Peer to Peer Collaborative Editing on XML-like trees

Martin Stéphane and Denis Lugiez
Laboratoire d'Informatique Fondamentale
39 rue F. Jolio-Curie
13013 Marseille, France
{stephane.martin, denis.lugiez}@lif.univ-mrs.fr

ABSTRACT

Peer to peer collaborative editing allows distinct sites to work concurrently on a common document without the need of a centralization process, contrary to classical systems like CVS or SVN that rely on this centralization to detect conflicts. In this so-called optimistic approach, we investigate how one can perform collaborative edition on a XML document like data-structure which is used in the Harmony framework [1]. It is well-known that convergence of the process (i.e. all participants agree on a final result) is ensured when two properties $TP1$ and $TP2$ are satisfied. In the word case (i.e. documents that are strings) no existing algorithm satisfy both properties [2] and we show that for the tree case, a first natural set of operations can't satisfy $TP1$ but that a very close set of operations satisfies $TP1$ and $TP2$. Therefore our proposed set of operations can be implementable in different algorithms like ADOPTED,....

Keywords

Peer to peer, Collaborative Editing, Optimistic reconciliation, XML-documents

1. INTRODUCTION

Cooperative editing of documents or synchronization systems like shared calendar become more and more common. The best known cooperative systems are CVS or Subversion (SVN) which both use locks : When a participant edits a file nobody else can modify this file at the same time. This requires a central server and it isn't failure resilient. Furthermore, this contradicts a Peer to Peer (P2P in short) approach where the set of participants is not fixed in advance. In this paper, we study collaborative editing in a P2P framework using an optimistic approach. In this framework, each participant works concurrently on the document and informs the other participants of each operation that he has performed. The communication is asynchronous and conflicts may arise because concurrent operations performed by different participants transform the documents in different ways that may be incompatible. In the optimistic approach, conflicts are solved by a transformational approach that replaces an operation on one site by another operation that combines the operation that this site has performed and the operation which has been done by another site. When the transformation satisfies two confluence properties called $TP1$ and $TP2$, the convergence of the process is ensured: all participants will eventually agree on the same final document (proved in [7, 4]).

Like many questions involving concurrent process working on a shared resource, the apparent simplicity of the problem is delusive. For instance, when the document is modeled by a string and the operations are simply add or delete a letter at some position, the algorithms devised and used so far in all current systems don't have both properties $TP1$ and $TP2$ [3] and more complex operations must be considered to ensure a weak notion of convergence [2]. In this paper, we study a more elaborated data-structure which is the unranked unordered notion of edge-labeled trees used in Harmony project [1]. In this data structure, contrary to XML documents, the same label can't occur more than once between siblings. This is a first step towards a complete treatment of XML documents which are the standard representations for data exchange and manipulation on the Web.
Unpublished work [5] propose an approach to deal with XML documents but they use the So6 Framework based on the SOCT4 [8] algorithm which use time-stamp delivered by central server.

We study two similar sets of operations for adding and deleting nodes in this structure differing only by the deletion of nodes. In the first set, a single node is deleted, but some reorganization of the tree may follow to keep the single label property. In the second set, deletion means that the entire subtree located at this node is deleted. We prove that the first set of operations can't satisfy the $TP1$ property whatever transformation is used, and that the second set satisfies both $TP1$ and $TP2$. Therefore, this set is a good candidate for P2P editing specially with ADOPTED[6] algorithm

2. FORMAL DEFINITIONS

Documents (or trees) are modeled by unranked, unordered, trees where two different siblings have different labels. Two other classical models are ordered unranked trees (i.e. XML documents) or unranked unordered trees allowing multiple occurrences of the same label for siblings. The first data structure contains the string case, therefore presents the same difficulties as this case: no existing algorithm fulfills the $TP1$ and $TP2$ properties. The second data structure forbids the canonical definition of paths required to access the nodes of the trees that are modified during the edition process.

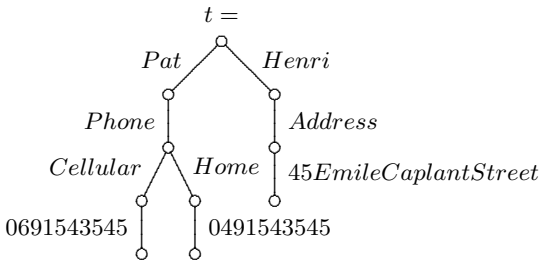
2.1 Tree Definition

Let Σ be a set of labels (names). Documents or trees are defined according to the grammar :

$$
\begin{array}{rll}
T ::= & \{\} & \text{// Empty tree} \\
| & \{n_1(T_1), ..., n_m(T_m)\}, & \text{// Set of tree } n_i(T_i) \\
& n_1, ..., n_m \in \Sigma,\ T_1, ..., T_m \in T, & \\
& \forall i, j \in [1..m] i \neq j \Rightarrow n_i \neq n_j. &
\end{array}
$$

As already mentioned, the definition ensures that two edges issued from the same node have different label: i.e. a given label occurs at most once on siblings. Trees are unordered i.e. $\{n_1(T_1), \ldots, n_m(T_m)\} = \{n_{\sigma(1)}(T_{\sigma(1)}), \ldots, n_{\sigma(m)}(T_{\sigma(m)})\}$ for any permutation σ.

Example :



$$
t = \left\{ \begin{array}{l} Pat\left(\left\{ \begin{array}{l} Phone\left(\left\{ \begin{array}{l} Home(\{0491543545(\{\})\}) \\ Cellular(\{0691543545(\{\})\}) \end{array} \right\}\right) \end{array} \right\}\right) \\ Henri(\{Address(\{45 \text{ Emile Caplant Street}(\{\})\})\}) \end{array} \right\}
$$

Union of Trees

We define $\oplus$ as follows : $T \oplus T \longmapsto T$

$$
. \oplus . = \left\{ \begin{array}{l} \{\} \oplus t = t \\ \\ t \oplus \{\} = t \\ \\ \{n_1(T_1), ..., n_i(T_i), ..., n_j(T_j), ..., n_m(T_m)\} \\ \oplus \\ \{n'_1(T'_1), ..., n_i(T'_k), ..., n_j(T'_g), ..., n'_o(T'_o)\} \\ = \\ \{n_1(T_1), ..., n_i(T_i \oplus T'_k), ..., n_j(T_j \oplus T_g), ... \\ , n'_1(T'_1), ..., n'_o(T'_o)\} \end{array} \right.
$$

2.2 Projection and Paths

We define $t_{|_n}$ the projection along n on t.

$$
\begin{array}{rcl}
\{n_1(T_1), ..., n_i(T_i), ..., n_m(T_m)\}_{|_{n_i}} & = & T_i \\
\{\}_{|_{n_i}} & = & \perp
\end{array}
$$

For exemple $t_{|_{Henri}} =$
$\{Address(\{45 \text{ Emile Caplant Street}(\{\})\})\}$.

A path is a sequence of names. We write ϵ for the empty path and $p.p'$ for the concatenation of paths p and p'. The set of paths is written $\mathcal{P}$. The projection of tree t along a path p, written $t_{|_p}$, is defined as follows:

$$
t_{|_p} = \left\{ \begin{array}{ll} t_{|_\epsilon} & = t \\ t_{|_{n.p}} & = t_{|_{n\,|_p}}, n \in \Sigma, p \in \mathcal{P} \end{array} \right.
$$

Domain

The function $Dom : T \longmapsto \mathbb{P}(\Sigma)$ is defined by:

$$
\begin{array}{rcl}
Dom(\{\}) & = & \emptyset \\
Dom(\{n_1(T_1), ..., n_m(T_m)\}) & = & \{n_1, ..., n_m\}
\end{array}
$$

Example :

$$
\begin{array}{rcl}
Dom(t) & = & \{Pat, Henri\} \\
Dom(t_{|_{Pat}|_{Phone}}) & = & \{Home, Cellular\}
\end{array}
$$

We write $p_1 \triangleleft p_2$, when a path p_1 is a prefix of another path p_2. It defines a partial order on paths.

2.3 Operations on Trees

Operations are functions $T \longmapsto T$ that modify trees and the set of operations is denoted by Op. Editing a document consists in performing a sequence of operations on a tree.

The following operations are used in this paper:

- $Add(p, n)$: Add a edge labeled n at end of path p.

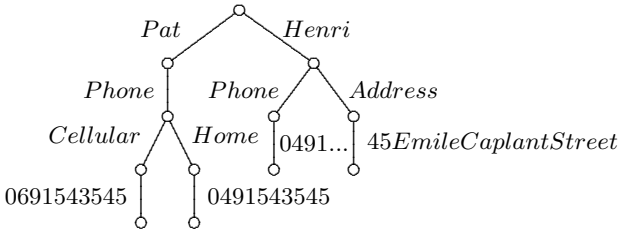
  $Add(n'.p, n)(\{n_1(t_1), ..., n_q(t_q)\}) =$
  $\{n_1(t_1), ..., n_q(t_q), n'(Add(p, n)(\{\}))\}$, if $n' \notin Dom(t)$

  $Add(n_i.p, n)(\{n_1(t_1), ..., n_i(t_i), ..., n_q(t_q)\}) =$
  $\{n_1(t_1), ..., n_i(Add(p, n)(t_i)), ..., n_q(t_q)\}$

  $Add(\epsilon, n)(t) =$
  t, if $n \in Dom(t)$

  $Add(\epsilon, n)(\{n_1(t_1), ..., n_q(t_q)\}) =$
  $\{n_1(t_1), ..., n_q(t_q), n(\{\})\}$

  Example : $t' = Add(Henri.Phone, 0491835469)(t)$

  And $Add(Henri, Phone)(t')$ does nothing because $Henri.Phone$ already exist.

- $Nop()$: Do nothing.

  $$Nop()(t) = t$$

Figure 1: Common operations

Operations to delete nodes are defined later on.

2.4 The *Do* Function

The function $Do : T \times Op \longmapsto T$ applies an operation $op \in Op$, on a tree t: $Do(t, op) = op(t)$

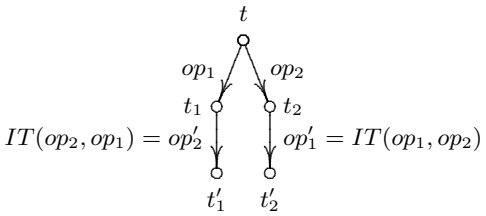
The following notation is used for a sequence of operations

$$(t)[op_1; op_2; ...; op_n] = Do(op_n, ..., Do(op_2, Do(op_1, t))...)$$

2.5 The IT Function for Collaborative Editing

Collaborative edition involves several participants (sites) that independently performs operations on a common document. In the P2P model, there is no centralization process that

enforces convergence (i.e. all participants agree on the same resulting document). The only communication involved is that each site informs the other sites of the operations that it performs. Therefore two sites 1 and 2 may intend to realize respectively op_1 and op_2, and inform the other site of the operation that has been done. The IT function is the transformation that a site computes to integrate an operation performed by another site with the last operation that he has done. More precisely $IT(op_2, op_1)$ is the operation that replaces op_2 on $site_1$ when op_2 was integrated previously by $site_2$ for competing operations op_1 and op_2. If no integration occurs, conflicts may happen: let us assume that $site_1$ deletes a node at some position when $site_2$ creates a node at the same position. If the initial tree doesn't contain a node at this position, the first site actually adds the note, then receives the notification that the second node has deleted this node, therefore the node is suppressed. Meanwhile, the initial deletion operation on $site_2$ does nothing, then it gets notification of the addition done by $site_1$ which results in the addition of the node: the trees on each site are in contradictory states. To avoid the conflict, operations must be integrated using IT, for instance one possible IT operation replaces the Add operation on $site_2$ by $Nop()$ which suppresses the conflict.



The classical property $TP1$ required on the IT function states that $t'_1 = t'_2$.

Remark: To design IT functions that ensures convergence is a difficult problem and often results in counter-intuitive definitions and results.

2.6 The $TP1$ and $TP2$ Properties

Convergence of collaborative editing is ensured [7, 4] when the IT function satisfies

- $TP1$ property: $(t)[op_1; IT(op_2, op_1)] = (t)[op_2; IT(op_1, op_2)]$ (i.e. $t'_1 = t'_2$ in previous figure)
- $TP2$ property: $IT(IT(op, op_1), IT(op_2, op_1)) = IT(IT(op, op_2), IT(op_1, op_2))$

In this paper we show that different choices of operations to delete node may or may not ensure that $TP1$ holds.

3. FIRST MODEL : DELETING A SINGLE NODE

3.1 Operations on Trees

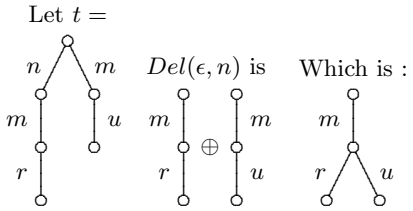
The first set of operations on trees that we use is:
$Op = \{Nop(), Add(p, n), Del(p, n)\}$ $p \in \mathcal{P}, n \in \Sigma$ with $Nop()$ and $Add(p, n)$ are as before (see figure 1).
$Del(p, n)$: Replace a edge labeled n at end of path p by the set of its successors.

$Del(n'.p, n)(t) = t$, if $n \notin Dom(t)$

$Del(n_i.p, n)(\{n_1(t_1), ..., n_i(t_i), ..., n_q(t_q)\}) = \{n_1(t_1), ..., n_i(Del(p, n)(t_i)), ..., n_q(t_q)\}$

$Del(\epsilon, n)(t) = t$, if $n \notin Dom(t)$

$Del(\epsilon, n_i)(\{n_1(t_1), ..., n_i(t_i), ..., n_q(t_q)\}) = \{n_1(t_1), ..., n_q(t_q)\} \oplus t_i$

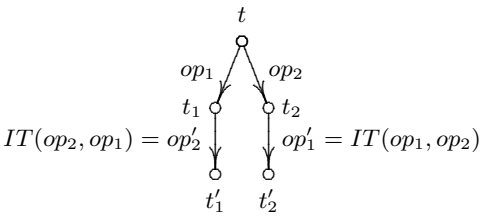


3.2 No IT Compatible with $TP1$ Exists

The following theorem states an negative result for the set Op.

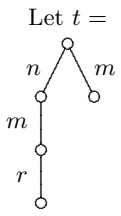
THEOREM 1. *There is no definition of $IT(op_1, op_2) \in Op$ such that IT satisfies $TP1$.*

PROOF. The function IT satisfies property $TP1$ iff $(t)[op_1; IT(op_2, op_1)] = (t)[op_2; IT(op_1, op_2)]$
i.e. $t'_1 = t'_2$ in the next picture



We prove that there is no such transformation IT (on the set of operation Op). The proof is a case analysis on the possible definition of IT.

Let $n, m, r \in \Sigma$ pairwise different element.



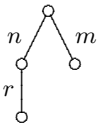
We choose op_1 as $Del(\epsilon, n)$ and op_2 as $Del(n, q)$
We want to perform $Del(\epsilon, n)$ concurrently with $Del(n, m)$
$TP1 : (t)[Del(\epsilon, n); IT(Del(n, m), Del(\epsilon, n))] = (t)[Del(n, m); IT(Del(\epsilon, n), Del(n, m))]$
We do the first operation of this sequence.

The first site execute $Del(\epsilon, n)$ and we have

$$t_1 = Del(\epsilon, n)(t) =$$

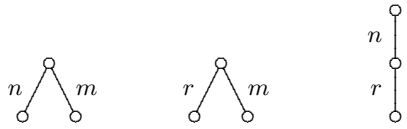
The second site execute $Del(n, m)$ and we obtain :

$$t_2 = Del(n, m)(t) =$$



Let as assume that function IT exists in this model. To have the property TP1,op'_1 and op'_2 must have operation of Op and $(t)[op_1; op'_2] = (t)[op_2; op'_1]$. We assume that $TP1$ hold and we prove that $IT(op_1, op_2)$ can't be defined on an operation of Op.

- $op'_2 = Nop()$
  - $op'_1 = Nop()$: Trivial because $t_1 \neq t_2$
  - $op'_1 = Add(\_, \_)$
    There is at least one more edge on t'_2.
  - $op'_1 = Del(x, y)$ we get :



$(x = n, y = r)_{or}(x = \epsilon, y = n)_{or}(x = \epsilon, y = m)$

    Any possible operation doesn't change t_2. In all case $t'_1 \neq t'_2$

- $op'_2 = Add(\_, \_)$
  - $op'_1 = Nop()$
    We have r under m on t_1 under n on t'_2.
  - $op'_1 = Add(\_, \_)$
    Number of edge on t'_1 and on t'_2 are different.
  - $op'_1 = Del(\_, \_)$ idem
- $op'_2 = Del(\_, \_)$

$t'_1 = m$  or r  or we return on $Nop()$ case.

  - $op'_1 = Nop()$: The number of node are different. Therefore $t'_1 \neq t'_2$
  - $op'_1 = Add(\_, \_)$ idem
  - $op'_1 = Del(\_, \_)$ idem

□

4. SECOND MODEL : DELETING A SUBTREE

4.1 A New Delete Operation

In this model, we replace the Del operation by a deletion operation that removes the entire subtree located at some position (hence no fusion is needed after deletion).
$Del(p, n)$:Delete a node labeled n at end of path p and delete his subtree.

$$Del(n'.p, n)(t) = t, \text{ if } n \notin Dom(t)$$

$$Del(n_i.p, n)(\{n_1(t_1), ..., n_i(t_i), ..., n_q(t_q)\}) = \{n_1(t_1), ..., n_i(Del(p, n)(t_i)), ..., n_q(t_q)\}$$

$$Del(\epsilon, n)(t) = t, \text{ if } n \notin Dom(t)$$

$$Del(\epsilon, n_i)(\{n_1(t_1), ..., n_i(t_i), ..., n_q(t_q)\}) = \{n_1(t_1), ..., n_q(t_q)\}$$

4.2 Function IT

We define the function IT as follows.
$IT(op_1, op_2) =$

$$\begin{cases} IT(Add(p, n), Add(p', n')) = Add(p, n), \\ IT(Add(p, n), Del(p', n')) = \begin{cases} Nop(), \text{ if } p = p' \wedge n = n' \\ Nop(), \text{ if } p'.n' \lhd p \\ Add(p, n), \text{ else.} \end{cases} \\ IT(Del(p, n), Add(p', n')) = Del(p, n) \\ IT(Del(p, n), Del(p', n')) = \begin{cases} Nop(), \text{ if } p = p' \wedge n = n' \\ Nop(), \text{ if } p'.n' \lhd p \\ Del(p, n), \text{ else.} \end{cases} \\ IT(op_1, Nop()) = op_1 \\ IT(Nop(), op_2) = Nop(); \end{cases}$$

4.3 The Main Result

The new set of operations ensures that collaborative editing is possible.

THEOREM 2. *IT satisfies $TP1$ and $TP2$.*

The proof proceeds as a case analysis (see appendix A.1 and A.2).

5. PERSPECTIVES

We aim at investigating several research directions. The first one is to add a typing information on documents like DTD for XML documents. Logics for typing documents that are unranked unordered trees have been investigated in [9] and can be used in this framework. Another research direction is to enrich the model of trees by adding strings at the leaves of the documents: for instance a TeX paper has a tree-like structure (section, subsections,. . .) where the actual text is a string that occurs at leaves positions. This model can be used both with ordered or unordered trees which are both meaningful depending on the applications.

6. REFERENCES

[1] J. Nathan Foster, Michael B. Greenwald, Christian Kirkegaard, Benjamin C. Pierce, and Alan Schmitt. Exploiting schemas in data synchronization. *Journal of Computer and System Sciences*, 2007. To appear.

Extended abstract in *Database Programming Languages (DBPL)* 2005.

[2] Abdessamad Imine. *Conception Formelle d'Algorithmes de Réplication Optimiste. Vers l'Edition Collaborative dans les Réseaux Pair-à-Pair*. PhD thesis, Université Henri Poincaré, Nancy, décembre 2006.

[3] Abdessamad Imine, Michaël Rusinowitch, Michaël Rusinowitch, Gérald Oster, Gérald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theor. Comput. Sci.*, 351(2):167–183, 2006.

[4] Brad Lushman and Gordon V. Cormack. Proof of correctness of ressel's adopted algorithm. *Inf. Process. Lett.*, 86(6):303–310, 2003.

[5] Gérald Oster, Hala Skaf-Molli andPalcal Molli, and Hala Naja-Jazzar. Supporting collaborative writing of xml documents. *unpublished*, 2007.

[6] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297, New York, NY, USA, 1996. ACM.

[7] Maher Suleiman, Michèle Cart, and Jean Ferrié. Concurrent operations in a distributed and mobile collaborative environment. In *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, pages 36–45, Washington, DC, USA, 1998. IEEE Computer Society.

[8] Nicolas Vidot, Michelle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 171–180, New York, NY, USA, 2000. ACM.

[9] S. Dal Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. In *Proceedings of Symposium on Principles of Programming Languages (POPL)*. ACM Press, 2004.

APPENDIX

A. PROOF

A.1 Proving TP1

$\forall op_1, op_2 \in Op, s \in State$,
$(t)[op_1; IT(op_2, op_1)] = (t)[op_2; IT(op_1, op_2)]$
We enumerate a different case :

1. $op_1 = Add(p,n)$ and $op_2 = Add(p',n')$
   $(t)[Add(p,n); IT(Add(p',n'), Add(p,n))] =$
   $(t)[Add(p,n); Add(p',n')]$

   $(t)[Add(p',n'); IT(Add(p,n), Add(p',n'))] =$
   $(t)[Add(p',n'); Add(p,n)]$ We will prove :
   $Do(Do(t, Add(p,n)), Add(p',n')) =$
   $Do(Do(t, Add(p',n')), Add(p,n))$.
   We will explore every cases.
   (a) Empty path :
   - If $n, n' \notin Dom(t)$ and $n \neq n'$
     $Add(\epsilon,n')(Add(\epsilon,n)(\{n_1(T_1), ..., n_q(T_q)\}))$
     $= \{n_1(T_1), ..., n_q(T_q), n(\{\}), n'(\{\})\}$
     $Add(\epsilon,n')(Add(\epsilon,n)(\{n_1(T_1), ..., n_q(T_q)\}))$
     $= \{n_1(T_1), ..., n_q(T_q), n'(\{\}), n(\{\})\}$
     Theses are equal.
   - If $n = n'$
     We obtain : $= \{n_1(T_1), ..., n_q(T_q), n(\{\})\}$
     Because we use the third choice of function $Add(\epsilon,n)(t)$ and first operation add $n(\{\})$.
   - If $n \in Dom(t)$
     We have $\{n_1(T_1), ..., n_q(T_q), n'(\{\})\}$ Third we use the second choice
   - idem if $n' \in Dom(t)$ with n.
   - If $n, n' \in Dom(t)$ the tree is unchanged.

   (b) if $p.n \lhd p' : \exists p'', p.n.p'' = p$ if $n \in dom(p)$ then $Add(p,n)$ do nothing.
   else We have $t_{|_p} = \{n_1(T_1), ..., m_1(T'_1), ...n_q(T_q)\}$
   $^{(1)} = \{n_1(T_1), ..., m_1(T'_1), ...n_q(T_q), n(\{\})\}$
   $\{n_1(T_1), ..., m_1(T'_1), ...n_q(T_q), n(Add(p'',n')(\{\})\}$
   By definition :
   $Add(p',n')(t) = \{n_1(T_1), ..., m_1(T'_1), ...,$
   $n_q(T_q, n(Add(p'',n')(\{\})\}$ therefor $n \in dom(t_{|_p})$ and $Add(p,n)$ do nothing.
   so $^{(2)} = \{n_1(T_1), ..., m_1(T'_1), ...n_q(T_q), n(Add(p'',n')(\{\})\}$

   (c) idem for $p'.n' \lhd p$

   (d) if $p \lhd p'$ We have : $p' = p.m_1.p_1$
   We have $t_{|_p} = \{n_1(T_1), ..., m_1(T'_1), ..., n_{q-1}(T_{q-1})\}$
   it's given in two cases, by recurrence definition :
   ${t'}_{|_p} =$
   $\{n_1(T_1), ..., m_1(Add(p'_1,n')(T'_1)), ..., n_{q-1}(T_{q-1}), n(\{\})\}$

   (e) idem for $p' \lhd p$

   (f) p, p' not empty (general case) $\exists p \in \mathcal{P} | p = p_{comon}.p'_1$ and $p' = p_{comon}.p'_2$ We have two path not empty then :
   $p'_1 = m_1.p''_1$ and $p'_2 = m_2.p''_2$
   We have
   $t_{|_p} = \{n_1(T_1), ..., m_1(T'_1), ..., m_2(T'_2), ...n_{q-2}(T_{q-2})\}$
   We have by definition :
   ${t'}_{|_p} = \{n_1(T_1), ..., m_1(Add(p''_1,n)(T'_1), ...$
   $, m_2, (Add(p''_2,n')(T_2)), ..., n_{q-2}(T_{q-2})\}$

2. $op_1 = Add(p,n)$ and $op_2 = Del(p',n')$
   $(t)[Add(p,n); IT(Del(p',n'), Add(p,n))]^{(1)}$
   $(t)[Del(p',n'); IT(Add(p,n), Del(p',n'))]^{(2)}$
   - $p = p'$ and $n = n'$
     $^{(1)} = (t)[Add(p,n), Del(p,n)]$
     $^{(2)} = (t)[Del(p,n; Nop()]$
     - if $n \in Dom(p)$ then $Add(p,n)(t)$ do nothing. Therefore $^{(1)} =^{(2)}$
     - if $n \notin Dom(p)$ then $Add(p,n)(t)$ create a node who delete by $Del(n,p)$ in $^{(1)}$. and $Del(n,p)$ do nothing in $^{(2)}$
       Therefore $^{(1)} =^{(2)}$
   - $p'.n' \lhd p$
     $^{(1)} = (t)[Add(p,n); Del(p',n')]$
     $^{(2)} = (t)[Del(p',n'); Nop()]$
     if p=p'.n'.p"
     We take : $t_{|_{p'}} = \{n_1(T_1), ..., n'(T), ..., n_{q-1}(T_{q-1})\}$
     We have :
     $^{(1)}{}_{|_p} = \{n_1(T_1), ..., n'(Add(p'',n)(T)), ..., n_{q-2}(T_{q-2})\}$
     $= \{n_1(T_1), ..., n_{q-2}(T_{q-2})\}$
     $^{(2)}{}_{|_p} = \{n_1(T_1), ..., n_{q-2}(T_{q-2})\}$
   - else : same demo of 1f.

3. idem for $op_1 = Del(p,n)$ and $op_2 = Add(p',n')$
4. $op_1 = Del(p,n)$ and $op_2 = Del(p',n')$
   $(t)[Del(p,n); IT(Del(p',n'), Del(p,n))]^{(1)}$
   $(t)[Del(p',n'); IT(Del(p,n), Del(p',n'))]^{(2)}$
   - p.n=p'.n' :
     $^{(1)} = (t)[Del(p,n), Nop()]$
     $^{(2)} = (t)[Del(p,n), Nop()]$
   - $p.n \lhd p'$
     We have p'=p.n.p";
     We take $t_{|_p} = \{n_1(T_1), ..., n(T), ..., n_{q-1}(T_{q-1})\}$
     $^{(1)} = (t)[Del(p,n); Nop()]$
     $^{(2)} = (t)[Del(p',n'); Del(p,n)]$
     $^{(1)}{}_{|_p} = \{n_1(T_1), ..., n_{q-1}(T_{q-1})\}$
     first time : $Del(p,n)(t)_{|_p}\{n_1(T_1), ...,$
     $n(Del(p'',n')(T)), ..., n_{q-1}(T_{q-1})\}$
     therefor $^{(2)}{}_{|_p} = \{n_1(T_1), ..., n_{q-1}(T_{q-1})\}$
   - idem for $p'.n' \lhd p'$
   - else: same 1f we have two independant subtree.
5. case Nop() is trivial. □

A.2 Proving TP2

$IT(IT(Op, Op_1), IT(Op_2, Op_1))^{(1)} =$
$IT(IT(Op, Op_2), IT(Op_1, Op_2))^{(2)}$
We will explore every case :

- $Op = Add(p,n), Op_1 = Add(p_1,n_1)$ and $Op_2 = Add(p_2,n_2)$
  therefore $^{(1)} = Add(p,n)$ and $^{(2)} = Add(p,n)$
- $Op = Add(p,n), Op_1 = Add(p_1,n_1)$ and $Op_2 = Del(p_2,n_2)$
  $IT(IT(Add(p,n), Add(p_1,n_1)), IT(Del(p_2,n_2), Add(p_1,n_1)))^{(1)}$
  $IT^{(b)}(IT(Add(p,n), Del(p_2,n_2)),$
  $IT^{(a)}(Add(p_1,n_1), Del(p_2,n_2)))^{(2)}$
  $^{(1)} = IT(Add(p,n), Del(p_2,n_2))$
  $^{(2)} = IT^{(b)}(IT(Add(p,n), Del(p_2,n_2)), Add(X,n_1)) =$
  $IT(Add(p,n), Del(p_2,n_2))$
  or $^{(2)} = IT^{(b)}(IT(Add(p,n), Del(p_2,n_2)), Nop()) =$
  $IT(Add(p,n), Del(p_2,n_2))$
  Because $^{(a)}$ give a Add() or a Nop() the second argument of $^{(b)}$ is a Add or a Nop.
- Idem for $Op = Add(p,n), Op_1 = Del(p_1,n_1)$ and $Op_2 = Add(p_2,n_2)$
- $Op = Add(p,n), Op_1 = Del(p_1,n_1)$ and $Op_2 = Del(p_2,n_2)$
  $IT(IT(Add(p,n), Del(p_1,n_1)), IT(Del(p_2,n_2), Del(p_1,n_1)))^{(1)}$
  $IT(IT(Add(p,n), Del(p_2,n_2)), IT(Del(p_1,n_1), Del(p_2,n_2)))^{(2)}$
  - If $p_1 = p_2$ and $n_1 = n_2$
    $^{(1)} = IT(IT(Add(p,n), Del(p_1,n_1)), Nop())$
    $^{(2)} = IT(IT(Add(p,n), Del(p_1,n_1)), Nop())$
  - If $p_2.n_2 \lhd p_1$
    $^{(1)} = IT(IT(Add(p,n), Del(p_1,n_1)), Del(p_2,n_2)))$
    because $p_1.n_1 \neq p_2.n_2$
    $^{(2)} = IT(IT(Add(p,n), Del(p_2,n_2)), Nop())$
    * if $p_2.n_2 \lhd p$
      $^{(1)} = IT(IT(Add(p,n), Del(p_1,n_1)), Del(p_2,n_2)))$
      $^{(2)} = Nop()$
      · if $p_1.n_1 \lhd p$
        $^{(1)} = Nop()$
        $^{(2)} = Nop()$
      · else :
        $^{(1)} = IT(Add(p,n), Del(p_2,n_2)) = Nop()$
        $^{(2)} = Nop()$
    * idem for $p_1.n_1 \lhd p$
    * else :
      $^{(1)} = IT(Add(p,n), Del(p_2,n_2)) = Add(p,n)$
      because
      $p_2.n_2 \not\lhd p \wedge p_1.n_1 \not\lhd p$
      $^{(2)} = IT(IT(Add(p,n), Del(p_2,n_2)), Nop())) =^{(1)}$
  - idem if $p_1.n_1 \lhd p_2$
  - Else :
    $^{(1)} = IT(IT(Add(p,n), Del(p_1,n_1)), Del(p_2,n_2)))$
    $^{(2)} = IT(IT(Add(p,n), Del(p_2,n_2)), Del(p_1,n_1)))$
    * if $p = p_1 \wedge n = n_1$
      $^{(1)} = IT(Nop(), Del(p_2,n_2))) = Nop()$
      $^{(2)} = IT(IT(Add(p_1,n_1), Del(p_2,n_2)), Del(p_1,n_1)))$
      We deduct by hypotheses :
      $^{(2)} = IT(Add(p_1,n_1), Del(p_1,n_1))) = Nop()$
    * idem if $p = p_2 \wedge n = n_2$
    * if $p_1.n_1 \lhd p$ $^{(1)} = IT(Nop(), Del(p_2,n_2)))$
      $^{(2)} = IT(IT(Add(p,n), Del(p_2,n_2)), Del(p_1,n_1)))$
      We know $p_2 \neq p \vee n_2 \neq n$ and $p_2.n_2 \not\lhd p$ because
      $p_2.n_2 \lhd p \wedge p_1.n_1 \lhd p \Rightarrow p_1.n_1 \lhd p_2.n_2 \vee p_2.n_2 \lhd p_1.n_1$
      $^{(2)} = IT(Add(p,n), Del(p_1,n_1))) = Nop() =^{(1)}$
    * idem if $p_2.n_2 \lhd p$
    * else :
      $^{(1)} = IT(Add(p,n))$
      $^{(2)} = IT(Add(p,n))$
- Trivial for $Op = Del(p,n), Op_1 = Add(p_1,n_1)$ and $Op_2 = Add(p_2,n_2)$
- if $Op = Del(p,n), Op_1 = Del(p_1,n_1)$ and $Op_2 = Add(p_2,n_2)$
  $^{(1)} = IT(IT(Del(p,n), Del(p_1,n_1)),$
  $IT(Add(p_2,n_2), Del(p_1,n_1))$
  $^{(1)} = IT(Del(p,n), Del(p_1,n_1))$ because the first argument will be a 'Del' and the second will be a 'Add'.
  $^{(2)} = IT(IT(Del(p,n), Add(p_2,n_2)),$
  $IT(Del(p_1,n_1), Add(p_2,n_2))$
  $^{(2)} = IT(Del(p,n), Dell(p_1,n_1))$
- idem for $Op = Del(p,n), Op_1 = Add(p_1,n_1)$ and $Op_2 = Del(p_2,n_2)$
- if $Op = Del(p,n), Op_1 = Del(p_1,n_1)$ and $Op_2 = Del(p_2,n_2)$
- Trivial If $Op = Nop()$
- if $Op = X(n,p), Op_1 = Nop()$ and $Op_2 = X'(n_2,p_2)$
  $^{(1)} = IT(IT(X(p,n), Nop()), IT(X'(p_2,n_2), Nop())$
  $= IT(X(p,n), X'(p_2,n_2))$
  $^{(2)} = IT(IT(X(p,n), X'(p_2,n_2)), IT(Nop(), X'(p_2,n_2))$
  $= IT(X(p,n), X'(p_2,n_2))$
- idem $Op = X(n,p), Op_1 = X(p_1,n_1)$ and $Op_2 = Nop()$
- Trivial, if $Op = X(n,p), Op_1 = Nop()$ and $Op_2 = Nop()$

□