Verifying Strong Ground Bisimilarity of Quantum Communication Protocols

First Author[1], Second Author[2,3], and Third Author[3]

[1] University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. This paper gives a strong ground bisimulation verification algorithm for quantum programs. We further implement the algorithm that enable us to make experiments on existing quantum communication protocols. As a preparation of the experiments, we encode the quantum communication protocol into a quantum program. Then we check whether a quantum program is bisimilar with its specification. According to the results, we make some improvements on the quantum programs.

Keywords: Quantum programs · Verification · Bisimualtion.

1 Introduction

2 Preliminaries

3 qCCS

4 Bisimulation Verification

In this section, we give an algorithm to verify the strong ground bisimulation between two pLTSs and show its implementation.

4.1 A Strong Ground Bisimilarity Checking Algorithm

The algorithm 1 is based on the work of [1]. The main function is **Bisim**(t, u), its job is to initialize the start states pair (t, u), visited states pair W which is an empty set and then find the bisimulation basing on that initialization. The difference between it and the previous work in several aspects.

The algorithm keeps updating two sets W for visited states pairs and N for non-bisimilar states pairs. The function **Match**(t, u, W) invokes a depth-first traversal to match a pair of states (t, u) with all their possible behaviors. The states pair is checked to be non-bisimilar if one of their transitions are

not matched or their quantum variables, so do the quantum registers, are not matched. Then the algorithm adds the current states pair into N.

An auxiliary function $\mathbf{Act}(t, u)$ is called in **Match** to discover the next action that both two states can behave. If both two states have no more action it returns an empty set. Furthermore, if there is only one of them has no more action it will lead to a non-bisimilarity. It makes the algorithm more efficient as it terminates at an early time point if those pLTSs are indeed not bisimilar. Next we prove why we can ensure that.

Lemma 1. *Todo: support that I can ensure non-bisimilar if there is a trace longer than any trace from another side.*

Theorem 1 (Early termination). *If the algorithm reaches a leaf state of the tree-like pLTS while the state of the other pLTS is not leaf state, then these two pLTSs are not bisimilar.*

Proof. We consider it on the aspect of the length of the traces. From the structure of the algorithm, each time **MatchAction** is called **Act** will be called before it. So the we can ensure that two states have the same action to behave. There exists the trace that

$$T = \langle t_0, \rho\rangle \xrightarrow{\gamma_0} ... \xrightarrow{\gamma_i} \langle t_i, \rho\rangle \text{ and } U = \langle u_0, \sigma\rangle \xrightarrow{\gamma_0} ... \xrightarrow{\gamma_i} \langle u_i, \sigma\rangle.$$

Let $|T| = |U| = n$, if one of these states u_i is not leaf state, then there has a longer trace $|U'| = n + 1$. As there is no loop contained, the trace $|T'| = n + 1$ does not exist, so we can not find a trace has the same length as the other one. According to the definition of the open bisimulation, these two pLTS should not satisfy the relation.

The other set W is update in function $\mathbf{MatchAction}(\gamma, t, u, W)$. It discovers next states pairs according to the action γ and recursively invokes the function **Match** when there is a pair of states or **MatchDistribution** when there is a pair of distributions of states. The current states pair is added to W when it invokes new function.

The $\mathbf{MatchDistribution}(\Delta, \Theta, R)$ is an extra step if we match a pair of distributions of states instead of a single pair of states. It returns a boolean value indicating if the distributions are bisimilar. It continues invoking **Match** to match next pair of states from the pair of states distributions. After checking the bisimilarity of their states, the function generates an equivalence relation of the states from the distribution those who are not contained in the non-bisimilar states set N. Another auxiliary set $\mathbf{Check}(\Delta, \Theta, R)$ is used for checking the lifting condition of the bisimulation relation. Besides the lifting condition, we check the disjunction of the returning boolean value from **Match** functions. The function return such result basing on the following theorem.

Theorem 2 (Bisimulation of distributions).

Let $R \subseteq Dist(Con) \times Dist(Con)$ be the (strong) open bisimulation relation between two distributions, then for any $\mu, \nu \in Dist(Con)$, $\mu R \nu$ can imply that:

(1) The relation satisfies the lifting condition, that is $\mu = \sum_{i\in I} p_i C_i$*, for each* $i \in I$*,* $C_i R D_i$ *for some* D_i*, and* $\nu = \sum_{i\in I} p_i D_i$*.*
(2) The set I *is not an empty set, s.t.* $\exists C, D \in Con, \mu(C) > 0 \wedge \nu(D) > 0$*.*

Proof. From the definition of the lift operation, the condition must be satisfied. And we need to filter the case that two distributions have no behaviour in common, so there need at least a pair of states is bisimilar.

The correctness of the algorithm is presented in the theorem below.

Theorem 3 (Termination).
Given two states t *and* u *from two pLTSs,* ***Bisim(t,u)*** *always terminates.*

Proof. So far there is no while-loop in the qCCS, that brings convenience to the proof of termination. Starting at the initial pair of states, the next action to do will be detected in the function **Match**. Then it invokes function **MatchAction** to find the next new pair of states and recursively call function **Match** to check them. Each time function **MatchAction** calls function **Match** it adds the current states pair into W at the same time. If we reach the leaf nodes, there is no more action, we only compare the quantum variables used and the state of quantum registers. After that, the function terminates, so do the calls to the other functions. Moreover, if there still exists actions to do in one of the pLTS while another one does not, that means they are not strong bisimilar and then the whole algorithm terminates.

Then we consider the correctness of the algorithm. First, we let (θ, N) to be the return pair of functions, moreover (θ_{ij}, N_{ij}) is the return of the $i \cdot j$-th execution of the function. To simplify the explanation, we use $\mathtt{R}(t, u, W, N)$ to mean the following condition is satisfied:

If $(t', u') \in N$, then $\forall t' \xrightarrow{\alpha} t'', u' \xrightarrow{\alpha'} u'',\ t'' \not\sim u''$.

If $(t', u') \notin N$, then $\forall t' \xrightarrow{\alpha} t'', \exists u' \xrightarrow{\alpha'} u''$ and $(t'', u'') \notin \{(t, u)\} \cup W$ such that:

- if $\alpha \equiv a$ (including $c?x$) then $\alpha' \equiv a$ and $(t'', u'') \notin W \wedge (t'', u'') \notin N \implies t'' \sim u''$.
- if $\alpha \equiv c!e$ then $\alpha' \equiv c!e'$ with $e = e'$ and $(t'', u'') \notin W \wedge (t'', u'') \notin N \implies t'' \sim u''$.
- let $t'' \equiv \Delta'$ and $u'' \equiv \Theta'$, if $\alpha \equiv \tau$ then $\alpha' \equiv \tau$, $\forall t'_i \in \lceil \Delta' \rceil, u'_j \in \lceil \Theta' \rceil,\ (t'_i, u'_j) \notin W \wedge (t'_i, u'_j) \notin N \implies t'_i \sim u'_j$.

We define the verification conditions of our three matching functions.

Definition 1. ***Match***(t, u, W) *is true if the following condition is satisfied:*

- *(C1)* $W \cap N = \emptyset$*.*
- *(C2) if* $(t, u) \in W$*, then* $(t, u) \notin N$*.*
- *(C3) if* $(t, u) \notin W$*, then either* $\theta = true \wedge (t, u) \notin N$ *or* $\theta = false \wedge (t, u) \in N$*.*

– *(C4)* $\mathtt{R}(t, u, W, N)$.

Let $\boldsymbol{Bisim}(t, u) = \boldsymbol{Match}(t, u, \emptyset)$.

Definition 2. $\boldsymbol{MatchAction}(\gamma, t, u, W)$ *is true if the following conditions are satisfied:*

– *(M1)* $W \cap N = \emptyset$, $(t, u) \notin W$ *and* $(t, u) \notin N$.
– *(M2)* $\mathtt{R}(t, u, W, N)$.
– *(M3)* $\forall t \xrightarrow{\alpha} t', \exists u \xrightarrow{\alpha'} u'$, $(t', u') \notin \{(t, u)\} \cup W$ *and*
  - *if* $\alpha \equiv a$ *(including* $c?x$*) then* $\alpha' \equiv a$ *and* $(t', u') \notin W \wedge (t', u') \notin N \implies t' \sim u'$.
  - *if* $\alpha \equiv c!e$ *then* $\alpha' \equiv c!e'$ *with* $e = e'$ *and* $(t', u') \notin W \wedge (t', u') \notin N \implies t' \sim u'$.
  - *let* $t' \equiv \Delta$ *and* $u' \equiv \Theta$, *if* $\alpha \equiv \tau$ *then* $\alpha' \equiv \tau$, $\forall t_i \in \lceil \Delta \rceil, u_j \in \lceil \Theta \rceil$, $(t_i, u_j) \notin W \wedge (t_i, u_j) \notin N \implies t_i \sim u_j$.

Definition 3. $\boldsymbol{MatchDistribution}(\Delta, \Theta, W)$ *is true if the following conditions are satisfied:*

– *(D1)* $W \cap N = \emptyset$, $\forall t_i \in \lceil \Delta \rceil, u_j \in \lceil \Theta \rceil$, $(t_i, u_j) \notin W$.
– *(D2) Let* $t \xrightarrow{\alpha} \Delta, u \xrightarrow{\alpha'} \Theta$, $\mathtt{R}(t, u, W, N)$.

Proposition 1. *Let* $\boldsymbol{MatchAction}_\gamma(\gamma, t, u, W)$ *is the execution of the function with action* γ. *If* $\boldsymbol{MatchAction}_\gamma(\gamma, t, u, W)$ *is true for each action* γ *then* $\boldsymbol{Match}(t, u, W)$ *is also true, where it returns* $\theta = \bigwedge_\gamma \theta_\gamma$ *and* $N = \bigcup_\gamma N_\gamma$.

Proof. As (t, u) is added into W during the execution of **MatchAction**, according to the Definition 2, $W \cap N = \emptyset$.

$(t, u) \notin N_\gamma$ implies that $(t, u) \notin N = \bigcup_\gamma N_\gamma$. (t, u) will be added into N if $\theta = \wedge_\gamma \theta_\gamma$ is $false$. Since the verified pLTS is a finite tree, if they reach the leaf states of the pLTSs, there should be $\theta = true$ and $N = \emptyset$, at the same time it satisfies that $(t, u) \notin W \wedge (t, u) \notin N$.

So it satisfies the verifying condition of **Match**.

Proposition 2. *Suppose* $(t, u) \notin W$. *If* $\boldsymbol{Match}(t_i, u_j, W \cup \{(t, u)\})$ *is true for all action* $\gamma \neq \tau$ *derivations*$(t \xrightarrow{\gamma} t_i,\ u \xrightarrow{\gamma} u_j)$ *or* $\boldsymbol{MatchDistribution}(\Delta_i, \Theta_j, W \cup \{(t, u)\})$ *is true for all action* $\gamma = \tau$ *derivations* $(t \xrightarrow{\tau} \Delta_i, u \xrightarrow{\tau} \Theta_j)$ *then* $\boldsymbol{MatchAction}(\gamma, t, u, W \cup \{(t, u)\})$ *is true where* $\theta = \bigwedge_i (\bigvee_j \theta_{ij}) \wedge \bigwedge_j (\bigvee_i \theta_{ij})$, $N = \bigcup N_{ij}$.

Proof. From the structure of the function **MatchAction**, the first condition is satisfied.

To show M2 and M3, we first consider the case where (t_i, u_j) is already the leaf of the finite tree, and if $\theta_{ij} = qv(t_i) = qv(u_j) \wedge (tr_{\overline{qv(u_i)}}\rho_i) = tr_{\overline{qv(u_j)}}(\sigma_j)$ is $true$, $(t_i, u_j) \notin N_{ij}$ and $t_i \sim u_j$.

If it is not the leaf node, by (C4), we have $\mathtt{R}(t_i, u_j, \{(t, u)\} \cup W, N_{ij})$. Since $\theta = \bigwedge_i (\bigvee_j \theta_{ij}) \wedge \bigwedge_j (\bigvee_i \theta_{ij})$ and $N = \bigcup_{ij} N_{ij}$, if θ is true, there exists θ_{ij} are

true, then from $(t_i, u_j) \notin N_{ij}$, we get $(t'_i, u'_j) \notin N_{ij} \implies t'_i \sim u'_j$. In the case θ is false, $(t, u) \in N$ has already distinguished the non-bisimilarity.

The final case we need consider is the distribution (Δ, Θ) instead of a node. If θ is true, then **Check** returns θ_{ij} is also *true*. So there must exist **Match** returns *true* implies that $(t_i, u_j) \notin N \implies t_i \sim u_j$.

Proposition 3. *Suppose* $\forall t_i \in \lceil \Delta \rceil, u_j \in \lceil \Theta \rceil, (t_i, u_j) \notin W$. *If* $\boldsymbol{Match}(t_i, u_j, W)$ *is true then* $\boldsymbol{MatchDistribution}(\Delta, \Theta, W)$ *is true where* Δ *and* Θ *satisfy the condition for lifting condition,* $\theta = \boldsymbol{Check}(\Delta, \Theta, R) \wedge \bigvee_{ij} \theta_{ij}$ *and* $N = \bigcup_{ij} N_{ij}$.

Proof. According to the verification conditions of **March**, as all the $\mathbf{Match}(t_i, u_j, W)$ have been finished before we get R and call **Check**. If $\Delta \sim \Theta$, then we have $(t_i, u_j) \notin N \implies t_i \sim u_j$.

Theorem 4 (Correctness).
Given two states t and u from two pLTSs, $\boldsymbol{Bisim}(t, u)$ *returns true if and only if they are bisimilar.*

Proof. We use the definition of bisimulation and the propositions proved above. From the verification condition of **Match**, we have that if $\mathbf{Bisim}(t, u) = \mathbf{Match}(t, u, \emptyset)$ returns $(true, N)$, we guarantee the bisimilarity $t \sim u$.

At the end of this section, we compute the time complexity of the algorithm.

Theorem 5 (Complexity). *Let the number of nodes in two transition graphs reachable from t and u is n. The time complexity of function* $\boldsymbol{Bisim}(t, u)$ *is* $O(n^5/log\ n)$ *and the space complexity of it is* $O(n^2)$.

Proof. The number of state pairs is bounded by n^2. Since the graphs there are finite trees, for each pair of states, the number of comparisons of transitions could be n^2 in the worst case. So the visited state pairs set W contains at most $O(n^2)$ elements.

And if in each call of **MatchAction**, it call **MactchDistribution** other than **Match**, there will cost more time checking the condition of lifting operation through **Check**. In previous work **??**, we know **Check** cost time $O(n^3/log\ n)$. As a result, the execution of $\mathbf{Bisim}(t, u)$ takes at most $O(n^5/log\ n)$ time in total.

4.2 Implementation

Our tool is implemented in Python 3.7. Its workflow is illustrated in Fig. 1. The input is a quantum program and its specification, both of them are described in qCCS. Execution of the tool yields a terminal output showing the details of the whole process, including the pLTS generation and the checking algorithm, and the result of the checking by a table mapping each pair of pLTS states to its most general boolean. The tool invokes Z3 solver to verify the most general boolean of the initial state pair. A counterexample will be given if the boolean can be unsatisfied.

pLTS Generation The tool inputs programs codes containing three parts,a description of process behaviors, an initialization of their variables and a set of user-defined quantum gates. Process behaviors are described in qCCS semantics. Processes are separated by semicolons. Quantum gates can be defined through a set of kraus operators, they are also separated by semicolons. The intermediate output of the module is the pLTS which will be used as the input of bisimilarity checking module.

Bisimulation Checking We implement the previously defined ground bisimlarity checking algorithm to verify the generated pLTSs. The input needs two pLTSs, one for protocol description and another for specification description. They are processed by the pLTS generation module. We start at the initial states of these two pLTSs. The result of the module is also the final result of the tool presenting whether these two pLTSs are bisimilar, always with a set preserving non-bisimilar state pairs.

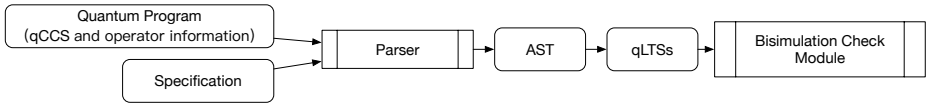

Fig. 1. Verification workflow.

5 Case Studies and Experiments

We provide several classical quantum communication protocols described in qCCS as the use cases for our experiment.

5.1 Examples: Quantum Communication Protocols

Super-dense Coding Protocol There are two roles *Alice* and *Bob*. To simplify the experiment, we only consider the smallest case of the protocol, sending only one qubit. So in this example, there is totally one entanglement on two qubits. Besides the Clifford operators, we use a quantum operation Set^{Ψ} to present the generation of Bell state instead of the combination of the quantum gates. The operation elements of Set^{Ψ} is $\{|\beta_{00}\rangle\langle 00|, |\beta_{00}\rangle\langle 01|, |\beta_{00}\rangle\langle 10|, |\beta_{00}\rangle\langle 11|\}$. The measurement is according to the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. The specification of the super-dense protocol is defined as *Bob* sets the 2-qubit variable to the value according to the classical value he received from *Alice*.

Quantum Teleportation Protocol In this example, there are still two roles. The operators we used here is similar with the last example containing Clifford

operators, Set^{Ψ} and the measurement according to the computational basis. However, we need one more entanglement and one more qubit if we just consider the smallest case. As there are entanglements between these qubits, the measurement on just a part of them may also affect the rest qubits. We considers the final result of this protocol, that is the third one, $Bob's$ qubit, becomes the same value of the first qubit. So the specification of that can be presented by applying a $SWAP$ operation between the first and the third qubit.

BB84 Quantum Key Distribution Protocol In BB84 protocol, there is no entanglement at all, its idea is generating qubits on different basis and using different measurement method without any contacts in advance with the other side. If someone try intercepting the information, the qubits might be measured in wrong basis, it brings a possible case that *Alice* and *Bob* are aware of the attack. So the protocol uses one more measurement according to the diagonal basis $\{|+\rangle, |-\rangle\}$. In common use case, BB84 will send a sequence of the qubits while qubits will not influence each other. We consider two kinds of result of the communication. First case is that *Alice* and *Bob* choose the same measurement then the results they get are also the same. Another case is that they choose different measurements then the result is discarded at this time. In the specification, we get results from the same sequence instead of two result sequence separately. Considering results from both sides is always the same, this operation will not bring any difference.

BB84 Protocol with an Eavesdropper This example is an extension of the BB84 example, supposing there is an eavesdropper attending into the communication. There have three roles and the new role *Eve* also randomly choose the measurement just as what *Alice* and *Bob* do. The specification is also similar with the one without an eavesdropper. It is possible that the eavesdropper will be recognized. It is a new result of the program. We conclude these results into three messages: emitting through the channel *alarm* as the measurement methods are not matched; emitting through the channel $fail$ as the measurement methods are matched while the eavesdropping is recognized; normal emitting as the communication finished without recognizing the eavesdropper.

5.2 Experimental Results

We conducted experiments on those quantum communication protocols, and improved our input program according to the experiment results. The results were obtained on a macOS machine with an Intel Core i7 2.5 GHz processor and 16GB of RAM.

Experimental Results and Improvement Table 1 provides a summary of our experimental results over those four examples. In each case, we report the bisimilarity, the number of non-bisimilar states pair in N and the runtime of our checking algorithm.

We verify the super-dense coding with two different initial valuations of variable x in the first two lines. In the case $x = 1$, we can check that protocol and its specification are bisimilar. However, in the case $x = 5$, when none of the four branches is chosen, they are not bisimilar because of the different length of the trace. The result shows that the program misses the solution for the valuation out of the expected scale. We improve the program through adding a new branch solving all the unexpected value. The result of the improved program is presented on the third line.

Another example brings a non-bisimilarity is on the sixth line of the table, the BB84 protocol considering the eavesdropper. *Alice* and *Bob* will make an alert if their measurement methods are not matched. The parallelism between the final test process and them leads to the process continues behaving some actions. That is not what the specification exactly describes. To improve this program, we modify the behavior, move the alert to the test process. *Alice* and *Bob* only send messages when they find they use different measurements. As a result, on the last line of the table, we find the program is bisimilar with the specification.

Discussion Not all the cases of Table 1 present the size of the non-bisimilar states set N, as the checking algorithm has terminated in advance. To ensure the bisimilarity between program with a large set of states and its specification requires much more time, over 24 times of the runtime of checking non-bisimilarity. However, the runtime of finding two pLTSs are non-bisimilar is not that long enables us to try making improvement in an acceptable time waiting feedback.

6 Conclusion and Future Works

References

1. Feng, Y., Deng, Y., Ying, M.: Symbolic Bisimulation for Quantum Processes. ACM Trans. Comput. Log. **15**(2), 14:1–14:32 (2014)
2. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
3. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
4. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
5. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
6. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017

Program	**Variables**	**Bisimilarity**	**Size of** N	**Runtime**(s)
Super-dense coding 1	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $x = 1$	Yes	0	2.2
Super-dense coding 2	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $x = 5$	No	-	2.2
Super-dense coding (modified)	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $x = 5$	Yes	0	2.5
Teleportation	$q1 = \|1\rangle$ $q2 = \|0\rangle$ $q3 = \|0\rangle$	Yes	0	2.7
BB84	$q1 = \|0\rangle$ $q2 = \|0\rangle$	Yes	304	4.7
BB84 (with eavesdropper)	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $q3 = \|0\rangle$	No	-	74.6
BB84 (with eavesdropper & modified)	$q1 = \|0\rangle$ $q2 = \|0\rangle$ $q3 = \|0\rangle$	Yes	17272	1834

Table 1. Experimental Results

Algorithm 1 Bisim(t,u)

Require: A pair of initial states for matching t,u.
Ensure: A boolean value θ showing if two pLTSs are bisimilar and a set of non-bisimilar state pairs N.

function Bisim(t, u)
  return Match(t,u,W)

function Match(t, u, W) $\triangleright$ $t = \langle t, \rho\rangle$ *and* $u = \langle u, \sigma\rangle$
  if $t, u \in W$ **then**
    $\theta := \mathtt{tt}$
  else
    for $\gamma \in \mathtt{Act}(t, u)$ **do**
      $(\theta_\gamma, N_\gamma)$:=**MatchAction**$(\gamma,t,u,W)$
    θ:=$\bigwedge_\gamma \theta_\gamma \wedge qv(t) = qv(u) \wedge tr_{\overline{qv(t)}}(\rho) = tr_{\overline{qv(t)}}(\sigma)$
    $N = \bigcup_\gamma N_\gamma$
    if $\theta = \mathtt{ff}$ **then** N:=$N \cup \{(t, u)\}$
  return (θ, N)

function MatchAction(γ, t, u, W)
  switch γ **do**
    case $c!$
      for $t \xrightarrow{c!e_i} t_i$ and $u \xrightarrow{c!e'_j} u_j$ **do**
        $(\theta_{ij}, N_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j(\theta_{ij} \wedge e_i = e'_j)) \wedge \bigwedge_j(\bigvee_i(\theta_{ij} \wedge e_i = e'_j)), \bigcup_{ij} N_{ij})$
    case τ
      for $t \xrightarrow{\tau} \Delta_i$ and $u \xrightarrow{\tau} \Theta_j$ **do**
        $(\theta_{ij}, N_{ij}) :=$ **MatchDistribution**$(\Delta_i, \Theta_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij})$
    otherwise
      for $t \xrightarrow{\gamma} t_i$ and $u \xrightarrow{\gamma} u_j$ **do**
        $(\theta_{ij}, N_{ij}) :=$ **Match**$(t_i, u_j, W \cup \{(t, u)\}$
      return $(\bigwedge_i(\bigvee_j \theta_{ij}) \wedge \bigwedge_j(\bigvee_i \theta_{ij}), \bigcup_{ij} N_{ij})$

function MatchDistribution(Δ, Θ, W)
  for $t_i \in \lceil\Delta\rceil$ and $u_j \in \lceil\Theta\rceil$ **do**
    $(\theta_{ij}, N_{ij}) :=$ **Match**(t_i, u_j, W)
  R:=$\{(t_i, u_j) | (t_i, u_j) \notin \bigcup_{ij} N_{ij}\}*$
  return (**Check**$(\Delta,\Theta,R) \wedge \bigvee_{ij} \theta_{ij}, \bigcup_{ij} N_{ij}$)