International Journal of Control and Automation
Vol. 9, No. 5 (2016), pp. 331-340
http://dx.doi.org/10.14257/ijca.2016.9.5.32

The π-calculus-based Algorithm in Concurrency Comparison

Hao Bu[123], Shihong Chen[13*], Rong Zhu[123] and Xiaoqiong Tan[123]

[1]*Computer School of Wuhan University, China*
[2]*Collaborative Innovation Center of Geospatial Technology, China*
[3]*National Engineering Research Center for Multimedia Software, Computer School of Wuhan University, China*
chen_lei0605@sina.com

Abstract

Targeting at the features of concurrency and mobility, π calculus nowadays could serve as a highly effective tool of modeling and evaluating the software system. After a brief introduction of π calculus, the paper mainly discusses a comparison algorithm based on π calculus which is used frequently in software modeling. Since there are two different ways of data storage: the random storage and the linked list storage of elements, the paper analyzes both, with the special emphasis on the algorithm in the comparison carried on by any two elements.

Keywords*: π calculus; well-ordered set; comparison*

1. Introduction

As is known to all, the first step to solve a problem with the help of the computer is an unambiguous formal description of the problem. However, the traditional functional theory is no longer suitable to deal with this kind of description carried on in the system of the modern software, due to the new features of concurrency and mobility that exist in the system. So in 1990s, Prof. Robin Miller (Turing Prize winner) and his collaborator based their ideas on the Calculus of Communicating System (CCS) to form π calculus, a new concurrency theory taking the moving communication between processes as the research subject and supporting the modeling and synchronic testing of the dynamic system. On the one hand, π calculus is endowed with very simple mathematical structures with great expressing ability, because π calculus deals with all the things like the values, variables, parameters and the channels used for data transmission --- all without any distinction but by names, the most fundamental concepts in π calculus. On the other hand, π calculus also owns the special features appropriate for system description: concurrency and mobility. Therefore, in comparison with the functional theory, the merits of π calculus are considerable in describing a complex concurrent moving system at present.

In dealing with the comparison, the most basic calculation that frequently occurs in the system, π calculus itself does not provide the comparing operators though. According to different ways of element storage, this paper discusses the comparing calculation between any two elements in a well-ordered set, in order to solve the comparison problems based on π calculus in the actual system modeling.

2. A Brief Introduction of π-calculus

2.1. Interaction

In π calculus, representing the unit of the concurrent entities, each process has one or several channels communicating with other processes. Since data and

ISSN: 2005-4297 IJCA

channels used for the data transmission do not make any distinction but unanimously defined by names, processes and names are two basic entities in π calculus. Under these rules, the interaction in π calculus is actually the communication between two processes or the communication of two system data by using the channel.

Still, the interaction also concerns the division of π calculus into monadic π calculus and polyadic π calculus. The former limits the number of the input or the output to be only one, while the latter allows the number to be either none or many. Similar changes in π calculus are some extensions from its original definition to its promoted versions like the high order π calculus and asynchronous in π calculus *etc*., but the essential ideas in π calculus change little.

The basic interaction in π calculus is presented in Figure 1:

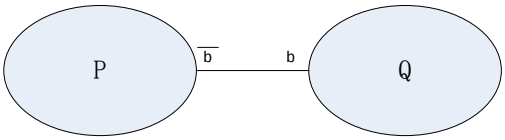

Figure 1. The Basic Interaction in Π Calculus

In Figure 1, the process P goes through the port b to send messages along the channel, through which the process Q receives it.

The concurrent interaction in π calculus is presented in Figure 2:

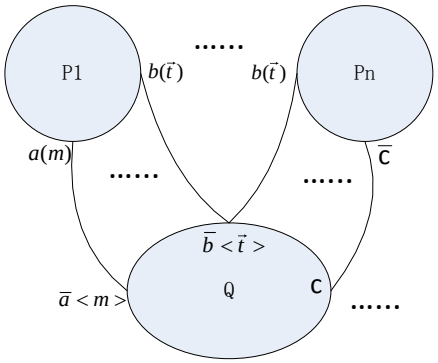

Figure 2. The Concurrent Interaction in Π Calculus

In Figure 2, the process Q goes through the port a to send messages m to P1 who could receive it through the port a. The process Q can send the message sequence $\vec{t}$ to the process P1 and Pn through the port b. Vice versa, the process P1 and Pn both can receive the message sequence $\vec{t}$ from the process Q through the port b. The process Pn can also send empty messages to the process Q through the port c. In the Figure, "....." means the omitted process(es) or channel(s).

2.2. π -Calculus Syntax

In π-calculus, sending or receiving a message (or a name) or making a silent transition could be represented by action prefixes. Its syntax and corresponding meanings are [1, 2, 3]:

π ::= m(n) receive the limited name n from the channel m.

m($\vec{n}$) receive the limited name sequence $\vec{n}$ from the channel m (Notes: the number of the receiver and the sender must be the same) .

$\overline{m}$ <n> send the free name n from the channel m.

$\overline{m} < \vec{n} >$ send the free name sequence $\vec{n}$ from the channel m (Notes: the number of the sender and the receiver must be the same).

τ the internal action within the process, invisible from the outside.

The process P in π-calculus can be syntactically defined as:

$$P ::= \sum_{i \in I} \pi_i . P_i \mid P_1 \mid P_2 \mid new\ n\ P \mid !P$$

$\sum_{i \in I} \pi_i . P_i$ is termed as a sum of the processes, in which I is the finite subscript set.

In the sum of the processes, π is guarded by π_i. That is, π has to start its action right after the action represented by π_i is finished.

The p1|p2 means the concurrent operation of the process P_1 and the process P_2.

The new n P means that the name n included in the process P is highly restricted or limited. Different from any other name outside P, this new n is quite unique even if it is still called n. For example,

$P = new\ a(a.Q + b.R) \mid \overline{a}.0) \mid (\overline{b}.S + \overline{a}.T)$ could lead to the two following deductions.

$P \to new\ a(Q) \mid (\overline{b}.S + \overline{a}.T)$

$P \to new\ a(R) \mid \overline{a}.0 \mid (S)$

But the following deduction is wrong.

$P \not\to new\ a(Q) \mid \overline{a}.0) \mid (T)$

!P means the concurrent operation of any multiple duplicates of P.

Similar to the sum of processes, $\prod_{i \in I} P_i$ could be used to mean the concurrent execution of multiple processes in order to achieve the conciseness of expressions, so the syntactic expression could be presented as:

$$P ::= \sum_{i \in I} \pi_i . P_i \mid \prod_{i \in I} P_i \mid new\ n\ P \mid !P$$

3. The Comparison of Any Two Elements in the Well-Ordered Set by Using Random Storage of Π Calculus

In the well-ordered set, when two elements are randomly selected for sequence comparison, the information on the order of the two elements requires different algorithms according to different ways of storages. In the pattern of random storage, every element saves its own ordinal information with other elements[4]. For example, it is supposed that there are [a1. .an] in a set. Without loss of generality, suppose that the ordinal relation between elements happens to be consistent with the sequence of element subscripts, that is, suppose k<h, then $A_K < A_h$. If a_m is taken as the example, then we can express its storage like the following:

$$A_m = \sum_{i=1}^{k-1} a_i . greater + a_m . equal + \sum_{j=k+1}^{n} a_j . less$$

Then, to compare a_m and a_n, we can use $A_m \mid \overline{a_m}$. If the result is "greater", it indicates a_m lies before a_n. If the result is "less", then it indicates a_m lies after a_n.

4. The Comparison of Any Two Elements in the Well-Ordered Set by Using Linked-List Storage of Π Calculus

When the well-ordered set is stored in the structure of the linked list, the algorithm in the comparison of two elements is relatively complicated.
The linked list is illustrated as in Figure 3:

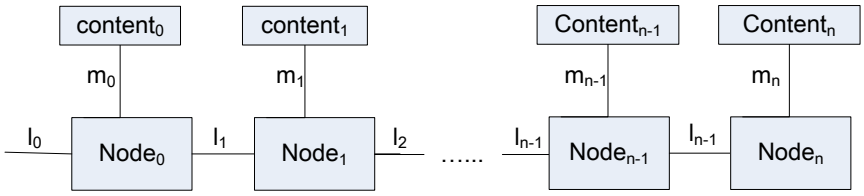

Figure 3. The Illustration of the Linked List

Suppose there are two elements that need comparing, the general plan is first to compare one of the two elements with the element in the linked list. When the first element finds a match of the corresponding element in the list, then the second element starts from where the first match is made to both directions for the match: towards the beginning and towards the end of the list. When the second element finds its match at the beginning direction of the list, it indicates that the second element is located before the first one. On the contrary, if the match is found at the end direction of the list, it indicates that the second element is located after the first element.
When we make the comparison, six processes are needed.

The first is the well-ordered process "welord", which serves as the standard for the comparison of the two elements.

The second is the initial process "launch", whose function is to initiate the comparison of the elements.

The third is the comparing process P1, which carries the first comparing element a_s。

The fourth is the comparing process P2, which carries the second comparing element a_j for one direction.

The fifth is the comparing process P3, which also carries the second comparing element a_j for the other direction.

The sixth is the result process "RESULT", which shows the result of the comparison.

welord=

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\,|\,!m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i\,|\,\overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

The main names and their related functions:

$l_i(c_{i+1})$: l_i is the node pointer. When i=0, that is, when l_0 is the initial pointer, the channel l_i can be used to receive C_{i+1} from the comparing process.

$\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})$: The C_{i+1} received is used as the channel to send to the comparing process $m_{i+1}\ l_{i-1}\ l_{i+1}$, in which m_{i+1} is responsible for receiving data from the comparing process. l_{i-1} is the pointer towards the beginning of the linked list, while l_{i+1} is the one towards the end of it.

$m_{i+1}(h_{i+1}\ re)$: m_{i+1} serves as the channel to receive h_{i+1} and re from the comparing process. In the format, h_{i+1} is mainly responsible for sending a_i or a_s; re is responsible for receiving the location information from the comparing process; when re=loc, it indicates the location at the moment; when re=back, it indicates the second element lies after the

first element a_i; when re=front, it indicates the second element lies before the first element a_i.

$\sum_{j=1}^{3}\overline{switch_j}$ functions as giving a notice to the comparing process of carrying on the next comparison after it receives the data sent from the comparing process. Since the major function of $\sum_{j=1}^{3}\overline{switch_j}$ is the control of the on/off switch, the name it sends will be omitted in the expression.

$a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re)$: a_i is used to match to h_{i+1} received from m_{i+1}. If the match is successful, then the location of the first element could be identified, which means that the guard of $\overline{g}(l_{i+1}).\overline{match}(re)$ can be hence released. If the match is successful, it could lead to the execution of $\overline{g}(l_{i+1})$. The function of $\overline{g}(l_{i+1})$ is to send through g the current pointer l_{i+1} (the node pointer that matches to the first value a) to the comparing process P2 and P3, so that the comparison of the second round could begin right from the matching position for the comparing processes. Then after the action, the pointer sends the information of re to the result display process RESULT through the channel match.

$$\text{LAUNCH} = \overline{switch_1}.\overline{r_1}(l_0).g(link).(\prod_{t=2}^{3}(\overline{open}.\overline{switch_t})).(\overline{r_2}(link) \mid \overline{r_3}(link))$$

The main names and their related functions:

$Switch_1$ $Switch_2$ and $Switch_3$ in LAUNCH is the on/off switch to start the matching operation between the comparison string and the well-ordered string.

g(link) is used to receive the information of the first successful match as well as the message on the node location of the well-ordered process at the moment.

$\overline{r_1}$ (l_0), $\overline{r_2}$ (link) and $\overline{r_3}$ (link) is used to send the message about the well-ordered node to the comparing process. Among them, $\overline{r_1}$ (l_0) is used to send the initial node pointer l_0 of the well-ordered process to the first comparison process P1. $\overline{r_2}$ (link) is used to send the pointer link received from g(link) to the second comparing process P2. $\overline{r_3}$ (link) is used to send the pointer link received from g(link) to the third comparing process P3.

open is used to initiate the second comparison process P2 and the third comparison process P3 respectively.

P1(a_s,K1 ,r1)= $!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$

P2(a_j,K2 ,r2)=

$open.(!switch_2.r_2(k_2).\overline{k_2}(pc_2).(pc_2(f_2\ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc)))$

P3(a_j,K3 ,r3)=

$open.(!switch_3.r_3(k_3).\overline{k_3}(pc_3).(pc_3(f_3\ pre\ suc).\overline{f_3}(a_j\ front).\overline{r_3}(suc)))$

The main names and their related functions:

P1, P2 and P3 separately use r1(k1), r2(k2) and r2(k3) to receive channel information from the control process and the well-ordered process in order to form a loop. The function of the loop is to send nonstop the data that need matching to the nodes of the well-ordered process for matching. Still, switch1, switch2 and switch3 are needed to prevent the loss of control in the self-reaction of P1, P2 and P3.

On the other hand, another control is needed. When P1 has already been successfully matched but the message of the initial location for the comparison between P2 and P3

(the information on the first element) hasn't been sent, open, as the control switch, is also needed to prevent switch 2 and switch 3 from being wrongfully opened by (switch1+switch2+switch3) in the matching between P1 and the well-ordered process, because P1 cannot carry on the normal matching if the switch of P1 is used by P2 or P3. In order to avoid this kind of situation, only when P1 is successfully matched can the open switch be really opened. Only then can the two comparing processes P2 and P3 continue the matching activity while P1 finishes the rest of the movement. But no matter which open switch of the comparing process is turned on, the matching will go on without any problem.

RESULT=match(outcome).match(outcome).outcome

RESULT is used to show the comparing results.

The outcome received from the first match is loc, the location information of the first element The outcome received from the second match is the location information of the second element relative to the first element. If its location is before the first element, then the display is front; if its location is after the first element, then the display is back.

The computational description of the whole comparison is as follows:

welord |LAUNCH|P1|P2|P3|RESULT

The specification process is as follows:

1) Initiate the process launch. When the switch of the comparing process P1 is turned on, the initial process launch becomes:

$$\overline{r_1}(l_0).g(link).(\prod_{t=2}^{3}(\overline{open}.\overline{switch_t})).(\overline{r_2}(link)\,|\,\overline{r_3}(link))$$

The comparing process P1 becomes:

$$r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\,|\,!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

2) Then the initial process launch sends the initial pointer l_0 to the comparing process P1, so the initiating process becomes:

$$g(link).(\prod_{t=2}^{3}(\overline{open}.\overline{switch_t})).(\overline{r_2}(link)\,|\,\overline{r_3}(link))$$

The comparing process becomes:

$$\overline{l_0}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\,|\,!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

3) The comparing process P1 uses the channel l_0 to send pc1 to the well-ordered process.

The comparing process P1 becomes:

$$(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\,|\,!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

The well-ordered process becomes:

$$\overline{pc_1}(m_1 l_{-1} l_1)\,|\,m_1(h_1\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_0\,|\,\overline{h_1}.\overline{g}(l_1).\overline{match}(re))\,|$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\,|\,!m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i\,|\,\overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

4) The well-ordered process uses the only channel pc1 to send (m1 l_{-1} l1) to the comparing process. Meanwhile, the name of (f1 pre suc) in the comparing process will be changed into (m1 l_{-1} l1). Since the process P1 looks for the match in the one-way direction towards the end of the list, it just needs to receive the parameter pre without doing any other things. What needs to be done with is to change f1 into m1 and l1 into suc. So now the comparing process becomes:

$$(\overline{m_1}(a_s\ loc).\overline{r_1}(l_1))\,|\,!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

At this moment, the well-ordered process becomes:

$$m_1(h_1\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_0 \mid \overline{h_1}.\overline{g}(l_1).\overline{match}(re))\mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

Note that: $m_1(h_1\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_0 \mid \overline{h_1}.\overline{g}(l_1).\overline{match}(re))$ can be combined with

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

But here it is presented alone, just for the clear explanation later.

5) The comparing process and the well-ordered process use the channel m_1 for communication. After the comparing process P1 sends a_s and loc through m_1 to the well-ordered process, it becomes:

$$(\sum_{j=1}^{3}\overline{switch_j})\ .(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

The comparing process becomes:

$$\overline{r_1}(l_1)\mid !switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

That is:
$$\overline{r_1}(l_1)\mid switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\mid$$
$$!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

6) The well-ordered process initiates through switch1 the next move of the comparing process P1. Then the comparing process becomes:

$$(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\mid\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

The comparing process P1 becomes:

$$\overline{r_1}(l_1)\mid r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\mid$$

$$!switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

7) The comparing process P1 stipulates internally that l_1 replaces k1. In this way, the node of the well-ordered process moves from l_0 to l_1. Then the comparing process P1 becomes:

$$\overline{l_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))\mid !switch_1.r_1(k_1).\overline{k_1}(pc_1).(pc_1(f_1\ pre\ suc).\overline{f_1}(a_s\ loc).\overline{r_1}(suc))$$

8) The comparison can be done in the similar reasoning. When the comparison comes to a_s, the well-ordered process becomes:

$$(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\mid(a_1 \mid \overline{a_s}.\overline{g}(l_2).\overline{match}(loc))\mid(a_2 \mid \overline{a_s}.\overline{g}(l_3).\overline{match}(loc))\mid ...\mid(a_s \mid \overline{a_s}.\overline{g}(l_{s+1}).\overline{match}(loc))\mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

9) In this case, a_s will be neutralized with $\overline{a_s}$, and the π guard of n will be released. Then the well-ordered process becomes:

$$(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\mid(a_1 \mid \overline{a_s}.\overline{g}(l_2).\overline{match}(loc))\mid(a_2 \mid \overline{a_s}.\overline{g}(l_3).\overline{match}(loc))\mid ...\mid(\overline{g}(l_{s+1}).\overline{match}(loc))\mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

10) The g in the well-ordered process initiates the process launch once again and sends

the information of the current node to the process launch, preparing for a new round of comparison, but in a concurrent way.

The initial process launch becomes: $(\prod_{t=2}^{3}(\overline{open}.\overline{switch_t})).(\overline{r_2}(l_{s+1})\,|\,\overline{r_3}(l_{s+1}))$

At this moment, if the message of match in the well-ordered process hasn't been sent yet, then the well-ordered process is:

$(a_0\,|\,\overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\,|\,(a_1\,|\,\overline{a_s}.\overline{g}(l_2).\overline{match}(loc))\,|\,(a_2\,|\,\overline{a_s}.\overline{g}(l_3).\overline{match}(loc))\,|\,...\,|\,\overline{match}(loc)\,|$

$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\,|\,!m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i\,|\,\overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$

11) After $\overline{g}(l_{s+1})$ in the well-ordered process sends l_{s+1} to P2， welord sends the information of loc to result through match, which indicates a successful match for P1. That is, a_s has been successfully matched. So now the result display process becomes match(loc).loc.

As for the well-ordered process which sends the message of loc through the channel match, it becomes:

$(a_0\,|\,\overline{a_s}.\overline{g}(l_1).\overline{match}(loc))\,|\,(a_1\,|\,\overline{a_s}.\overline{g}(l_2).\overline{match}(loc))\,|\,(a_2\,|\,\overline{a_s}.\overline{g}(l_3).\overline{match}(loc))\,|\,...\,|\,(a_{s-1}\,|\,\overline{a_s}.\overline{g}(l_s).\overline{match}(loc))\,|$

$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1})\,|\,!m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i\,|\,\overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$

12) The initial process concurrently sends two pieces of open information, which starts the comparing process P2 and P3 respectively, and then the initial process also starts switch 2 and switch 2.

So the initial process becomes: $\overline{r_2}(l_{s+1})\,|\,\overline{r_3}(l_{s+1})$

P2 becomes: $r_2(k_2).\overline{k_2}(pc_2).(pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc))\,|$
$(!switch_2.r_2(k_2).\overline{k_2}(pc_2).(pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc)))$

P3 becomes: $r_3(k_3).\overline{k_3}(pc_3).(pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc))\,|$
$(!switch_3.r_3(k_3).\overline{k_3}(pc_3).(pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc)))$

13) The initial process concurrently sends the node information to P2 and P3.

P2 becomes: $\overline{l_{s+1}}(pc_2).(pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc))\,|$
$(!switch_2.r_2(k_2).\overline{k_2}(pc_2).(pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc)))$

P3 becomes: $\overline{l_{s+1}}(pc_3).(pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc))\,|$
$(!switch_3.r_3(k_3).\overline{k_3}(pc_3).(pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc)))$

14) The comparing process P2 and P3 concurrently communicate with the well-ordered process welord through l_{s+1}. On the one hand, P2 sends pc_2 to the well-ordered process, which uses pc_2 (whose name is changed from c_{s+1}) to send m_{s+1} and l_{s+1} to the process P2. Then the comparing process p2 again uses m_{s+1} to send the second element aj to the well-ordered process. On the other hand, similar to P2, P3 uses the same way to send aj to the well-ordered process. Although P2 and P3 have the same starting point for matching in the well-ordered process, they differ a lot in the direction of the movement: P2 starts from the s+1th element towards the end of well-ordered set, while P3 moves towards the beginning of the well-ordered set for the one-by-one match.

At this moment, though P1 has already got a successful match, it will continue the

movement with the well-ordered process concurrently with the matching activities between the well-ordered process and P2 and between it and P3. Suppose the matching between the well-ordered process and P1 hasn't reached the next step and suppose the matchings of P2 and P3 have, then the well-ordered process becomes:

$$(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc)) \mid (a_1 \mid \overline{a_s}.\overline{g}(l_2).\overline{match}(loc)) \mid (a_2 \mid \overline{a_s}.\overline{g}(l_3).\overline{match}(loc)) \mid ... \mid (a_{s-1} \mid \overline{a_s}.\overline{g}(l_s).\overline{match}(loc)) \mid$$

$$\overline{pc_2}(m_{s+1}l_{s-1}l_{s+1}) \mid !m_{s+1}(h_{s+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_{s+1} \mid \overline{h_{s+1}}.\overline{g}(l_{s+1}).\overline{match}(re) \mid$$

$$\overline{pc_3}(m_{s+1}l_{s-1}l_{s+1}) \mid !m_{s+1}(h_{s+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_{s+1} \mid \overline{h_{s+1}}.\overline{g}(l_{s+1}).\overline{match}(re) \mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1}) \mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{3}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

P2 becomes：
$$pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc) \mid (!switch_2.r_2(k_2).\overline{k_2}(pc_2).(pc_2(f_2\ \ pre\ suc).\overline{f_2}(a_j\ back).\overline{r_2}(suc)))$$

P3 becomes：
$$pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc) \mid (!switch_3.r_3(k_3).\overline{k_3}(pc_3).(pc_3(f_3\ \ pre\ suc).\overline{f_3}(a_j\ \ front).\overline{r_3}(suc)))$$

If the well-ordered process and the concurrent matches with P2, P3 and the post-successful-match p1 have reached the next node of their own, the well-ordered process becomes:

$$(a_0 \mid \overline{a_s}.\overline{g}(l_1).\overline{match}(loc)) \mid (a_1 \mid \overline{a_s}.\overline{g}(l_2).\overline{match}(loc)) \mid (a_2 \mid \overline{a_s}.\overline{g}(l_3).\overline{match}(loc)) \mid ... \mid (a_{s-1} \mid \overline{a_s}.\overline{g}(l_s).\overline{match}(loc)) \mid$$

$$a_{s+1} \mid \overline{a_s}.\overline{g}(l_{s+1}).\overline{match}(loc) \mid$$

$$a_{s+1} \mid \overline{a_j}.\overline{g}(l_{s+1}).\overline{match}(back) \mid a_s \mid \overline{a_j}.\overline{g}(l_s).\overline{match}(front) \mid$$

$$\prod_{i=0}^{n}(!l_i(c_{i+1}).\overline{c_{i+1}}(m_{i+1}l_{i-1}l_{i+1}) \mid !m_{i+1}(h_{i+1}\ re)\ .\ (\sum_{j=1}^{5}\overline{switch_j})\ .(a_i \mid \overline{h_{i+1}}.\overline{g}(l_{i+1}).\overline{match}(re))$$

15) In this way, the post-successful-match p1 will continue the rest of the movement with the well-ordered process. Meanwhile, P2 and P3 will also concurrently look for the match with the well-ordered process. Through the time, the post-successful-match p1 and P2 will move towards the end of the well-ordered process, while P3 will move towards the opposite direction. Since P1 has already gone through a successful match, its function now is to locate the position of a_s in the well-ordered process. Therefore, whatever P1 is doing now, it will have no influence on the result. As for P2 and P3, their function is to locate the position of a_j in relation to the position of a_s. If P2 matches successfully, then the result display process will become back, indicating that a_j is located after a_s. On the contrary, if P3 matches successfully, then the result display process will become front, indicating a_j is located before a_s.

5. Conclusion

As a frequently used operation in the concurrent system modeling, the element comparison is thoroughly discussed with three different ways of algorithm according to different element storage manners. When it comes to the way of random storage of the elements, the first solution is the best choice in comparing two randomly selected elements in a set with relatively fewer elements. But the solution may be compromised in expressing when the comparison becomes more complicated with the increase of the elements in the set. The second solution aims to make comparisons between any two elements in the set stored in the linked list.

The corresponding expression and maintenance are relatively easier, but the comparative efficiency is a little lower due to its limitation by the storage manner of the linked list. In summary, all these two solutions have their own advantages and disadvantages suitable to deal with different kinds of element comparison.

Acknowledgements

The work has been funded by the Program of Critical Theories and Technological Researches on the New Information Network (SKLSE-2015-A-06) of the State Key Lab of Software Engineering, Computer School of Wuhan University, China.

References

[1] Milner, Bobin, "Communicating and Mobile Systems: the π-calculus[M]", Cambridge, UK: Cambridge University Press, **(1999)**.
[2] R. Milner, "The polyadic-calculus: a tutorial. Logic and Algebra of Specification", vol. 94, **(1993)**, pp. 91–180.
[3] R. Milner, "The Polyadic π-Calculus: a Tutorial. Theoretical Computer Science", vol. 198, **(1997)**, pp. 239–249.
[4] H. Bu and S. Chen, "The comparison realization of the expressions based on π-calculus", vol. 33, **(2015)**, pp. 1281-1296.
[5] X. Liu and D. Walker, "A Polymorphic Type System for the Polyadic pi-calculus", In Proceedings of the 6th International Conference on Concurrency Theory, Springer-Verlag London, UK, **(1995)**, pp. 103–116.
[6] J. Hongye, "Verification on the Web Service Composition Based on Pi-Calculus", Graduation thesis Taiyuan University of Technology.
[7] C. Xiaojuan, "The Expressiveness of π-Calculus via Programming", Dissertation Shanghai, Jiao Tong University.
[8] H. Kegang, "The Pi+ Calculus - An Extension of the Pi Calculus for Expressing Petri Nets[J]", Chinese Journal of Computers, vol. 34, no. 2, **(2011)**, pp. 193-203.
[9] K. Hui, "Modeling the Mobile Communication Service Based on PI-calculus[J]", Journal on Communications, vol. 30, no. 4l, **(2009)**, pp. 11-16.
[10] Y. Wei, "Research on formal verification of web services flow based on pi-calculus", Graduation thesis Zhejiang University.
[11] P. Yu, "WS-BPEL Modeling and Realization Based on π-Calculus", Graduation thesis Northwest University.

