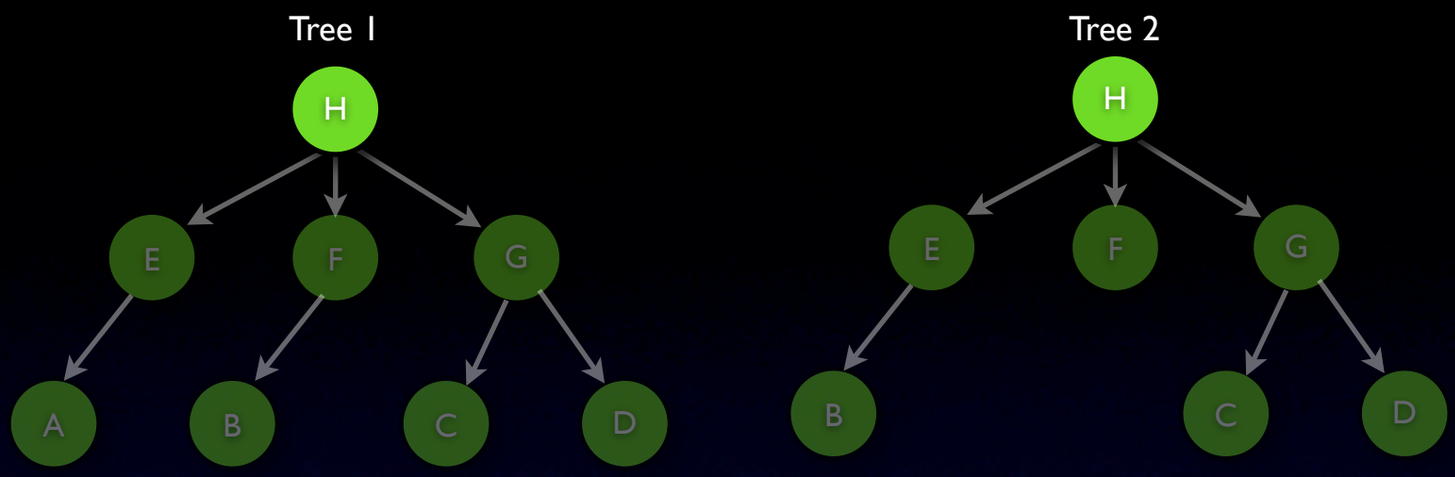
Calculating minimum-Cost matching



dist(A,B) = 1
dist(C,C) = 0
dist(D,D) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

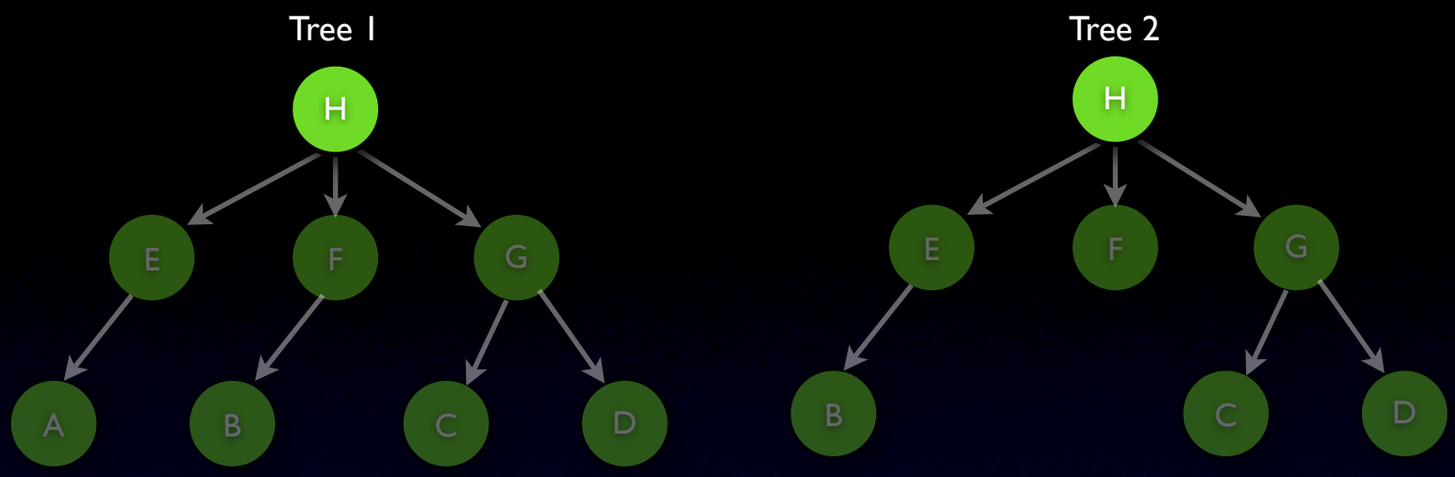
Calculating minimum-Cost matching



dist(A,B) = 1 dist(E,E) = 0
dist(C,C) = 0
dist(D,D) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node *x* in N_1
    For every node *y* in N_2
      If Signature(*x*) = Signature(*y*)
        Compute Dist(*x*, *y*);
        Save matching (*x*, *y*) with Dist(*x*, *y*) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping $(x, y) \in M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

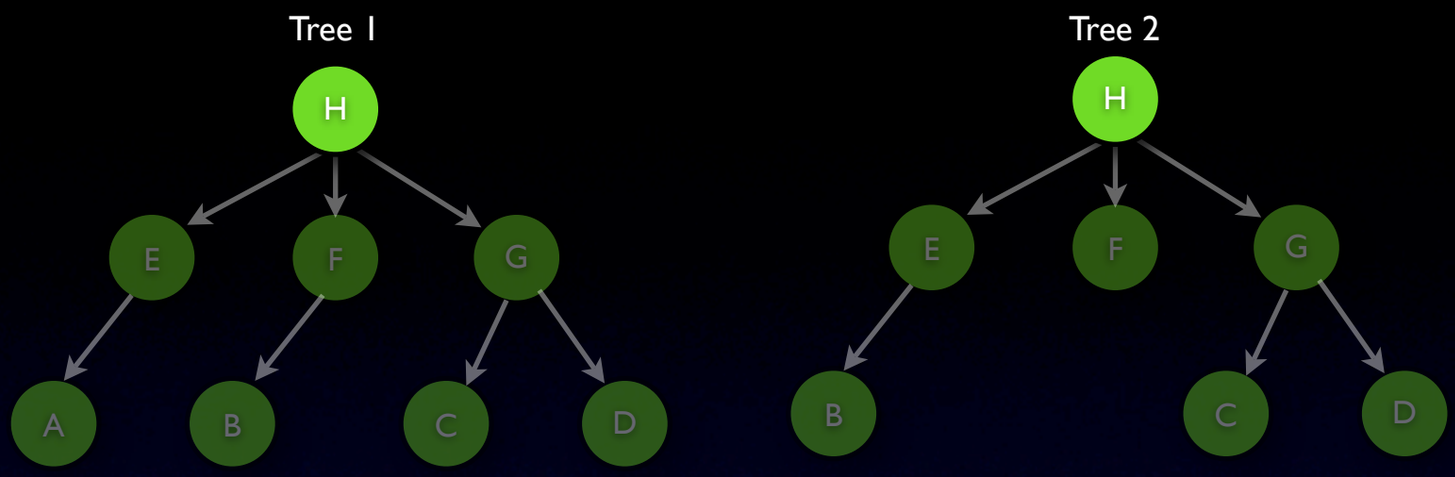
Calculating minimum-Cost matching



dist(A,B) = 1 dist(E,E) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

Calculating minimum-Cost matching



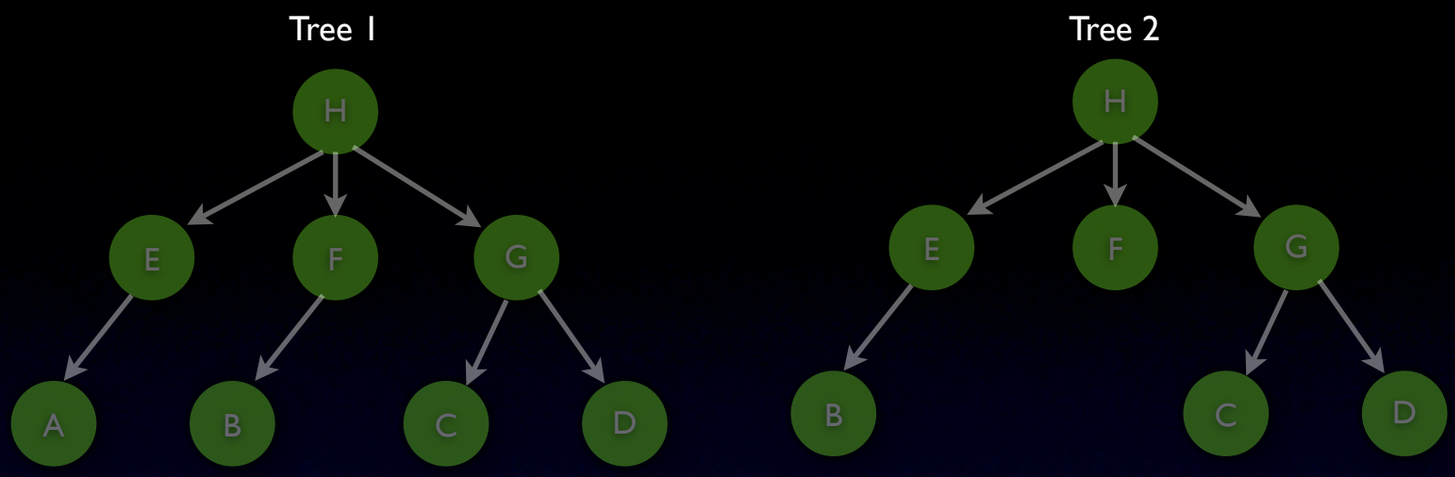
N1

N2

dist(A,B) = 1 dist(E,E) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

Calculating minimum-Cost matching



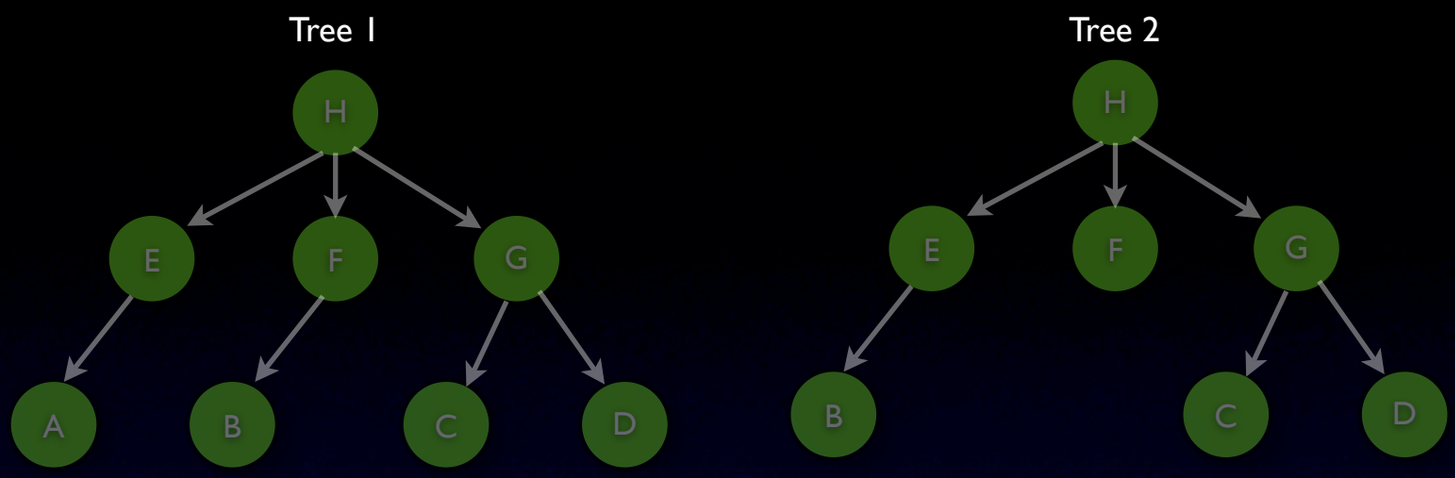
N1 H

N2 H

dist(A,B) = 1 dist(E,E) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

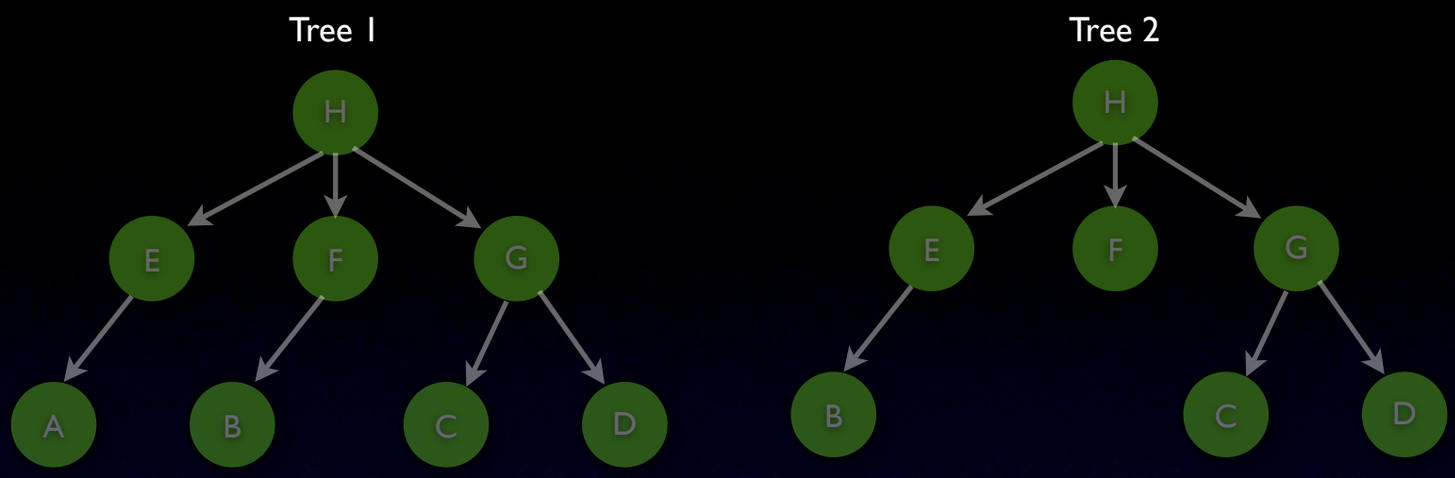
Calculating minimum-Cost matching



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

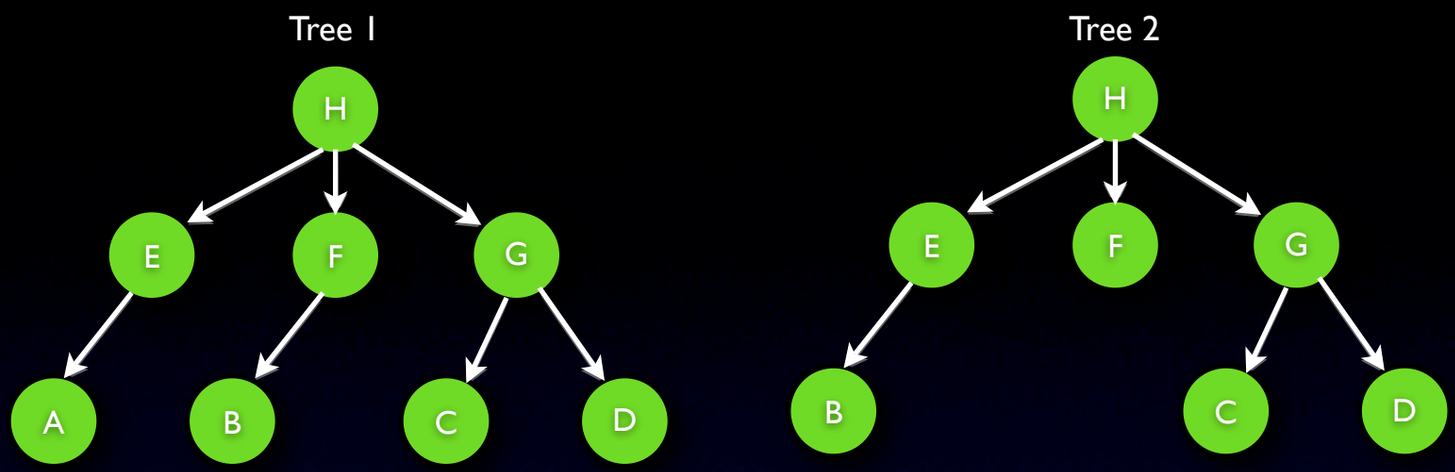
Calculating minimum-Cost matching



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2.
Output: a minimum-cost matching $M_{min}(T_1, T_2)$.
Initialize: set initial working sets
  N_1 = {all leaf nodes in T_1}, N_2 = {all leaf nodes in T_2}.
  Set the Distance Table DT = { }.
/* Step 1: Reduce matching space */
Filter out next-level subtrees that have equal XHash values.
/* Step 2: compute editing distance for ($T_1 \rightarrow T_2$) */
DO {
  For every node x in N_1
    For every node y in N_2
      If Signature(x) = Signature(y)
        Compute Dist(x, y);
        Save matching (x, y) with Dist(x, y) in DT.
  Set N_1 = {parent nodes of previous nodes in N_1};
    N_2 = {parent nodes of previous nodes in N_2}.
} While (both N_1 and N_2 are not empty).
/* Step 3: mark matchings on T_1 and T_2. */
Set $M_{min}(T_1, T_2)$ = { }
If Signature(Root(T_1)) ≠ Signature(Root(T_2))
  Return; /* $M_{min}(T_1, T_2)$ = { }*/
Else
  Add (Root(T_1), Root(T_2)) to $M_{min}(T_1, T_2)$.
  For every non-leaf node mapping (x, y) ∈ $M_{min}(T_1, T_2)$
    Retrieve matchings between their child nodes that are stored in DT.
    Add child node matchings into $M_{min}(T_1, T_2)$.

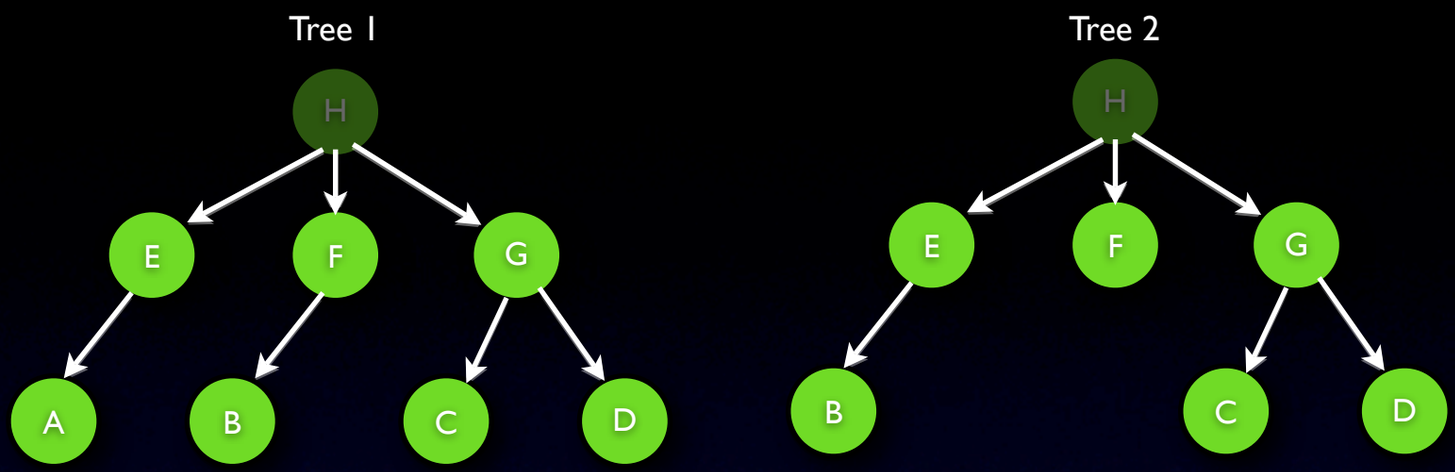
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

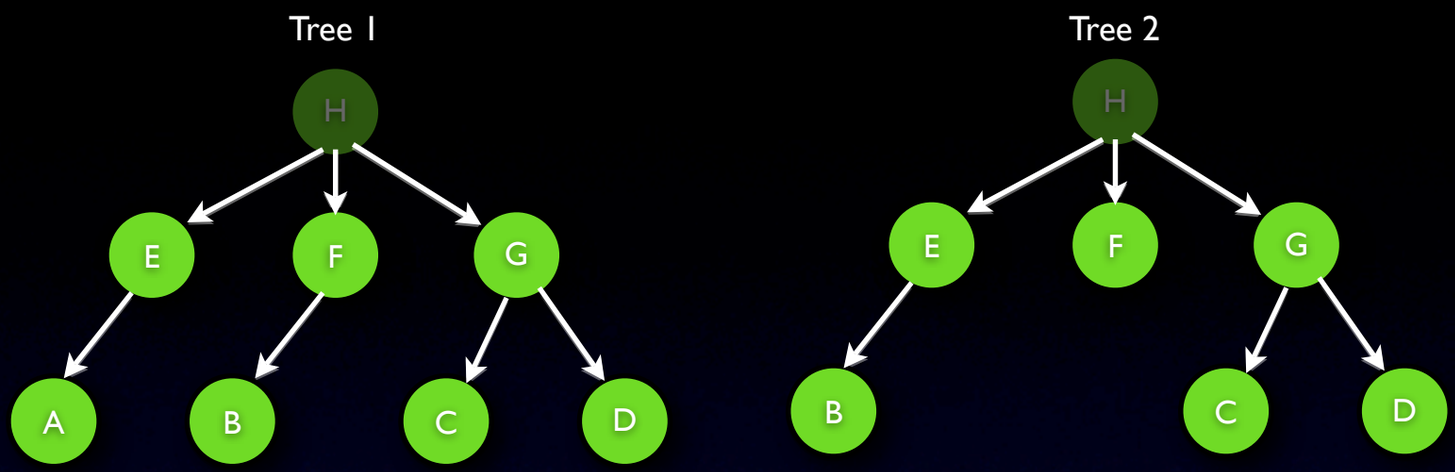
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

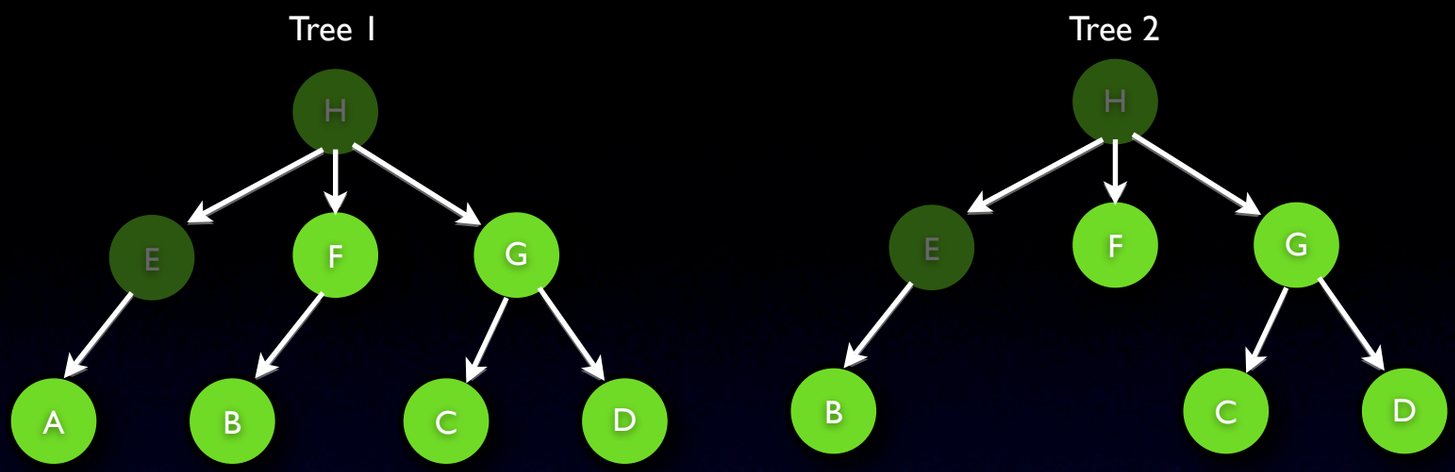
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

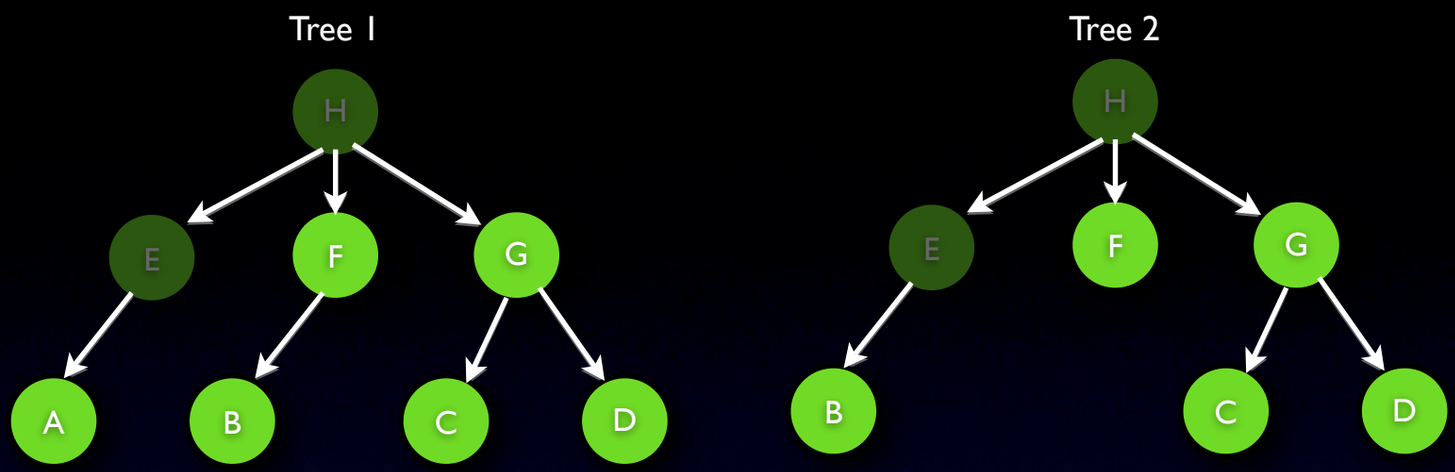
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

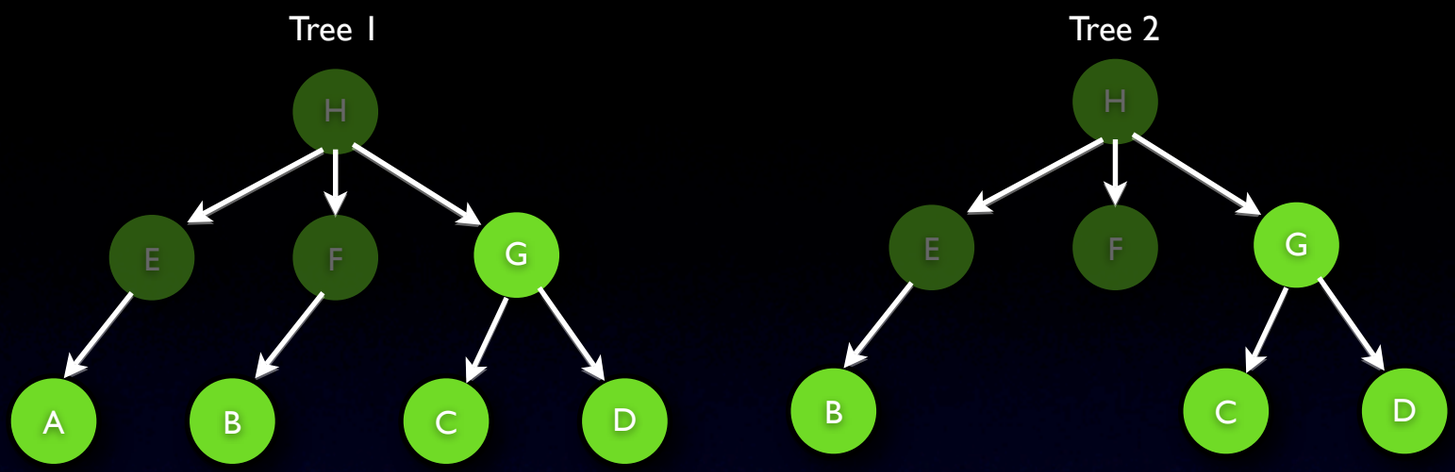
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

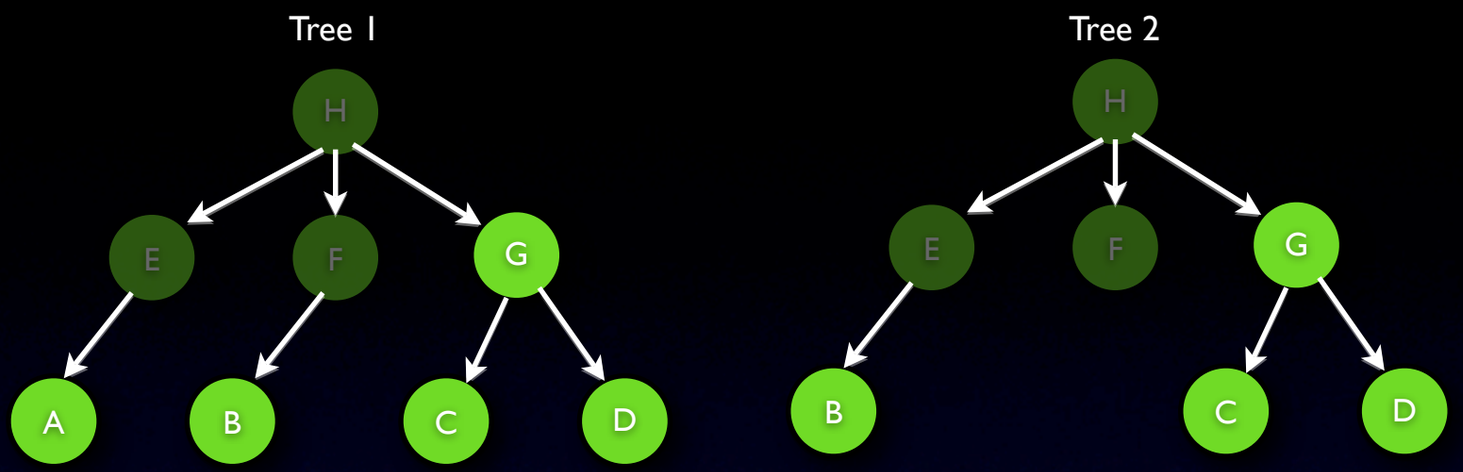
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

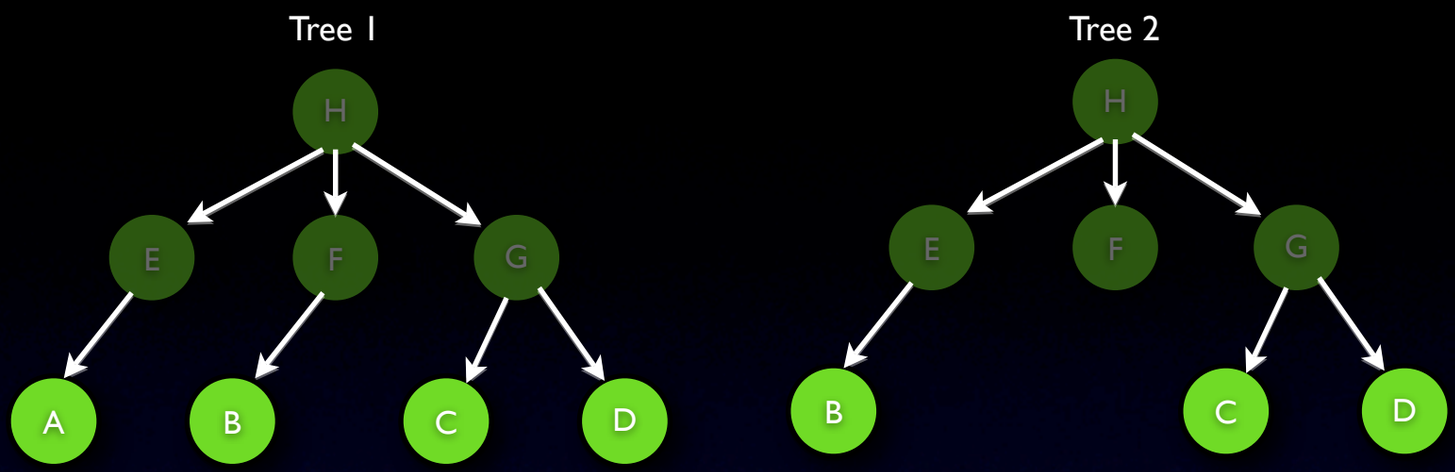
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

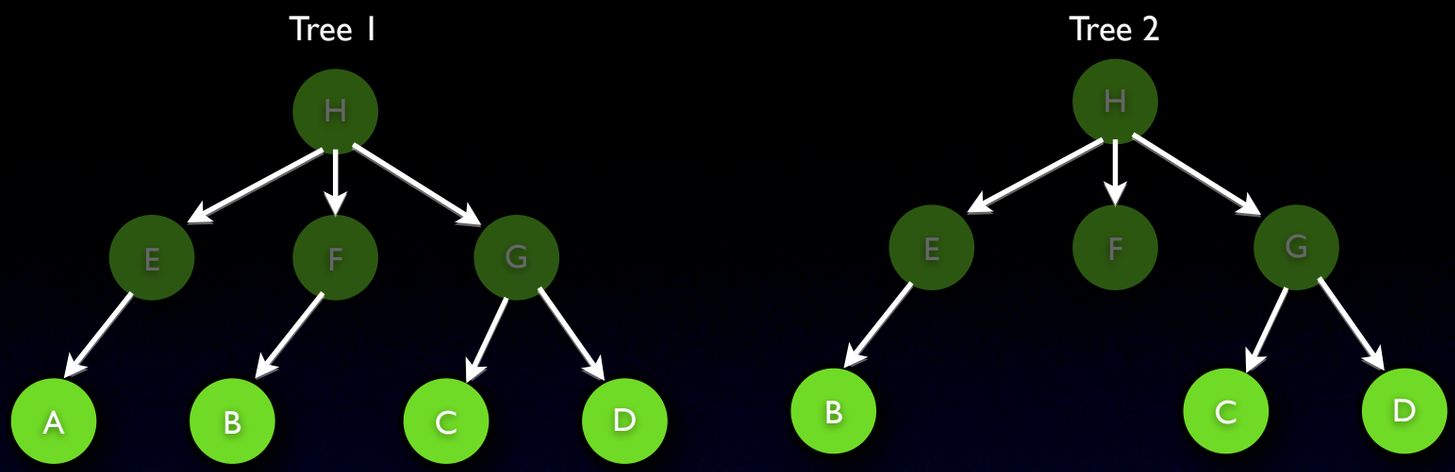
Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

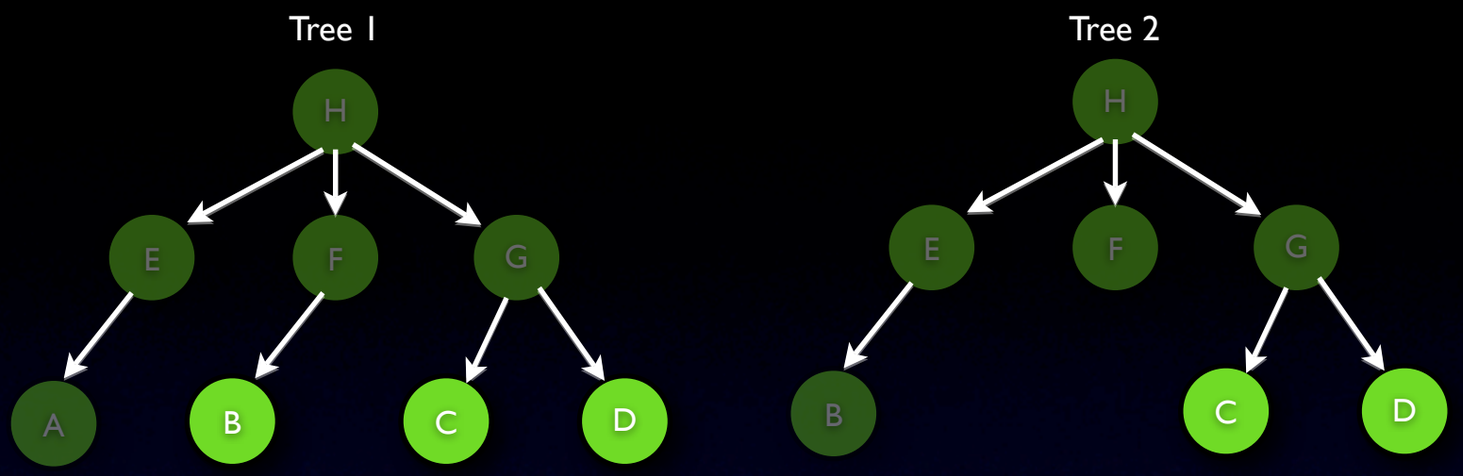
Creating the upgrade set



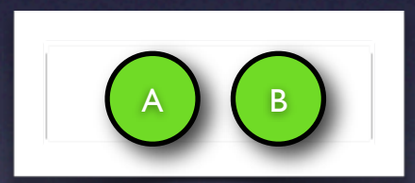
dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set



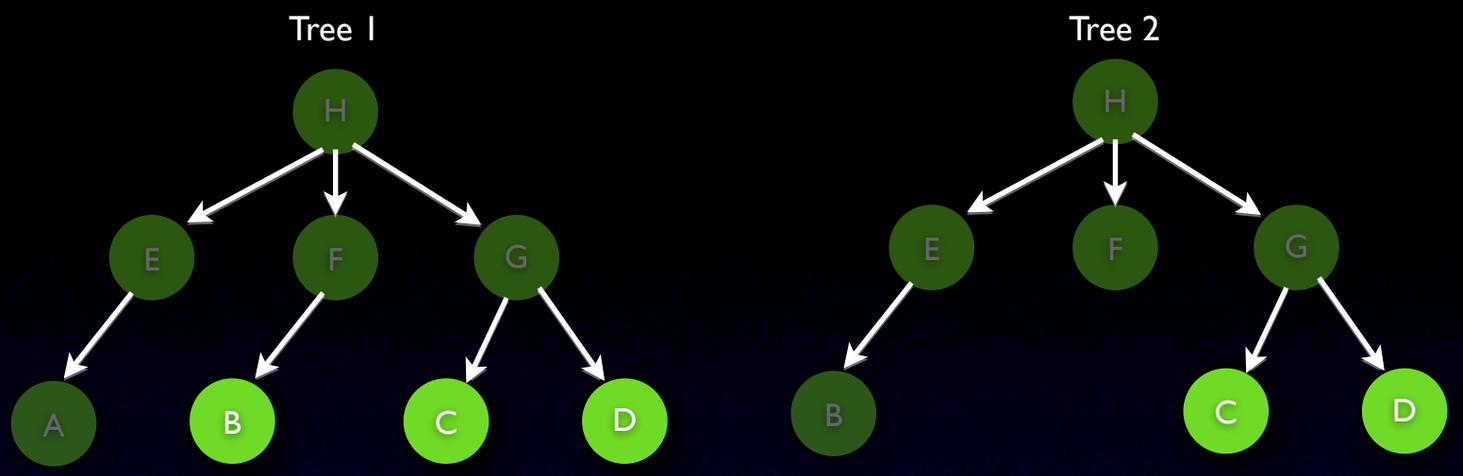
dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0



update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set

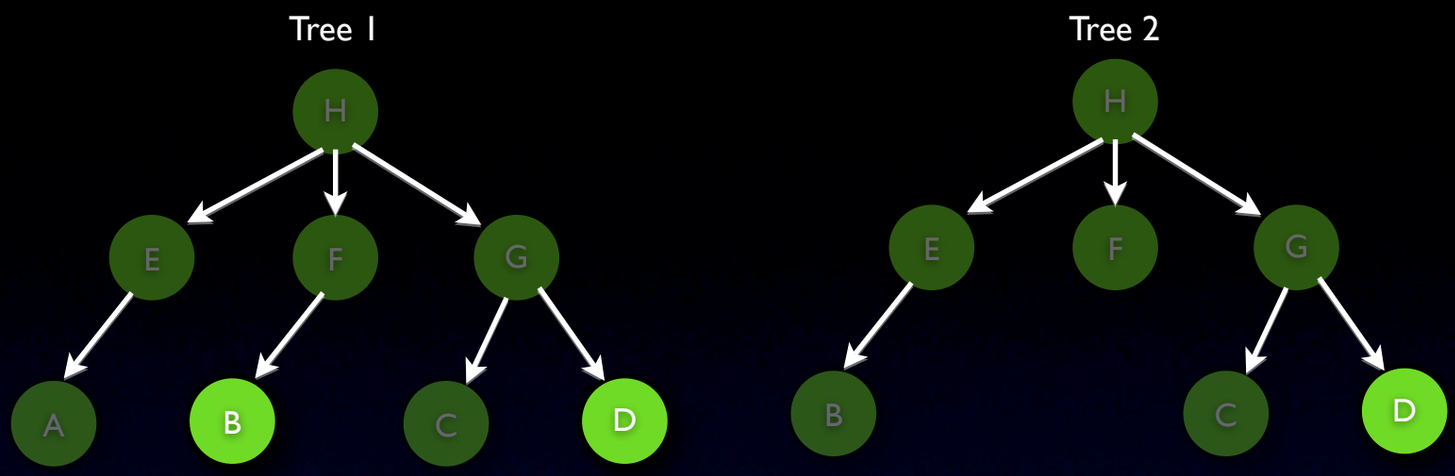


dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

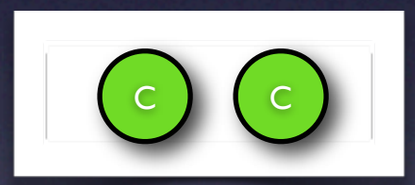
update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If (x, y) $\notin$ $M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair (x_i , y_j) $\in$ $M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i , Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set



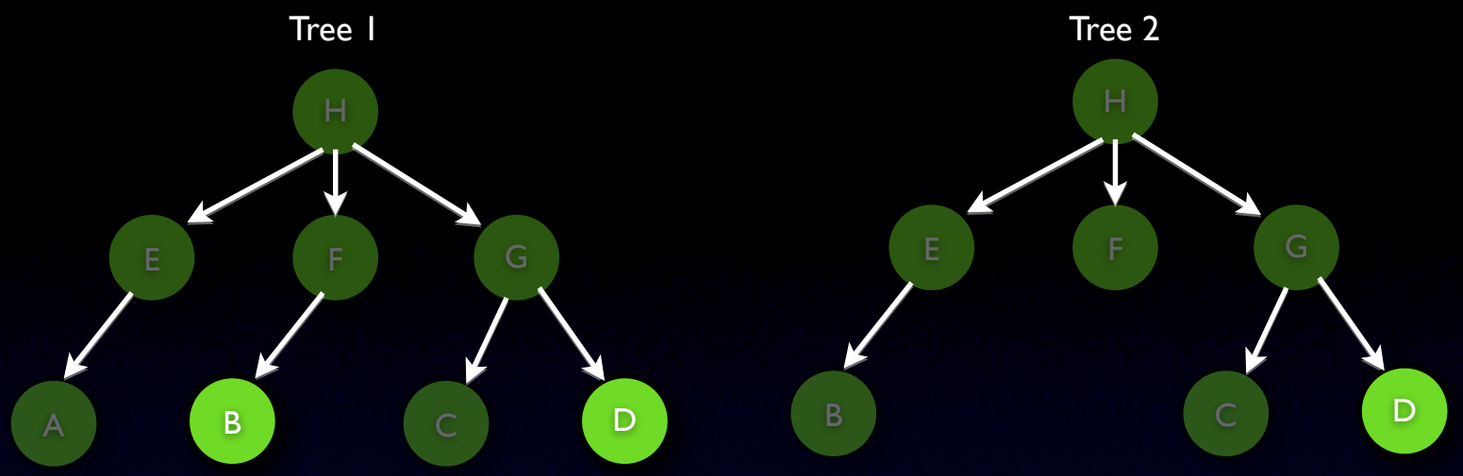
dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0



update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
$x = Root(T_1)$, $y = Root(T_2)$.
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if $Dist(T_1, T_2) = 0$
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If $Dist(x_i, y_j) = 0$
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set

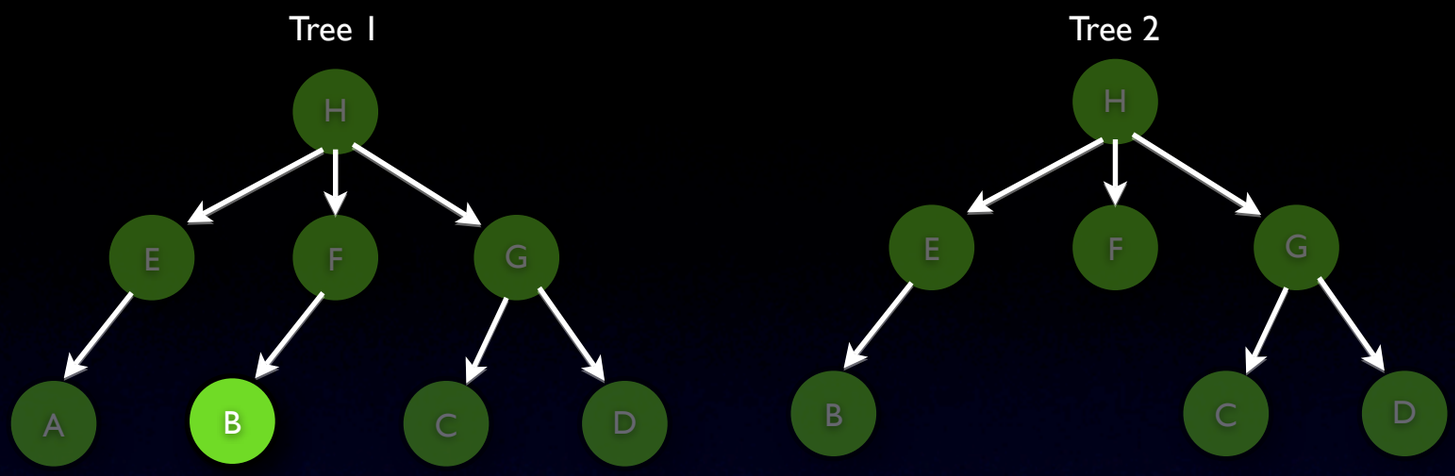


dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set

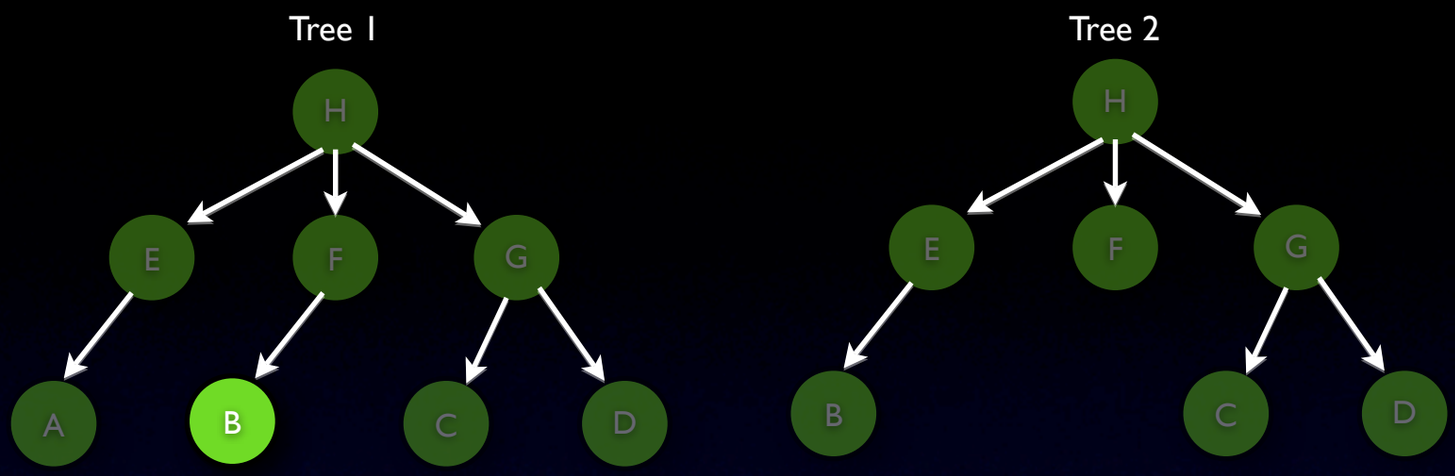


dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If (x, y) $\notin$ $M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair (x_i , y_j) $\in$ $M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i , Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set

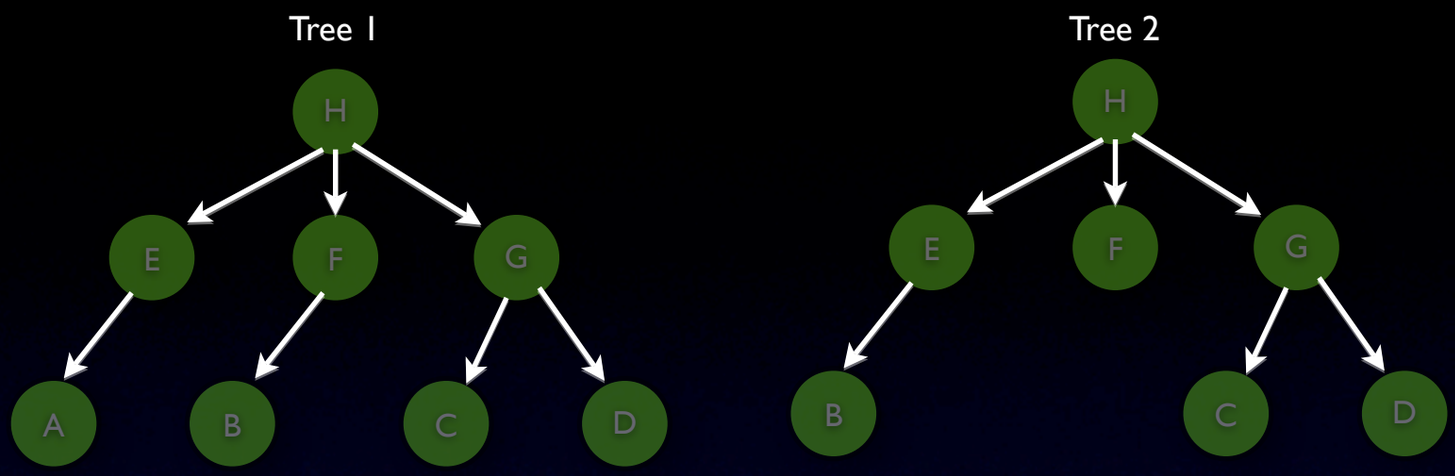


dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

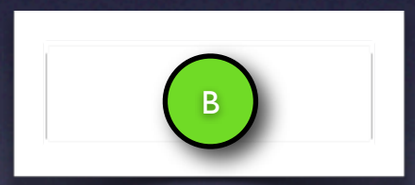
update(A,B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If (x, y) $\notin$ $M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair (x_i , y_j) $\in$ $M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i , Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set



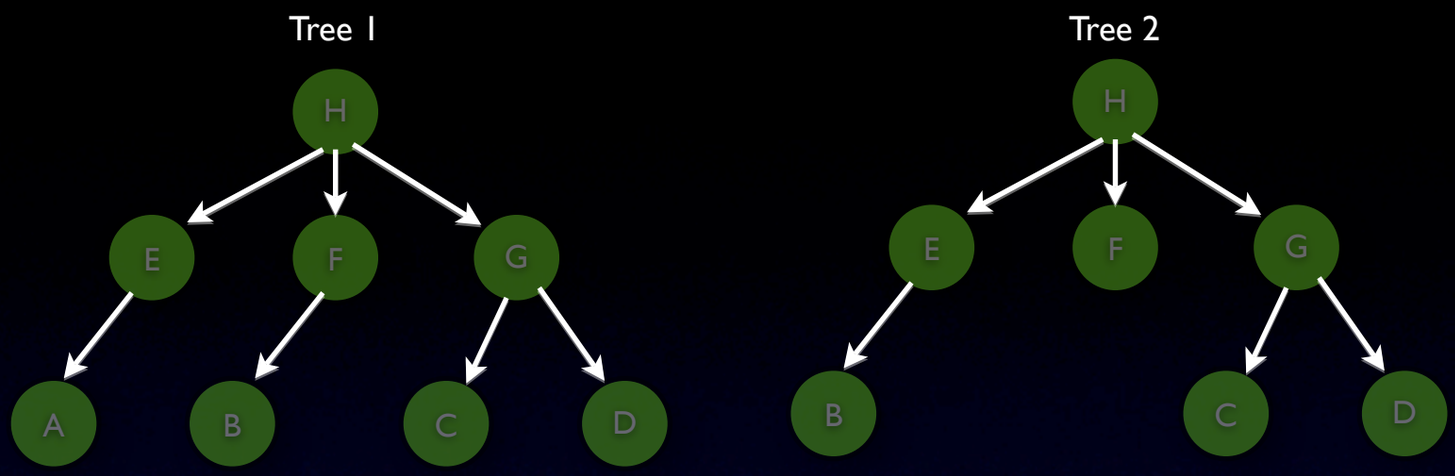
dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0



update(A,B)
delete(B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If (x, y) $\notin$ $M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair (x_i , y_j) $\in$ $M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i , Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

Creating the upgrade set



dist(A,B) = 1 dist(E,E) = 0 dist(H,H) = 0
dist(C,C) = 0 dist(F,F) = 0
dist(D,D) = 0 dist(G,G) = 0

update(A,B)
delete(B)

Input: Tree T_1 and T_2, a minimum-cost matching $M_{min}(T_1, T_2)$, the distance table DT.
Output: an edit script E.
Initialize: set E = Null;
x = Root(T_1), y = Root(T_2).
If $(x, y) \notin M_{min}(T_1, T_2)$ /* Subtree deletion and insertion */
  Return "Delete(T_1), Insert(T_2)".
Else if Dist(T_1, T_2) = 0
  Return "";
Else {
  For every node pair $(x_i, y_j) \in M_{min}(T_1, T_2)$, x_i is a child node of x, y_j is a child node of y.
  If x_i and y_j are leaf nodes
    If Dist(x_i, y_j) = 0
      Return "";
    Else /* Update leaf node */
      Add Update(x_i, Value(y_j)) to E;
  Else /* Call subtree matching */
    Add *EditScript*(Tx_i, Ty_j) to E;
    Return E;
For every node x_i not in $M_{min}(T_1, T_2)$
  Add "Delete(Tx_i)" to E;
For every node y_j not in $M_{min}(T_1, T_2)$
  Add "Insert(Ty_j)" to E;
Return E; }

X-Diff Conclusion

X-Diff Conclusion

- Since we can represent a model in XML we have with this algorithm a better way to compute differences

X-Diff Conclusion

- Since we can represent a model in XML we have with this algorithm a better way to compute differences
- However, in some cases additional model data has to be stored in the XML file

X-Diff Conclusion

- Since we can represent a model in XML we have with this algorithm a better way to compute differences
- However, in some cases additional model data has to be stored in the XML file
- We are in most of the cases not interested in all changes of the XML file, only the direct relevant ones from our model

X-Diff Conclusion

- Since we can represent a model in XML we have with this algorithm a better way to compute differences
- However, in some cases additional model data has to be stored in the XML file
- We are in most of the cases not interested in all changes of the XML file, only the direct relevant ones from our model
- The plain XML diff is not sufficient for that

Project Idea

A Metamodel Independent Approach to Difference Representation

Idea

A. Cicchetti, D.D. Ruscio and A. Pierantonio

"A Metamodel Independent Approach to Difference Representation"

University of Aquila, Italy (2007)

- In my project I would like to use the X-Diff algorithm in order to detect differences in models
- In order to refine the results I would like to propose a rule system which helps me specifying which differences are semantically relevant
- As a representation of this difference model I could use the information from the paper

Questions?

J. W. Hunt and M. D. McIlroy,

"An algorithm for differential file comparison"

Bell Telephone Laboratories CSTR #41 (1976)

Questions?

A. Cicchetti, D.D. Ruscio and A. Pierantonio

"A Metamodel Independent Approach to Difference Representation"

University of Aquila, Italy (2007)