An algebraic approach to transactional processes

An Algebraic Approach to Transactional Processes

H.M.A. van Beek

Department of Mathematics and Computing Science
Technische Unversiteit Eindhoven
P.O. Box 513, NL–5600 MB Eindhoven, The Netherlands
harm@win.tue.nl

Abstract

We present a set of operators in order to simplify the modelling of transactional behaviour of processes using process algebra. We give an axiomatic semantics of the operators presented. Apart from that, we give their operational semantics using Plotkin-style deduction rules. Our goal is to give formal specifications of Internet applications using process algebra, for which transactional behaviour should be modelled.

1 Introduction

Nowadays, a growing amount of activities take place via the Internet. To give some examples, one can vote, visit an auction and make payments via the Internet instead of going to a polling place, auction hall or a bank. Most of these processes are based on the concept of transactions. A transaction can be seen as a "set" of (inter)actions which occur "as a group" [14]. I.e., they either all or none succeed. This means that if during a transaction something goes wrong, the visitor does not have to fear that his vote is counted without him wanting to, that his correct bid for an object is ignored or that he pays an amount of money which will never reach the receiver's account. If the transaction isn't finished correctly, it is undone as if it never took place.

Apart from Internet applications, nearly all database systems implement transactional access: after doing table updates, one has to commit the updates to make them visible to other users. As long as the updates are not committed, they can be undone (rolled back).

So transactions cover a significant part of all processes. Our goal is to give formal specifications of Internet applications for distributed consensus [4, 3] which comes as close to real-life applications as possible. We therefore need to have a formalism to handle transactional processes. Since we have chosen to use process algebra [7] for the specification of Internet applications, we prefer modelling transactional behaviour of processes in process algebra. Although transactional behaviour can already be specified in process algebra with recursion (as will be proven in Section 6), introducing specific transactional operators cause specifications being shorter and thus more legible and manageable. In this paper we present these operators by giving both an axiomatic and operational semantics.

In Section 2, transactions are explained in more detail. We give an example by which the usage of transactional processing is clarified in Section 3. Next, in Section 4, we adapt the concept of transactions in a way that makes it able to specify transactional behaviour using process algebra. We do this by using axiomatic semantics. In Section 5 we give operational semantics for the operators. We make use of the semantics for proving soundness of the axioms in Section 6. Finally, we discuss future work and related work in Sections 7 and 8 and we draw some final conclusions in Section 9.

2 Transactions

Gray and Reuter [17] define a *transaction* as a "set" of (inter)actions which occur "as a group", meaning that they either all succeed, or none of them do. If all actions within a transaction are completed, the global state can be changed by an atomic and synchronized *commit* statement. During execution of this commit statement no other process can access data updated by that statement.

To get a feeling for the usage of transactions, have a look at the following example. Suppose someone is going to the polls, e.g. to vote for a president. Then this voting process consists of several steps. First of all, the voter has to identify himself by showing his passport. If the identification succeeds, he gets a ballot containing the names of the candidates. The voter enters the polling booth, fills in the ballot and leaves the polling booth. Up till now, the voter has not voted yet. By destroying the ballot any time in the process, the voting is rolled back. However, if the voter puts his ballot in the ballot box he "commits" his vote. In this example, the commit action is an atomic action: putting the ballot in the ballot box. However, it might be possible that this action cannot be atomic. E.g. when doing a payment from one bank account to the other. The first account should be reduced and the second one increased. Suppose that something goes wrong while increasing the second account after the first one has been reduced. Then the entire transaction should be undone in such a way that the owner of the first account gets his money back and the second account contains exactly the amount of money as before starting the transaction. So transactions help in coupling related actions that should all succeed or none of them.

Another example where transactions can be convenient is when two or more parallel processes access shared data. E.g., suppose that two persons try to book the final seat for a flight. They both log in to the airline's website and fill in and submit the booking form. Then by submitting the form, they do both update shared data, viz. the set of available seats. When not using transactions, the flight can get overbooked or one of the two passengers does not get registered although he received a confirmation. So transactions can be of interest if parallel processes access shared data.

As shown in the examples, the sharing of data can lead to unwanted or unexpected behaviour since updating and reading of the data can interleave. To prevent applications from having unexpected behaviour, its parallel components should meet the so-called ACID properties [16, 18]. ACID is an acronym for *atomicity*, *consistency*, *isolation* and *durability*.

Atomicity A transaction's changes to the state are atomic: either all happen or none happen.

Consistency A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state.

Isolation Even though transactions execute concurrently, it appears to each transaction, T, that other transactions take place either before T or after T. So isolation means that a program under transaction protection must behave exactly as it would do in single-user mode.

Durability Once a transaction completes successfully (commits), its change to the state survives failure.

Processes that meet all four characteristics are called *transactions*. Transactions, and therefore atomic actions, are the basic building blocks for constructing applications. Transactions can be nested. So a transaction can contain subtransactions.

In general, a transaction consists of subtransactions, read and write actions, ended by a commit action. If during a transaction something goes wrong, a rollback takes places, undoing all data changes, and the transaction can start over again. If all actions succeed, the commit statement causes the data changes to be durable.

During execution of parallel transactions, a transaction can lock other transactions by accessing shared data. That is, transactions can cause other transactions to come in a state in which they are not allowed to execute specific actions. This locking mechanism prevents accessing so-called dirty data (i.e. data that has been changed, but not committed yet) by using *read locks*. Furthermore,

by using *write locks* it prevents having lost updates, i.e. changed data is updated by another transaction before it had been committed. Unlocking takes place while committing or rolling back a transaction.

In this paper we focus on transactions that are not allowed to update data that is updated by another running transaction, so-called first degree isolated[1] transactions. So we only take write locks into account. Read locks can be added to the formal definition for transactions in a similar way as write locks. So by leaving out read actions and read locks we do not reduce the complexity in a major way. Since we focus on adding transactional behaviour of the processes, we also leave out the explicit changes to the data space.

To give an idea on how transactions are used, we first give an example in Section 3. After that, we explain the way we adapt the concepts of transactions to process algebra.

3 Example

We give a small example which nicely shows the behaviour of the transactional operator in defining processes. In this section, we only give some informal definitions of the operators. They will be formalized in later sections. Have a look at the following two processes:

$$(a := 0 \cdot a := a + 2) \parallel (a := 1 \cdot a := a \times 2)$$

and

$$\langle\!\langle a := 0 \cdot a := a + 2 \rangle\!\rangle \parallel \langle\!\langle a := 1 \cdot a := a \times 2 \rangle\!\rangle \quad .$$

The assignments to variable a can be seen as atomic actions. By using the $\cdot$ operator we compose these actions into sequentially executable processes. E.g., $a := 0 \cdot a := a + 2$ specifies that first a becomes 0 after which a is increased by 2.

Both processes consist of two subprocesses which are placed in parallel using the merge operator ($\parallel$). As mentioned, each subprocess sequentially executes two assignments to variable a. In the second process, we make use of $\langle\!\langle$ and $\rangle\!\rangle$ brackets to embrace the subprocesses, which turns them into transactions. Since both transactional processes $\langle\!\langle a := 0 \cdot a := a + 2 \rangle\!\rangle$ and $\langle\!\langle a := 1 \cdot a := a \times 2 \rangle\!\rangle$ access shared variable a simultaneously, write access to variable a in one of the transactions locks the other transactions until a rollback or commit takes place. See Figure 1 for the intended process graphs of both processes.

In the first process, normal interleaving of the two subprocesses on either side of the merge operator is allowed. This leads to 6 ($\frac{4!}{2!2!}$) possible solutions, resulting in three possible outcomes: a equals 2, 4 or 6.

By turning both subprocesses into transactions, write access to a in one of the subprocesses leads to locking the other subprocess (see the right graph in Figure 1). If a subprocess is not finished (i.e. if only one of the two actions is executed) a rollback ($\mathcal{R}$) can take place, resulting in a transition to the state before starting the subprocess. If both actions in a subprocess are executed, the transaction can commit ($\mathcal{C}$), which leads to unlocking the other subprocess.

It can be easily seen that although we have an infinite number of possible executions (viz. rollbacks can take place infinitely often), if the process finishes then a equals 2.

The example given in this section nicely shows the expressiveness of the transactional operator for defining transactional behaviour.

4 Adding Transactions to Process Algebra

Our starting point is an algebraic axiomatisation BPA (Basic Process Algebra), as described in [7]. The signature of BPA consists of action alphabet $\mathbb{A}$, *alternative composition operator* $+$ and *sequential composition operator* $\cdot$. The axioms for BPA, A1–5, are given in Table 1.

[1] More information on degrees of isolation can be found in Section 7.2.

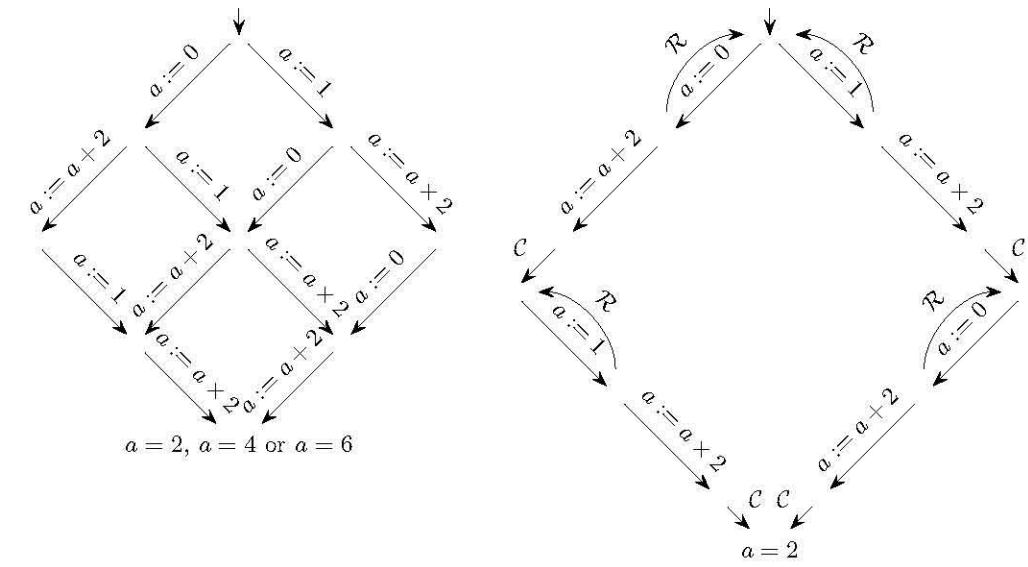

Figure 1: An example of using the transactional operator.

To group actions into transactions, we need a transactional composition operator. As mentioned in Section 3, we turn a process into a transaction by embracing it using $\langle\langle$ and $\rangle\rangle$ brackets. In this paper we focus on transactional behaviour of the processes, so we leave out the actual data changes. Furthermore, we abstract from read access to shared variables. Therefore we can look at an action a as being a write action to a shared variable which is uniquely identifiable by a. The right process in Figure 1, for example, would be modelled by $\langle\langle a \cdot a \rangle\rangle \| \langle\langle a \cdot a \rangle\rangle$.

If a transaction executes action a, $a \in \mathbb{A}$, all transactions running in parallel with this transaction should be locked with respect to write access for shared variable a. During execution of a transaction, something can go wrong, e.g. a connection gets lost or a time-out takes place. The entire transaction has to be rolled back, unlocking all other transactions that were locked by actions executed in this transaction. After this rollback, the transaction can start over again. If no rollback takes place, the transaction can commit, causing other transactions to get unlocked as well. For specifying this mechanism, we make use of an auxiliary operator $\langle\langle \_, \_, \_ \rangle\rangle$. The first parameter will be used for storing the actual transactional process. In case of a rollback, we make use of this parameter to restart the process. The second parameter is used for storing the set of executed actions, i.e. the shared variables that are updated. This set is used for the unlocking of other transactions and resetting the variable's values. Note that a set will be sufficient since all variables have only one value before being updated by a transaction. Apart from that, only the first write action from within a transaction influences the locking mechanism. Finally, the last parameter contains that part of the process that needs to be executed before the transaction can commit. So $\langle\langle x, A, y \rangle\rangle$ can be read as *"transactional process x, which has already executed the set of (unique) actions A and still has to execute process y before a commit statement can take place"*.

Transaction $\langle\langle x, \emptyset, x \rangle\rangle$ can be compared with transaction $\langle\langle x \rangle\rangle$ which has not executed any of its actions yet. If a transaction $\langle\langle x, A, y \rangle\rangle$ has already executed an action, i.e. if $A \neq \emptyset$, it can roll back, using rollback action $\mathcal{R}_A$. This causes all transactions that are locked with respect to variables in A to get unlocked. Next, $\langle\langle x \rangle\rangle$ can start over again. If a transaction commits, action $\mathcal{C}_A$ is executed which also unlocks other transactions. Since we abstracted from the data changes, $\mathcal{R}_A$ and $\mathcal{C}_A$ behave equally. So we make use of $\mathcal{U}$ (Unlocking action) to represent either $\mathcal{C}$ or $\mathcal{R}$.

Definition 4.1 *Let A be a set of actions, $A \subseteq \mathbb{A}$ and $\mathcal{U}$ be an unlocking action, $\mathcal{U} \in \{\mathcal{C}, \mathcal{R}\}$. Then $\mathcal{U}_A$ is the unlocking action that unlocks all actions in A that are locked in other transactions, once.*

$x+y$	$=$	$y+x$	A1	$\lceil \mathcal{U}_B \rceil_b$	$=$	$\mathcal{U}_B$		L1
$(x+y)+z$	$=$	$z+(y+z)$	A2	$\lceil a_n \rceil_b$	$=$	a_{n+1}	if $a=b$	L2
$x+x$	$=$	x	A3	$\lceil a_n \rceil_b$	$=$	a_n	if $a \neq b$	L3
$(x+y)z$	$=$	$xz+yz$	A4	$\lceil a \rceil_b$	$=$	a		L4
$(xy)z$	$=$	$x(yz)$	A5	$\lceil \mathbf{a}x \rceil_b$	$=$	$\lceil \mathbf{a} \rceil_b \cdot \lceil x \rceil_b$		L5
				$\lceil x+y \rceil_b$	$=$	$\lceil x \rceil_b + \lceil y \rceil_b$		L6
$x \| y$	$=$	$x \lfloor\!\lfloor y + y \lfloor\!\lfloor x$	M1					
$\mathcal{U}_A \lfloor\!\lfloor x$	$=$	$\mathcal{U}_A \lfloor x \rfloor_A$	M2	$\lfloor \mathcal{U}_B \rfloor_A$	$=$	$\mathcal{U}_B$		UL1
$a_n \lfloor\!\lfloor x$	$=$	$a_n \lceil x \rceil_a$	M3	$\lfloor a_n \rfloor_A$	$=$	a_{n-1}	if $a \in A \wedge n > 0$	UL2
$a \lfloor\!\lfloor x$	$=$	ax	M4	$\lfloor a_n \rfloor_A$	$=$	a_n	if $a \notin A \vee n = 0$	UL3
$\mathcal{U}_A x \lfloor\!\lfloor y$	$=$	$\mathcal{U}_A(x \| \lfloor y \rfloor_A)$	M5	$\lfloor a \rfloor_A$	$=$	a		UL4
$a_n x \lfloor\!\lfloor y$	$=$	$a_n(x \| \lceil y \rceil_a)$	M6	$\lfloor \mathbf{a}x \rfloor_A$	$=$	$\lfloor \mathbf{a} \rfloor_A \cdot \lfloor x \rfloor_A$		UL5
$ax \lfloor\!\lfloor y$	$=$	$a(x \| y)$	M7	$\lfloor x+y \rfloor_A$	$=$	$\lfloor x \rfloor_A + \lfloor y \rfloor_A$		UL6
$(x+y) \lfloor\!\lfloor z$	$=$	$x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M8					

$\langle\!\langle x \rangle\!\rangle$	$=$	$\langle\!\langle x, \emptyset, x \rangle\!\rangle$		TR1
$\langle\!\langle x, \emptyset, \mathcal{U}_B \rangle\!\rangle$	$=$	$\mathcal{U}_\emptyset \cdot \mathcal{C}_\emptyset$		TR2
$\langle\!\langle x, A, \mathcal{U}_B \rangle\!\rangle$	$=$	$\mathcal{U}_\emptyset \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $A \neq \emptyset$	TR3
$\langle\!\langle x, A, a_n \rangle\!\rangle$	$=$	$\mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $n > 0 \wedge A \neq \emptyset$	TR4
$\langle\!\langle x, \emptyset, a_0 \rangle\!\rangle$	$=$	$a_0 \cdot \mathcal{C}_{\{a\}}$		TR5
$\langle\!\langle x, A, a_0 \rangle\!\rangle$	$=$	$a_0 \cdot \mathcal{C}_{A \cup \{a\}} + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \notin A \wedge A \neq \emptyset$	TR6
$\langle\!\langle x, A, a_0 \rangle\!\rangle$	$=$	$a \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \in A$	TR7
$\langle\!\langle x, \emptyset, a \rangle\!\rangle$	$=$	$a_0 \cdot \mathcal{C}_{\{a\}}$		TR8
$\langle\!\langle x, A, a \rangle\!\rangle$	$=$	$a_0 \cdot \mathcal{C}_{A \cup \{a\}} + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \notin A \wedge A \neq \emptyset$	TR9
$\langle\!\langle x, A, a \rangle\!\rangle$	$=$	$a \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \in A$	TR10
$\langle\!\langle x, \emptyset, \mathcal{U}_B y \rangle\!\rangle$	$=$	$\mathcal{U}_\emptyset \cdot \langle\!\langle x, \emptyset, y \rangle\!\rangle$		TR11
$\langle\!\langle x, A, \mathcal{U}_B y \rangle\!\rangle$	$=$	$\mathcal{U}_\emptyset \cdot \langle\!\langle x, A, y \rangle\!\rangle + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $A \neq \emptyset$	TR12
$\langle\!\langle x, A, a_n y \rangle\!\rangle$	$=$	$\mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $n > 0 \wedge A \neq \emptyset$	TR13
$\langle\!\langle x, \emptyset, a_0 y \rangle\!\rangle$	$=$	$a_0 \cdot \langle\!\langle x, \{a\}, y \rangle\!\rangle$		TR14
$\langle\!\langle x, A, a_0 y \rangle\!\rangle$	$=$	$a_0 \cdot \langle\!\langle x, A \cup \{a\}, y \rangle\!\rangle + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \notin A \wedge A \neq \emptyset$	TR15
$\langle\!\langle x, A, a_0 y \rangle\!\rangle$	$=$	$a \cdot \langle\!\langle x, A, y \rangle\!\rangle + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \in A$	TR16
$\langle\!\langle x, \emptyset, ay \rangle\!\rangle$	$=$	$a_0 \cdot \langle\!\langle x, \{a\}, y \rangle\!\rangle$		TR17
$\langle\!\langle x, A, ay \rangle\!\rangle$	$=$	$a_0 \cdot \langle\!\langle x, A \cup \{a\}, y \rangle\!\rangle + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \notin A \wedge A \neq \emptyset$	TR18
$\langle\!\langle x, A, ay \rangle\!\rangle$	$=$	$a \cdot \langle\!\langle x, A, y \rangle\!\rangle + \mathcal{R}_A \cdot \langle\!\langle x \rangle\!\rangle$	if $a \in A$	TR19
$\langle\!\langle x, A, y+z \rangle\!\rangle$	$=$	$\langle\!\langle x, A, y \rangle\!\rangle + \langle\!\langle x, A, z \rangle\!\rangle$		TR20

Table 1: PAtrans: Process Algebra with Transactions

We introduce a new action alphabet, $\mathbb{UL}$, containing unlock actions:

$$\mathbb{UL} = \{\mathcal{U}_A \mid \mathcal{U} \in \{\mathcal{C}, \mathcal{R}\}, A \subseteq \mathbb{A}\} .$$

As can be seen in Definition 4.1, execution of a unlocking action unlocks actions once. If more than two transactions run in parallel, actions can get locked more than once. Therefore, we provide a mechanism to extend actions with a locking counter indicating how many times the action is locked.

Definition 4.2 *Let a be an action and n be a natural number. Then a_n is a lockable action. Action a_n represents action a which is locked n times. In a_n, n is called a* lock counter*.*

Again, we introduce a new action alphabet, $\mathbb{L}$, containing lockable actions:

$$\mathbb{L} = \{a_n \mid a \in \mathbb{A}, n \in \mathbb{N}\} .$$

If transaction $\langle\!\langle x, A, y \rangle\!\rangle$ executes action a for the first time, i.e. $a \notin A$, then a is extended with a lock counter having value 0 and A is extended with a. If the transaction executed an a before, i.e. $a \in A$, no locking counter is added since the transaction already has exclusive rights on action a. We now have all ingredients for giving the axiomatic semantics of the transactional operators, TR1–20 in Table 1. TR2 and TR3 state that if the only action in a transaction is a unlock action ($\mathcal{C}_B$ or $\mathcal{R}_B$), then this action is the result of a nested sub-transaction. This unlock action should not unlock actions outside the transactions, so B is replaced by $\emptyset$. After this (final)

action, the transaction can be committed. TR4–7 handle lockable actions. The transaction is not allowed to execute a locked action ($a_n, n > 0$, TR4), so in that case only a rollback can take place, after which the transaction can start over again. If the lockable action is not locked, i.e. its lock counter equals 0, it can be executed. Next, the transaction can be committed. As mentioned, we differentiate between actions that occur in A and those that do not (TR5–7). Axioms TR8–10 are similar to TR5–7, and TR11–19 are similar to TR2–10.

We still need some operators to increase and decrease the lock counters. We introduce two operators, $\lceil x \rceil_a$ and $\lfloor x \rfloor_A$ on processes to lock and unlock processes, respectively. The locking operator has two parameters, process x and action a. $\lceil x \rceil_a$ means that all lockable a actions in x get locked (once more). Unlocking operator $\lfloor x \rfloor_A$ also has two parameters, viz. process x and a set of actions A meaning that all locked actions that occur in x which are elements of A get unlocked once. The axioms for both operators, L1–6 and UL1–6, are given in Table 1. We introduce a new variable, $\mathbf{a}$, which has a range of $\mathbb{A} \cup \mathbb{L} \cup \mathbb{UL}$.

Up till now, we have not mentioned how the operators introduced so far cooperate to reach the expected transactional behaviour. Since blocking of transactions is only of interest when transactions run in parallel, accessing a shared data space, we specify parallel composition using the merge ($\|$) and left-merge ($\lfloor\!\lfloor$) operators based on the merge operators introduced in [8]. The axioms for the parallel composition, M1–8, are also given in Table 1. If a unlocking action is executed in parallel with other processes, the action is executed and the process running in parallel gets unlocked once (M2 and M5). Execution of a lockable action locks the processes running in parallel (M3 and M6). All other actions do not influence the parallel running processes' behaviour (M4 and M7). The process algebra given by the axioms in Table 1 is denoted by PAtrans, Process Algebra with Transactions.

5 Operational Semantics

The semantics of the process algebra with transactions is given by the term deduction system induced by the deduction rules shown in Table 2. The variables are defined as in the axioms. The deduction rules are given using a Plotkin-style notation [20]. These are similar to the operational rules of most process algebras. We use predicate $\xrightarrow{\mathbf{a}} \surd$ to denote that a process may execute $\mathbf{a}$ and then terminate. We make a case distinction over the atomic actions that can be executed.

Each process $\mathbf{a}$ can do an $\mathbf{a}$-step and then terminate (rule 1). Unlocking actions and non-lockable actions are influenced by neither the locking (rules 6, 7, 9, 10, 11 and 13) nor the unlocking operator (rules 14, 15, 17, 18, 19 and 21). Depending on the parameters of the locking and unlocking operators, lock counters can be increased (rules 8 and 12) or decreased (rules 16 and 20), respectively.

Rule 22 handles rollback actions: If a transaction has started ($A \neq \emptyset$), the transaction can roll back by executing a rollback action $\mathcal{R}_A$, after which the transaction can start over again. A unlocking action that comes from within a transaction may not influence actions outside the transaction (rules 23, 26, 31 and 34). Furthermore, normal actions and lockable actions which are not locked, i.e. a_n with $n = 0$, can always be executed (rules 24, 25, 27–30, 32, 33 and 35–38). Depending on whether they have been executed by the transaction before, i.e. if a is in A, they get lockable and they are added to the set of executed actions A.

For defining the deduction rules for the parallel composition operators, again we distinguish between atomic action in $\mathbb{A}$, $\mathbb{L}$ and $\mathbb{UL}$. Execution of unlock actions cause the process running in parallel to get unlocked once (rules 39, 42, 45 and 48) where execution of lockable actions introduce the lock operator, causing the parallel running processes to get locked once more (rules 40, 43, 46 and 49). All other executions do not influence parallel running processes (rules 41, 44, 47 and 50).

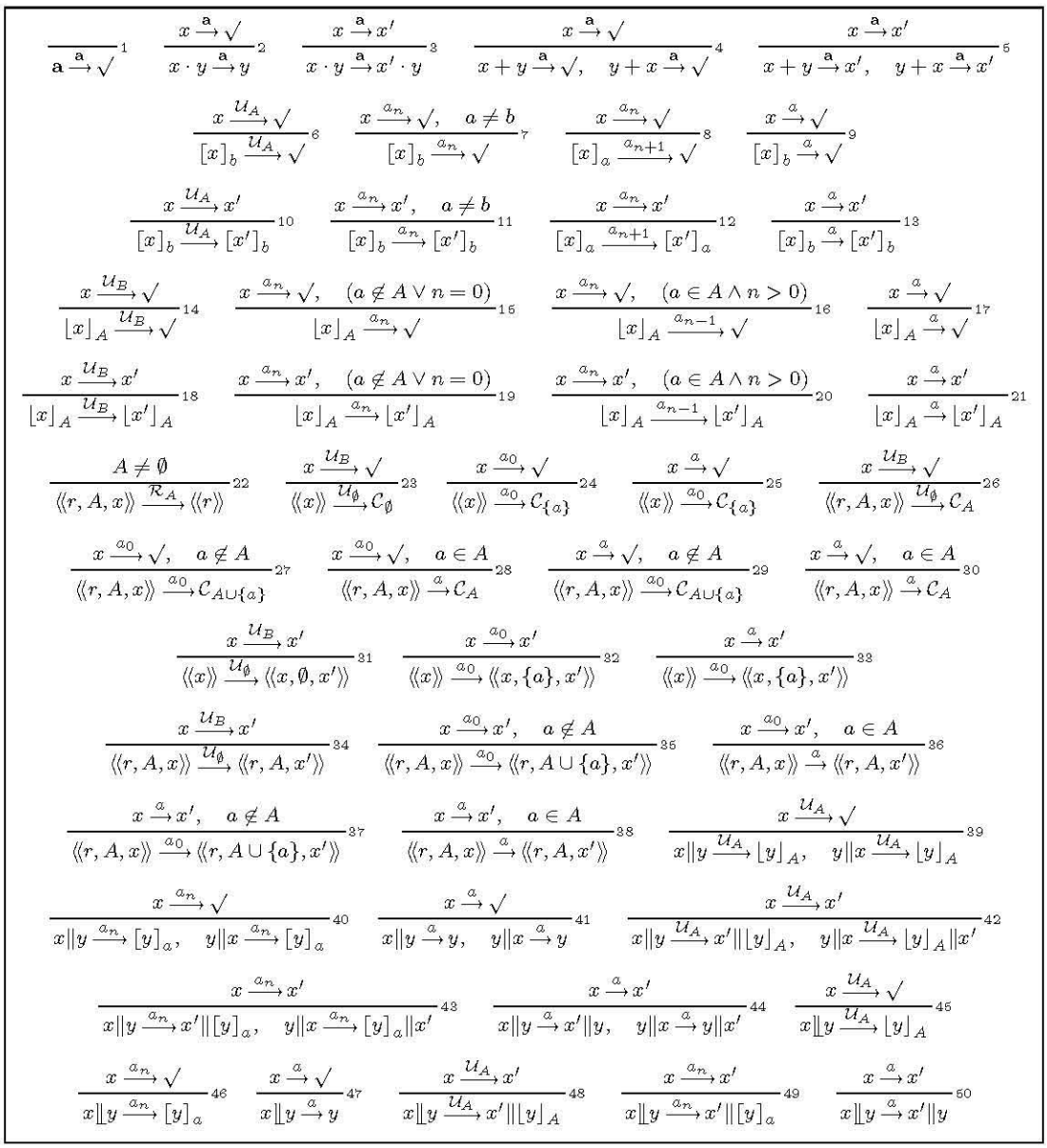

Table 2: Operational Semantics of PAtrans

6 Properties of Process Algebra with Transactions

In this section we prove some properties of PAtrans. First of all, we give a soundness proof, i.e. we prove that the set of closed PAtrans terms modulo bisimulation equivalence, $T(\text{PAtrans})/\underline{\leftrightarrow}$, is a model for PAtrans.

Definition 6.1 (Bisimulation for PAtrans**)** *Bisimulation for* PAtrans *is defined as follows: a binary relation R on closed terms in* PAtrans *is a bisimulation if and only if the following* transfer conditions *hold for all closed* PAtrans *terms p and q:*

1. *if $R(p,q)$ and $T(\text{PAtrans}) \models p \xrightarrow{\mathbf{a}} \surd$, where $\mathbf{a} \in \mathbb{A} \cup \mathbb{L} \cup \mathbb{UL}$, then $T(\text{PAtrans}) \models q \xrightarrow{\mathbf{a}} \surd$.*
2. *if $R(p,q)$ and $T(\text{PAtrans}) \models p \xrightarrow{\mathbf{a}} p'$, where $\mathbf{a} \in \mathbb{A} \cup \mathbb{L} \cup \mathbb{UL}$, then there exists a process term q' such that $T(\text{PAtrans}) \models q \xrightarrow{\mathbf{a}} q'$ and $R(p',q')$.*

Two closed PAtrans *terms p and q are bisimular, notation $p \leftrightarrows q$, if there exists a bisimulation relation R such that $R(p, q)$.*

Using bisimulation, we can now construct a model for the axioms of PAtrans. In order to do this, we first need to know that bisimulation is a congruence with respect to all operators.

Lemma 6.2 (Bisimulation is a congruence) *Let T(*PAtrans*) be the term deduction system induced by the deduction rules shown in Table 2. Then bisimulation equivalence is a congruence on the set of closed* PAtrans *terms.*

Proof It can be easily seen that the operational semantics given in Table 2 is in *path format* [5]. Since it is proven that if a term deduction system is in path format, strong bisimulation is a congruence [6] (based on [5, 15]), this immediately proves this lemma. □

So now that we know that bisimulation is a congruence, we can construct a model for the axioms of PAtrans.

Definition 6.3 (Bisimulation model for PAtrans) *The bisimulation model for* PAtrans *is constructed by taking the equivalence classes of the set of all closed* PAtrans *terms with respect to bisimulation equivalence. As bisimulation is a congruence, the operators can be pointwise defined on the equivalent classes.*

Theorem 6.4 (Soundness) *The set of closed* PAtrans *terms modulo bisimulation equivalence, T(*PAtrans*)$/\leftrightarrows$, is a model for* PAtrans*.*

Proof We will prove this theorem by proving that each axiom is sound, i.e., prove that for all closed instantiations of the axiom both sides of the axiom correspond to the same element of the bisimulation model. This proof outline is taken from [24, 6]. The proof can be found in Section A.1. □

Apart from proving soundness, we prove that all PAtrans terms can be eliminated to a term in a basic process algebra with recursion (BPArec), as defined in [6]. By doing this, we prove that the expressiveness of the process theory has not increased. Since the same set of processes can be defined without using PAtrans operators, the transactional operators introduced can be considered a syntactic extension for simplifying the notation of transactional processes. The following definitions are based on [6].

Definition 6.5 (Recursive specification) *Let V be a set of variables. A* recursive specification *$E = E(V)$ is a set of so-called recursion equations, $E = \{X = s_X(V) \mid X \in V\}$, where each $s_X(V)$ is a* BPA *term that only contains variables of V. The set of actions of the* BPA *is $\mathbb{A} \cup \mathbb{L} \cup \mathbb{UL}$.*

Definition 6.6 (Solution) *A* solution *$\{\langle X|E\rangle \mid X \in V\}$ of a recursive specification $E(V)$ is a set of processes in some model of* BPA *such that replacing variable X by $\langle X|E\rangle$ in the recursion equations of $E(V)$ yields true statements in that model.*

Definition 6.7 *Let $E = E(V)$ be a recursive specification and let t be an open* BPA *term. Then $\langle t|E\rangle$ is the process t with all variables X both occurring in t and V replaced by $\langle X|E\rangle$.*

We give a set of recursion equations in the proof of the elimination theorem. Apart from the atomic actions of PAtrans and all lockable and unlock actions, the domain of BPArec, $\mathbb{A}_{\text{BPArec}}$, contains the set of constants $\langle X|E\rangle$ for all $X \in V$:

$$\mathbb{A}_{\text{BPArec}} = \mathbb{A} \cup \mathbb{L} \cup \mathbb{UL} \cup \{\langle X|E\rangle \mid X \in V\} .$$

Since a solution is a (possibly empty) set of processes, we extend the model T(PAtrans) with two assumptions, RSP and RDP$^-$, which are defined in Definitions 6.8 and 6.9, respectively. These assumptions state that there is exactly one solution. Before giving the exact definitions, we introduce the concept of guardedness. We call an occurrence of variable X in term t *guarded* if t has a sub-term of the form $a \cdot s$ with s a BPA term containing this occurrence of X. If we can rewrite a term to a guarded term using the axioms, we call this term guarded too.

Definition 6.8 *The* Recursive Specification Principle (RSP) *is the following assumption:*

A guarded recursive specification has at most one solution.

Definition 6.9 *The* Restricted Recursive Definition Principle (RDP^-) *is the following assumption:*

Every guarded recursive specification has a solution.

We can only make use of these two principles if all recursive occurrences of processes in PAtrans are guarded.

Theorem 6.10 *All recursive occurrences of processes in* PAtrans *are guarded.*

Proof The proof can be easily seen by looking at the axioms for the transactional composition in Table 1, TR1–20. Since recursion only takes place after a rollback, all recursion specifications, i.e. all occurrences of $\langle\langle x\rangle\rangle$ on the right-hand side of the equal sign in axioms T2–20, are preceded by a rollback action $\mathcal{R}$. □

Since all recursive specifications in PAtrans are guarded, we can make use of model $T(\mathrm{PAtrans})+\mathrm{RDP}^- + \mathrm{RSP}$ to prove the following elimination theorem.

Theorem 6.11 (Eliminiation to BPArec) *For every* PAtrans *term t there exists a guarded recursive specification E over* BPA *such that t is a solution of E.*

Proof This theorem is proven by induction on the general structure of t and can be found in Section A.2. □

Herewith we have proven that the transactional operators can be considered a syntactic extension for simplifying the process algebraic notation of transactional processes.

7 Towards Real-life Transactions

We introduce some additional concepts which enable us to better model real-life processes. As mentioned in Section 2, we abstracted from read access to variables and we only took first degree isolated transactions into account. We shortly discuss some concepts to come closer to real-life transactions.

7.1 Explicit Write Actions

Up till now, we only took write actions into account. However, both read actions and internal actions might be needed when giving real-life examples. Therefore, we need to make a distinction between write actions and other (i.e. read or internal) actions. In former sections, we assumed that execution of action a meant a write action to (shared) variable a. We can explicitly indicate whether an action is a write action to a variable. The set of actions A in $\langle\langle x, A, y\rangle\rangle$ will then be redefined such that it contains only write actions.

Although this is a major change in the notation and the way we deal with variables, this does not influence the axioms as presented in Section 4 drastically. If we consider 1° isolated transactions, there are dependencies between write actions only. So both read actions and internal actions are not influenced by the locking operator and therefore only write actions should have their lock counter increased. We make this distinction between different kinds of actions by adapting the conditions in the axioms and operational rules.

Note that since only (more) conditions on the format of the actions are added, this extension does not influence the soundness nor the elimination to BPArec substantially.

7.2 Degrees of Isolation

In real-life transactions, a distinction is drawn between four degrees of isolation, mainly due to performance issues [17]. For reaching our goal, i.e. for giving formal specifications of Internet applications, this notion of degrees of isolation should be added. A short overview of the different degrees of isolation is given in Table 3.

Issue	Degree 0	Degree 1	Degree 2	Degree 3
Common name	Chaos	Browse	Cursor Stability	Isolated
Protection provided	Lets others run at higher isolation	0° and no lost updates	No lost updates, no dirty reads	No lost updates, no dirty reads, repeatable reads
Transaction structure	Well-formed w.r.t. write	Well-formed w.r.t. write and two-phase w.r.t. write	Well-formed and two-phase w.r.t. write	Well-formed and two-phase
Dependencies	None	write → write	write → write write → read	write → write write → read read → write

Table 3: Degrees of isolation as given in [17].

A 0° isolated transaction is called *chaos*. It does not overwrite another transaction's dirty data if the other transaction is 1° or greater. A 1° isolated transaction is called *browse*. It prevents data updates to get lost. It is both well-formed and two-phase with respect to writes. A transaction is said to be *well-formed with respect to writes* if all data updates are preceded by locks, locking data updates to the same data in other transactions until it is committed or rolled back. *Two-phase* means that all locks precede all unlocks. A 2° isolated transaction, called *cursor stability*, has the same properties as a first degree transaction, but it also implements well-formedness with respect to reads. So also the reading of updated data by other transactions is locked. Finally, a 3° isolated transaction also locks data read by a transaction until it commits or rolls back. This is called *isolated*, *serializable* or *repeatable reads*. Optimally, all transactions should be 3° isolated.

Next to differentiating between the type of actions, we can also make a distinction between degrees of isolation. Although this does not increase the complexity of the transactional locking mechanism, we need to add lots of extra syntax to make this possible. One of the causes is that we need to introduce an explicit read action. On the other hand this is caused by the fact that actions from within 1° isolated transactions both are lockable and cause other actions to get locked. So the lock counter we make use of is not only a counter which stores the number of times the action is locked, but it also specifies that execution of the action causes other action to get locked (as stated in axiom M3). We call an action that causes other actions to get locked a *locking* action. In zeroth, second and third degree isolated transactions, lockable actions do not necessarily have to be locking and vice versa, as can be seen in Table 4.

degree of isolation	lockable		locking	
	write	read	write	read
zeroth	yes	no	no	no
first	yes	no	yes	no
second	yes	yes	yes	no
third	yes	yes	yes	yes

Table 4: Locking versus lockable actions

So apart from the lock counter, we need to extend actions from within transactions with a locking attribute to indicate whether the action is a locking action. This of course depends on the

degree of isolation, which should be added to the transactional operators.

Furthermore, we need to make a distinction between *shared locks* and *exclusive locks* [17]. Shared locks are set by read actions from within 3° isolated transactions and cause other transactions not to write to variables read by the transactions. However, the other transactions are allowed to read them. On the other hand, write actions set exclusive locks on variables, causing other transactions to get locked when accessing variables, even for read actions. A 3° isolated transaction can upgrade a shared lock to an exclusive lock by writing to a variable it has a shared lock on.

The set of variables that we stored in the auxiliary transactional operator (the A in $\langle\langle x, A, y\rangle\rangle$) is used for keeping track of the variables the transaction has exclusively locked. By introducing shared locks, we need to either split this set into two sets, one for shared and one for exclusive locks, or we should extend the elements in the set with an attribute which indicates whether the element is shared locked or exclusively locked.

To conclude, adding degrees of transactions to the formalism presented leads to the introduction of several extensions. Although these extensions do not make the concepts more complex, they introduce a considerably large amount of syntactical overhead, which goes beyond the scope of this article. Giving a detailed description of this is left for future work.

8 Related Work

Since both transactions and process algebra are widely used concepts, a lot of research is done in both areas.

Transactions can be considered as groups of actions. In process algebra there are mechanisms available to group actions. In [10] a mechanism is introduced for specifying asynchronous communication between processes, based on process algebra. Each communication consists of (independent) write and read actions. Each read action should be preceded by its corresponding write action. If this is not the case, actions can get locked which can be compared to transactional locking. The semantics of this mechanism is given in [13]. In [9] the *tight multiplication* operator is introduced. This operator is used in the same way as the sequential composition operator. However, no interleaving can take place between two actions which are composed into a process using this tight multiplication operator.

Transactions can roll back, causing actions being undone. Bergstra, Ponse and Wamel [11] add a mechanism for modelling this undoing of actions to process algebra.

The classical transaction concept appeared for the first time in [14]. In [16, 18] the ACID properties of transactions are explicitly indicated. A nice and complete overview of the main concepts of transactions is given and discussed by Gray and Reuter in [17].

Our model of nesting of transactions is an extension of the concept of nested transactions as developed by Moss [19]. Other extensions of Moss' concept are for example multi-level transactions [25] and open nested transactions [26].

For the concepts described in this paper we make use of the two-phase locking (2PL) protocol (i.e. all locks within a transaction precede all unlocks) as introduced in [14]. This concurrency control technique is widely used in most database management systems. Many variations on the 2PL technique exist [1, 23, 17, 21]. Apart from two-phase locking, other concurrency control mechanisms exist, like timestamp-ordering (TO) techniques [22] and optimistic schedulers [2]. All techniques can be combined into hybrid techniques, e.g. 2PL-TO combinations [12].

9 Conclusions

Both transactions and process algebra are widely used, however, as far as we know, no formalism to express transactions using process algebra has been developed before. In this paper we joined these concepts which led to a nice way of formally specifying transactional behaviour.

Although we abstracted from parts of the concept which must be added to make the formalism useful for specifying real-life processes, a basic framework for transactional reasoning using process algebra has been introduced. We summarized what has to be done to make the formalism put into practice. In our opinion, these extensions do not influence the complexity of the formalism, however, a lot of (syntactical) extensions have to be added.

By combining the concepts described in this paper with the process algebra we are making use of for modelling Internet applications [4, 3], we are able to give specifications that come closer to real-life applications. Currently, we are working on this combination to come to a complete process algebra for specifying Internet applications.

Acknowledgements

I would like to thank Jos Baeten and Sjouke Mauw for their valuable comments on previous versions of this paper.

References

[1] P.A. Alsberg and J.D. Day. A principle for resilient sharing of distributed resources. In *2nd International Conf. on Software Eng., San Francisco*, pages 562–570, San Francisco, CA, USA, 1976.

[2] D.Z. Badal. Correctness of concurrency control and implications for distributed databases. In *Proceedings of the IEEE COMPSAC 79*, Chicago, USA, November 1979.

[3] J.C.M. Baeten, H.M.A. van Beek, and S. Mauw. Operational semantics of DiCons, a formal language for developing internet applications. CS-Report 01/12, Dept. of Mathematics and Computer Science, Technische Universiteit Eindhoven, October 2001.

[4] J.C.M. Baeten, H.M.A. van Beek, and S. Mauw. Specifying internet applications with DiCons. In *Proceedings 16th ACM Symposium on Applied Computing*, pages 576–584, Las Vegas, USA, March 2001.

[5] J.C.M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *Proceeding of the International Conference on Concurrency Theory – CONCUR'93*, number 715 in Lecture Notes in Computer Science, pages 477–492, Hildesheim, Germany, 1993. Springer-Verlag.

[6] J.C.M. Baeten and C. Verhoef. Concrete process algebra. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Semantic Modelling*, volume 4 of *Handbook of Logic in Computer Science*, pages 149–268. Oxford University Press, 1995.

[7] J.C.M. Baeten and W.P. Weijland. *Process algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[8] J.A. Bergstra and J.W. Klop. Fixed point semantics in process algebras. Report IW 206, Mathematisch Centrum, Amsterdam, 1982.

[9] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[10] J.A. Bergstra, J.W. Klop, and J.V. Tucker. Process algebra with asynchronous communication mechanisms. In S.D. Brookes, A.W. Roscoe, and G. Winskel, editors, *Proc. Seminar on Concurrency*, number 197 in Lecture Notes in Computer Science, pages 76–95. Springer-Verlag, 1985.

[11] J.A. Bergstra, A. Ponse, and J.J. van Wamel. Process algebra with backtracking. In J. W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *REX Workshop*, number 803 in Lecture Notes in Computer Science, pages 46–91, Noordwijkerhout, The Netherlands, 1994. Springer-Verlag.

[12] P.A. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221, June 1981.

[13] F.S. de Boer, J.W. Klop, and C. Palamidessi. Asynchronous communication in process algebra. In Andre Scedrov, editor, *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 137–147, Santa Cruz, CA, June 1992. IEEE Computer Society Press.

[14] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. On the notions of consistency and predicate locks in a data base system. *Communications of the ACM*, 19(11), November 1976. Also published in/as: IBM, Res.R. RJ1487, San Jose, CA, December 1974.

[15] W.J. Fokkink. The tyft/tyxt format reduces to tree rules. In M. Hagiya and J.C. Mitchell, editors, *Proc. 2nd Symposium on Theoretical Aspects of Computer Software – TACS'94*, number 789 in Lecture Notes in Computer Science, pages 440–453, Sendai, April 1994. Springer-Verlag.

[16] J. Gray. The transaction concept: Virtues and limitations. In *International Conference On Very Large Data Bases (VLDB '81)*, pages 144–154, Los Angeles, CA, USA, September 1981. IEEE Computer Society Press.

[17] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[18] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, December 1983.

[19] J.E.B. Moss. *Nested Transactions: An Approach to Reliable Computing*. PhD thesis, MIT, 1981.

[20] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, Denmark, 1981.

[21] K. Salem, H. García-Molina, and J. Shands. Altruistic locking. *ACM Transactions on Database Systems*, 19(1):117–165, March 1994.

[22] R.M. Shapiro and R.E. Millstein. Reliability and fault recovery in distributed processing. In *OCEANS'77, Conference Record*, volume II, pages 31D.1–31D.5, Los Angeles, CA, USA, October 1977.

[23] R.E. Stearns and D.J. Rosenkrantz. Distributed database concurrency controls using before-values. In Y.E. Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 74–83, Ann Arbor, Michigan, 1981. ACM, New York.

[24] J.J. Vereijken. *Discrete-Time Process Algebra*. PhD thesis, Technische Universiteit Eindhoven, 1997.

[25] G. Weikum. A theoretical foundation of multilevel concurrency control. In *Proceedzngs of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 31–42, Cambridge, Massachusetts, March 1986.

[26] G. Weikum and H.-J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In A.K. Elmagarmid, editor, *Transaction Models for Advanced Database Applications*. Morgan Kaufmann, February 1992.

A Proofs

A.1 Soundness

Theorem A.1 (Soundness) *The set of closed* PAtrans *terms modulo bisimulation equivalence,* $T(\text{PAtrans})/\underline{\leftrightarrow}$*, is a model for* PAtrans*.*

Proof We will prove this theorem by proving that each axiom is sound, i.e., prove that for all closed instantiations of the axiom both sides of the axiom correspond to the same element of the bisimulation model. This proof outline is taken from [24, 6]. For each axiom, we take the relation which relates each process to itself (identity) and which relates the left-hand side of the equation to its right-hand side. So e.g. for proving axiom A1, we take relation

$$R = \{(x, x), (x + y, y + x) \mid x, y \text{ closed PAtrans terms}\} .$$

Furthermore, x, y, and z are closed PAtrans terms. We use subscript notation to indicate the deduction rules we make use of.

Axiom A1 We have to show that $x + y \underline{\leftrightarrow} y + x$.
First we look at the transitions of the left-hand side.

- Suppose $x + y \xrightarrow{a} \surd$. Then$_4$ $x \xrightarrow{a} \surd$ or $y \xrightarrow{a} \surd$. But then also$_4$ $y + x \xrightarrow{a} \surd$.
- Suppose $x + y \xrightarrow{a} z$. Then$_5$ $x \xrightarrow{a} z$ or $y \xrightarrow{a} z$. But then also$_5$ $y + x \xrightarrow{a} z$ and $R(z, z)$.

The proof of the right-hand side is analogous, replacing x by y and y by x.

Axiom A2 We have to show that $(x + y) + z \underline{\leftrightarrow} x + (y + z)$.
First we look at the transitions of the left-hand side.

- Suppose $(x+y)+z \xrightarrow{a} \surd$. Then$_4$ $x+y \xrightarrow{a} \surd$ or $z \xrightarrow{a} \surd$ iff$_4$ $x \xrightarrow{a} \surd$, $y \xrightarrow{a} \surd$ or $z \xrightarrow{a} \surd$ iff$_4$ $x \xrightarrow{a} \surd$ or $y + z \xrightarrow{a} \surd$ iff$_4$ $x + (y + z) \xrightarrow{a} \surd$.
- Suppose $(x+y)+z \xrightarrow{a} x'$. Then$_4$ $x+y \xrightarrow{a} x'$ or $z \xrightarrow{a} x'$ iff$_4$ $x \xrightarrow{a} x'$, $y \xrightarrow{a} x'$ or $z \xrightarrow{a} x'$ iff$_4$ $x \xrightarrow{a} x'$ or $y + z \xrightarrow{a} x'$ iff$_4$ $x + (y + z) \xrightarrow{a} x'$ and $R(x', x')$.

The proof of the right-hand side is analogous.

Axiom A3 We have to show that $x + x \underline{\leftrightarrow} x$.
First we look at the transitions of the left-hand side.

- Suppose $x + x \xrightarrow{a} \surd$. Then$_4$ $x \xrightarrow{a} \surd$ or $x \xrightarrow{a} \surd$, so $x \xrightarrow{a} \surd$.
- Suppose $x + x \xrightarrow{a} y$. Then$_5$ $x \xrightarrow{a} y$ or $x \xrightarrow{a} y$, so $x \xrightarrow{a} y$ and $R(y, y)$.

The proof of the right-hand side is trivial.

Axiom A4 We have to show that $(x + y)z \underline{\leftrightarrow} xz + yz$.
Note that there is no possibility for either the left-hand side or the right-hand side to execute a terminating action. First we look at the transitions of the left-hand side. Suppose $(x + y)z \xrightarrow{a} x'$, then

- either$_2$ $x + y \xrightarrow{a} \surd$ and $x' = z$. Then$_4$ $x \xrightarrow{a} \surd$ or $y \xrightarrow{a} \surd$. But then$_2$ $xz \xrightarrow{a} z$ or $yz \xrightarrow{a} z$ and thus$_5$ $xz + yz \xrightarrow{a} x'$. $R(x', x')$.
- or$_3$ $x + y \xrightarrow{a} y'$ and $x' = y'z$. Then$_5$ $x \xrightarrow{a} y'$ or $y \xrightarrow{a} y'$. But then$_3$ $xz \xrightarrow{a} y'z$ or $yz \xrightarrow{a} y'z$ and thus$_5$ $xz + yz \xrightarrow{a} x'$. $R(x', x')$.

Next, look at the transitions of the right-hand side. Suppose $xz + yz \xrightarrow{a} x'$, then either$_5$ $xz \xrightarrow{a} x'$ or $yz \xrightarrow{a} x'$. Note that this case is symmetric in x and y, so suppose $xz \xrightarrow{a} x'$. Then$_2$ $x \xrightarrow{a} \surd$ and $x' = z$ or$_3$ $x \xrightarrow{a} y'$ and $x' = y'z$.

- if $x \xrightarrow{\mathtt{a}} \surd$ and $x' = z$, then$_4$ $x + y \xrightarrow{\mathtt{a}} \surd$ and thus$_2$ $(x + y)z \xrightarrow{\mathtt{a}} x'$ and $R(x', x')$.
- if $x \xrightarrow{\mathtt{a}} y'$ and $x' = y'z$, then$_5$ $x + y \xrightarrow{\mathtt{a}} y'$ and thus$_3$ $(x + y)z \xrightarrow{\mathtt{a}} x'$ and $R(x', x')$.

Axiom A5 We have to show that $(xy)z \leftrightarrows x(yz)$.
Note that there is no possibility for either the left-hand side or the right-hand side to execute a terminating action. First we look at the transitions of the left-hand side. Suppose $(xy)z \xrightarrow{\mathtt{a}} x'$, then the only possibility is that$_3$ $xy \xrightarrow{\mathtt{a}} y'$ and $x' = y'z$. Then$_2$ $x \xrightarrow{\mathtt{a}} \surd$ and $y' = y$ or $x \xrightarrow{\mathtt{a}} z'$ and $y' = z'y$.

- if $x \xrightarrow{\mathtt{a}} \surd$ and $y' = y$, then $x' = yz$ and$_2$ $x(yz) \xrightarrow{\mathtt{a}} x'$. $R(x', x')$.
- if $x \xrightarrow{\mathtt{a}} z'$ and $y' = z'y$, then $x' = (z'y)z$ and$_3$ $x(yz) \xrightarrow{\mathtt{a}} z'(yz)$. Furthermore, $R(z'(yz), (z'y)z)$.

The proof of the right-hand side is analogous.

Axiom M1 We have to show that $x \between y \leftrightarrows x \between y + y \between x$.
We look at the transitions of both sides of the axiom at the same time, making a case distinction over the actions x can execute. Note that this axiom is symmetric in x and y.

- If $x \xrightarrow{\mathcal{U}_A} \surd$, then$_{39}$ $x \between y \xrightarrow{\mathcal{U}_A} \lfloor y \rfloor_A$, and$_{45}$ $x \between y \xrightarrow{\mathcal{U}_A} \lfloor y \rfloor_A$, so$_5$ $x \between y + y \between x \xrightarrow{\mathcal{U}_A} \lfloor y \rfloor_A$. Note that $R(\lfloor y \rfloor_A, \lfloor y \rfloor_A)$.
- If $x \xrightarrow{a_n} \surd$, then$_{40}$ $x \between y \xrightarrow{a_n} [y]_a$, and$_{46}$ $x \between y \xrightarrow{a_n} [y]_a$, so$_5$ $x \between y + y \between x \xrightarrow{a_n} [y]_a$. Note that $R([y]_a, [y]_a)$.
- If $x \xrightarrow{a} \surd$, then$_{41}$ $x \between y \xrightarrow{a} y$, and$_{47}$ $x \between y \xrightarrow{a} y$, so$_5$ $x \between y + y \between x \xrightarrow{a} y$. Note that $R(y, y)$.
- If $x \xrightarrow{\mathcal{U}_A} x'$, then$_{42}$ $x \between y \xrightarrow{\mathcal{U}_A} x' \between \lfloor y \rfloor_A$, and$_{48}$ $x \between y \xrightarrow{\mathcal{U}_A} x' \between \lfloor y \rfloor_A$, so$_5$ $x \between y + y \between x \xrightarrow{\mathcal{U}_A} x' \between \lfloor y \rfloor_A$. Note that $R(x' \between \lfloor y \rfloor_A, x' \between \lfloor y \rfloor_A)$.
- If $x \xrightarrow{a_n} x'$, then$_{43}$ $x \between y \xrightarrow{a_n} x' \between [y]_a$, and$_{49}$ $x \between y \xrightarrow{a_n} x' \between [y]_a$, so$_5$ $x \between y + y \between x \xrightarrow{a_n} x' \between [y]_a$. Note that $R(x' \between [y]_a, x' \between [y]_a)$.
- If $x \xrightarrow{a} x'$, then$_{44}$ $x \between y \xrightarrow{a} x' \between y$, and$_{50}$ $x \between y \xrightarrow{a} x' \between y$, so$_5$ $x \between y + y \between x \xrightarrow{a} x' \between y$. Note that $R(x' \between y, x' \between y)$.

Axiom M2 We have to show that $\mathcal{U}_A \between x \leftrightarrows \mathcal{U}_A \lfloor x \rfloor_A$.
We look at the transitions of both sides of the axiom at the same time. Since$_1$ $U_A \xrightarrow{\mathcal{U}_A} \surd$, we conclude that$_{45}$ $\mathcal{U}_A \between x \xrightarrow{\mathcal{U}_A} \lfloor x \rfloor_A$. Furthermore$_2$, $\mathcal{U}_A \lfloor x \rfloor_A \xrightarrow{\mathcal{U}_A} \lfloor x \rfloor_A$ and note that $R(\lfloor x \rfloor_A, \lfloor x \rfloor_A)$.

Axioms M3–M4 Similar to axiom M2, using deduction rules 46 and 47, respectively.

Axiom M5 We have to show that $\mathcal{U}_A x \between y \leftrightarrows \mathcal{U}_A(x \between \lfloor y \rfloor_A)$.
Note that neither side of the axiom can execute a terminating action. We look at the transitions of both sides of the axiom at the same time. $\mathcal{U}_A x \between y \xrightarrow{\mathtt{a}} z$ iff$_{48}$ $\mathcal{U}_A x \xrightarrow{\mathtt{a}} x'$ and $z = x' \between \lfloor y \rfloor_A$. Then$_2$ $\mathtt{a} = \mathcal{U}_A$ and $x' = x$, so $\mathcal{U}_A x \between y \xrightarrow{\mathcal{U}_A} x \between \lfloor y \rfloor_A$. Furthermore$_2$, $\mathcal{U}_A(x \between \lfloor y \rfloor_A) \xrightarrow{\mathcal{U}_A} x \between \lfloor y \rfloor_A$ and note that $R(x \between \lfloor y \rfloor_A, x \between \lfloor y \rfloor_A)$.

Axioms M6–M7 Similar to axiom M5, using deduction rules 49 and 50, respectively.

Axiom M8 We have to show that $(x + y) \between z \leftrightarrows x \between z + y \between z$.
First of all, note that this axiom is symmetric in x and y. First, we look at the transitions of the left-hand side. Suppose $(x + y) \between z \xrightarrow{\mathtt{a}} x'$. Then

- either$_{45}$ $x + y \xrightarrow{\mathcal{U}_A} \surd$ and $x' = \lfloor z \rfloor_A$. But then$_4$ $x \xrightarrow{\mathcal{U}_A} \surd$ or $y \xrightarrow{\mathcal{U}_A} \surd$, thus$_{45}$ $x \between z \xrightarrow{\mathcal{U}_A} \lfloor z \rfloor_A$ or $y \between z \xrightarrow{\mathcal{U}_A} \lfloor z \rfloor_A$ and thus$_5$ $x \between z + y \between z \xrightarrow{\mathcal{U}_A} \lfloor z \rfloor_A$. Note that $R(\lfloor z \rfloor_A, \lfloor z \rfloor_A)$.
- or$_{46}$ $x + y \xrightarrow{a_n} \surd$ and $x' = [z]_a$. But then$_4$ $x \xrightarrow{a_n} \surd$ or $y \xrightarrow{a_n} \surd$, thus$_{46}$ $x \between z \xrightarrow{a_n} [z]_a$ or $y \between z \xrightarrow{a_n} [z]_a$ and thus$_5$ $x \between z + y \between z \xrightarrow{a_n} [z]_a$. Note that $R([z]_a, [z]_a)$.

- or_{47} $x + y \xrightarrow{a} \surd$ and $x' = z$. But then_4 $x \xrightarrow{a} \surd$ or $y \xrightarrow{a} \surd$, thus_{47} $x \between z \xrightarrow{a} z$ or $y \between z \xrightarrow{a} z$ and thus_5 $x \between z + y \between z \xrightarrow{a} z$. Note that $R(z, z)$.
- or_{48} $x + y \xrightarrow{\mathcal{U}_A} y'$ and $x' = y' \lfloor z \rfloor_A$. But then_4 $x \xrightarrow{\mathcal{U}_A} y'$ or $y \xrightarrow{\mathcal{U}_A} y'$, thus_{48} $x \between z \xrightarrow{\mathcal{U}_A} y' \lfloor z \rfloor_A$ or $y \between z \xrightarrow{\mathcal{U}_A} y' \lfloor z \rfloor_A$ and thus_5 $x \between z + y \between z \xrightarrow{\mathcal{U}_A} y' \lfloor z \rfloor_A$. Note that $R(y' \lfloor z \rfloor_A, y' \lfloor z \rfloor_A)$.
- or_{49} $x + y \xrightarrow{a_n} y'$ and $x' = y' \lfloor [z]_a$. But then_4 $x \xrightarrow{a_n} y'$ or $y \xrightarrow{a_n} y'$, thus_{49} $x \between z \xrightarrow{a_n} y' \lfloor [z]_a$ or $y \between z \xrightarrow{a_n} y' \lfloor [z]_a$ and thus_5 $x \between z + y \between z \xrightarrow{a_n} y' \lfloor [z]_a$. Note that $R(y' \lfloor [z]_a, y' \lfloor [z]_a)$.
- or_{50} $x + y \xrightarrow{a} y'$ and $x' = y' \lfloor z$. But then_4 $x \xrightarrow{a} y'$ or $y \xrightarrow{a} y'$, thus_{50} $x \between z \xrightarrow{a} y' \lfloor z$ or $y \between z \xrightarrow{a} y' \lfloor z$ and thus_5 $x \between z + y \between z \xrightarrow{a} y' \lfloor z$. Note that $R(y' \lfloor z, y' \lfloor z)$.

The proof of the right-hand side is similar, again using a case distinction on the actions.

Axiom L1 We have to show that $[\mathcal{U}_B]_b \leftrightarrows \mathcal{U}_B$.
We look at the transitions of both sides at the same time. $\text{Observe}_{1,6}$ that either side can only do a $\mathcal{U}_B$-transition to $\surd$. No other transitions are possible.

Axioms L2–L4 These axioms are similar to L1, using rules 7, 8 and 9, respectively.

Axiom L5 We have to show that $[\mathbf{a}x]_b \leftrightarrows [\mathbf{a}]_b \cdot [x]_b$.
We look at the transitions of both sides at the same time. Note that neither the left-hand side nor the right-hand side can do a transition to $\surd$ since $x \cdot y$ cannot do a transition to $\surd$. We use a case distinction on the actions:

- $[\mathbf{a}x]_b \xrightarrow{\mathcal{U}_A} y$ iff_{10} $y = [x]_b$ and $\mathbf{a} = \mathcal{U}_A$.
  $[\mathbf{a}]_b \cdot [x]_b \xrightarrow{\mathcal{U}_A} y$ $\text{iff}_{6,2}$ $y = [x]_b$ and $\mathbf{a} = \mathcal{U}_A$. Note that $R([x]_b, [x]_b)$.

- Suppose, $a = b$. Then, $[\mathbf{a}x]_b \xrightarrow{a_{n+1}} y$ iff_{12} $y = [x]_b$ and $\mathbf{a} = a_n$.
  $[\mathbf{a}]_b \cdot [x]_b \xrightarrow{a_{n+1}} y$ $\text{iff}_{8,2}$ $y = [x]_b$ and $\mathbf{a} = a_n$. Note that $R([x]_b, [x]_b)$.

- Suppose, $a \neq b$. Then, $[\mathbf{a}x]_b \xrightarrow{a_n} y$ iff_{11} $y = [x]_b$ and $\mathbf{a} = a_n$.
  $[\mathbf{a}]_b \cdot [x]_b \xrightarrow{a_n} y$ $\text{iff}_{7,2}$ $y = [x]_b$ and $\mathbf{a} = a_n$. Note that $R([x]_b, [x]_b)$.

- $[\mathbf{a}x]_b \xrightarrow{a} y$ iff_{13} $y = [x]_b$ and $\mathbf{a} = a$.
  $[\mathbf{a}]_b \cdot [x]_b \xrightarrow{a} y$ $\text{iff}_{9,2}$ $y = [x]_b$ and $\mathbf{a} = a$. Note that $R([x]_b, [x]_b)$.

Axiom L6 We have to show that $[x + y]_b \leftrightarrows [x]_b + [y]_b$.
Note that this axiom is symmetric in x and y as a result of the commutativity of the alternative composition operator. First we have a look at the left-hand side. Suppose $[x + y]_b \xrightarrow{\mathbf{a}} \surd$, then_4

- either_6 $\mathbf{a} = \mathcal{U}_A$ and $x \xrightarrow{\mathcal{U}_A} \surd$ or $y \xrightarrow{\mathcal{U}_A} \surd$. But then_6 $[x]_b \xrightarrow{\mathcal{U}_A} \surd$ or $[y]_b \xrightarrow{\mathcal{U}_A} \surd$ and thus_4 $[x]_b + [y]_b \xrightarrow{\mathcal{U}_A} \surd$.
- or_7 $\mathbf{a} = a_n$, $a \neq b$ and $x \xrightarrow{a_n} \surd$ or $y \xrightarrow{a_n} \surd$. But then_7 $[x]_b \xrightarrow{a_n} \surd$ or $[y]_b \xrightarrow{a_n} \surd$ and thus_4 $[x]_b + [y]_b \xrightarrow{a_n} \surd$.
- or_8 $\mathbf{a} = a_{n+1}$, $a = b$, and $x \xrightarrow{a_n} \surd$ or $y \xrightarrow{a_n} \surd$. But then_8 $[x]_b \xrightarrow{a_{n+1}} \surd$ or $[y]_b \xrightarrow{a_{n+1}} \surd$ and thus_4 $[x]_b + [y]_b \xrightarrow{a_{n+1}} \surd$.
- or_9 $\mathbf{a} = a$ and $x \xrightarrow{a} \surd$ or $y \xrightarrow{a} \surd$. But then_9 $[x]_b \xrightarrow{a} \surd$ or $[y]_b \xrightarrow{a} \surd$ and thus_4 $[x]_b + [y]_b \xrightarrow{a} \surd$.

The proof of $[x + y]_b \xrightarrow{\mathbf{a}} z$ is similar to the proof of $[x + y]_b \xrightarrow{\mathbf{a}} \surd$, using axioms 5, 2 and 10–13.

Next, we have a look at the right-hand side. Suppose $[x]_b + [y]_b \xrightarrow{\mathbf{a}} \surd$, then_4 either $[x]_b \xrightarrow{\mathbf{a}} \surd$ or $[y]_b \xrightarrow{\mathbf{a}} \surd$. Since this axiom is symmetric in x and y, we suppose that $[x]_b \xrightarrow{\mathbf{a}} \surd$. Then,

- either$_6$ $\mathbf{a} = \mathcal{U}_A$ and $x \xrightarrow{\mathcal{U}_A} \surd$. But then$_4$ $x + y \xrightarrow{\mathcal{U}_A} \surd$ and thus$_6$ $[x+y]_b \xrightarrow{\mathcal{U}_A} \surd$.
- or$_7$ $\mathbf{a} = a_n$, $a \neq b$ and $x \xrightarrow{a_n} \surd$. But then$_4$ $x + y \xrightarrow{a_n} \surd$ and thus$_7$ $[x+y]_b \xrightarrow{a_n} \surd$.
- or$_8$ $\mathbf{a} = a_{n+1}$, $a = b$ and $x \xrightarrow{a_n} \surd$. But then$_4$ $x + y \xrightarrow{a_n} \surd$ and thus$_7$ $[x+y]_b \xrightarrow{a_{n+1}} \surd$.
- either$_9$ $\mathbf{a} = a$ and $x \xrightarrow{a} \surd$. But then$_4$ $x + y \xrightarrow{a} \surd$ and thus$_9$ $[x+y]_b \xrightarrow{a} \surd$.

The proof of $[x]_b + [y]_b \xrightarrow{\mathbf{a}} z$ is similar to the proof of $[x]_b + [y]_b \xrightarrow{\mathbf{a}} \surd$, using axioms 5, 2 and 10–13.

Axioms UL1–UL6 The proofs for axioms UL1 to UL6 are similar to the proofs for axioms L1 to L6, using rules 14–21 instead of rules 6–13.

Axiom TR1 We have to show that $\langle\langle x \rangle\rangle \leftrightarroweq \langle\langle x, \emptyset, x \rangle\rangle$.
We make a case distinction on the set of possible transitions $\langle\langle x \rangle\rangle$ can do and show that $\langle\langle x, \emptyset, x \rangle\rangle$ can do exactly the same transitions. Furthermore, we show that the right-hand side cannot do any other transitions.

- Suppose$_{23}$ $\langle\langle x \rangle\rangle \xrightarrow{\mathcal{U}_\emptyset} \mathcal{C}_\emptyset$. Then$_{23}$ $x \xrightarrow{\mathcal{U}_B} \surd$ and thus$_{26}$ $\langle\langle x, \emptyset, x \rangle\rangle \xrightarrow{\mathcal{U}_B} \mathcal{C}_\emptyset$. Note that $R(C_\emptyset, C_\emptyset)$.
- Suppose$_{24,25}$ $\langle\langle x \rangle\rangle \xrightarrow{a_0} \mathcal{C}_{\{a\}}$. Then either$_{24}$ $x \xrightarrow{a_0} \surd$ or$_{25}$ $x \xrightarrow{a} \surd$ and thus$_{27,29}$ $\langle\langle x, \emptyset, x \rangle\rangle \xrightarrow{a_0} \mathcal{C}_{\{a\}}$. Note that $R(C_{\{a\}}, C_{\{a\}})$.
- Suppose$_{31}$ $\langle\langle x \rangle\rangle \xrightarrow{\mathcal{U}_\emptyset} \langle\langle x, \emptyset, x' \rangle\rangle$. Then$_{31}$ $x \xrightarrow{\mathcal{U}_B} x'$ and thus$_{34}$ $\langle\langle x, \emptyset, x \rangle\rangle \xrightarrow{\mathcal{U}_\emptyset} \langle\langle x, \emptyset, x' \rangle\rangle$. Note that $R(\langle\langle x, \emptyset, x' \rangle\rangle, \langle\langle x, \emptyset, x' \rangle\rangle)$.
- Suppose$_{32,33}$ $\langle\langle x \rangle\rangle \xrightarrow{a_0} \langle\langle x, \{a\}, x' \rangle\rangle$. Then either$_{32}$ $x \xrightarrow{a_0} x'$ or$_{33}$ $x \xrightarrow{a} x'$ and therefore$_{35,37}$ $\langle\langle x, \emptyset, x \rangle\rangle \xrightarrow{a_0} \langle\langle x, \{a\}, x' \rangle\rangle$. Note that $R(\langle\langle x, \{a\}, x' \rangle\rangle, \langle\langle x, \{a\}, x' \rangle\rangle)$.

So $\langle\langle x, \emptyset, x \rangle\rangle$ can do all transitions $\langle\langle x \rangle\rangle$ can do. It can be easily verified that the right-hand side cannot do any other transitions since for all other transitions (using deduction rule 22, 28 or 36) $A \neq \emptyset$ or $a \in A$ is needed. However, $A = \emptyset$.

Axiom TR2 We have to show that $\langle\langle x, \emptyset, \mathcal{U}_B \rangle\rangle \leftrightarroweq \mathcal{U}_\emptyset \cdot \mathcal{C}_\emptyset$.
First of all, note that neither the left-hand side nor the right-hand side can do a transition to $\surd$. Suppose $\langle\langle x, \emptyset, \mathcal{U}_B \rangle\rangle \xrightarrow{\mathbf{a}} x'$. Then$_{26}$ the only possibility is that $x' = \mathcal{C}_\emptyset$ and $\mathbf{a} = \mathcal{U}_\emptyset$, so $\langle\langle x, \emptyset, \mathcal{U}_B \rangle\rangle \xrightarrow{\mathcal{U}_\emptyset} \mathcal{C}_\emptyset$. Furhtermore$_2$, $\mathcal{U}_\emptyset \cdot \mathcal{C}_\emptyset$ can only do a $\mathcal{U}_\emptyset$-transition to $\mathcal{C}_\emptyset$, and note that $R(\mathcal{C}_\emptyset, \mathcal{C}_\emptyset)$.

Axiom TR3 We have to show that if $A \neq \emptyset$, $\langle\langle x, A, \mathcal{U}_B \rangle\rangle \leftrightarroweq \mathcal{U}_\emptyset \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$.
Let $A \neq \emptyset$. We look at both sides of the axiom at the same time. Since $A \neq \emptyset$, $\langle\langle x, A, \mathcal{U}_B \rangle\rangle$ can do a $\mathcal{R}_A$-transition$_{22}$ to $\langle\langle x \rangle\rangle$. $\mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do this transition too, and thus$_5$, $\mathcal{U}_\emptyset \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do a $\mathcal{R}_A$-transition to $\langle\langle x \rangle\rangle$. Note that $R(\langle\langle x \rangle\rangle, \langle\langle x \rangle\rangle)$.

For the rest, $\langle\langle x, A, \mathcal{U}_B \rangle\rangle$ can do a $\mathcal{U}_\emptyset$-transition$_{26}$ to $\mathcal{C}_A$. This is exactly the only other transition $\mathcal{U}_\emptyset \cdot \mathcal{C}_A + \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do.

Axiom TR4 We have to show that if $n > 0$ and $A \neq \emptyset$, $\langle\langle x, A, a_n \rangle\rangle \leftrightarroweq \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$.
Let $n > 0$ and $A \neq \emptyset$. Since $A \neq \emptyset$, $\langle\langle x, A, \mathcal{U}_B \rangle\rangle$ can do a $\mathcal{R}_A$-transition$_{22}$ to $\langle\langle x \rangle\rangle$. $\mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do (only) this transition, and note that $R(\langle\langle x \rangle\rangle, \langle\langle x \rangle\rangle)$. Since $n > 0$, no other transitions are possible for $\langle\langle x, A, a_n \rangle\rangle$.

Axiom TR5 We have to show that $\langle\langle x, \emptyset, a_0 \rangle\rangle \leftrightarroweq a_0 \cdot \mathcal{C}_{\{a\}}$.
The right-hand side can only do a a_0-transition to $\mathcal{C}_{\{a\}}$. Furthermore$_{27}$, $\langle\langle x, \emptyset, a_0 \rangle\rangle \xrightarrow{a_0} \mathcal{C}_{\{a\}}$ and $R(\mathcal{C}_{\{a\}}, \mathcal{C}_{\{a\}})$. Note that rule 27 is the only deduction rule that can be applied to $\langle\langle x, \emptyset, a_0 \rangle\rangle$.

Axiom TR6 We have to show that if $a \notin A$ and $A \neq \emptyset$, then $\langle\langle x, A, a_0 \rangle\rangle \leftrightarroweq a_0 \cdot \mathcal{C}_{A \cup \{a\}} + \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$.
Suppose $a \notin A$ and $A \neq \emptyset$. We look at both sides of the axiom at the same time. Since $A \neq \emptyset$, $\langle\langle x, A, a_0 \rangle\rangle$ can do a $\mathcal{R}_A$-transition$_{22}$ to $\langle\langle x \rangle\rangle$. $\mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do this transition too, and thus$_5$, $a_0 \cdot \mathcal{C}_{A \cup \{a\}} + \mathcal{R}_A \cdot \langle\langle x \rangle\rangle$ can do a $\mathcal{R}_A$-transition to $\langle\langle x \rangle\rangle$. Note that $R(\langle\langle x \rangle\rangle, \langle\langle x \rangle\rangle)$.

Furthermore$_{27}$, $\langle\!\langle x, A, a_0\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$. $a_0 \cdot \mathcal{C}_{A\cup\{a\}}$ can do this transition too, and thus$_5$, $a_0 \cdot \mathcal{C}_{A\cup\{a\}} + \mathcal{R}_A \cdot \langle\!\langle x\rangle\!\rangle$ can do an a_0-transition to $\mathcal{C}_{A\cup\{a\}}$. Note that $R(\mathcal{C}_{A\cup\{a\}}, \mathcal{C}_{A\cup\{a\}})$. It can be verified that rules 22 and 27 are the only deduction rules that can be applied to $\langle\!\langle x, A, a_0\rangle\!\rangle$ and that the two transitions given for $a_0 \cdot \mathcal{C}_{A\cup\{a\}} + \mathcal{R}_A \cdot \langle\!\langle x\rangle\!\rangle$ are the only two transitions possible.

Axiom TR7 The proof of axiom TR7 is similar to the proof of axiom TR6, using deduction rule 28 instead of 27.

Axioms TR8–10 The proofs for axioms TR8 to TR10 are similar to the proofs for axioms TR5 to TR7, using rules 29 and 30 instead of rules 27 and 28.

Axioms TR11–19 The proofs for axioms TR11 to TR19 are similar to the proofs for axioms TR2 to TR10, using rules 34–38 instead of rules 26–30.

Axiom TR20 We have to show that $\langle\!\langle x, A, y + z\rangle\!\rangle \leftrightarrows \langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle$.
First of all, a rollback can take place iff $A \neq \emptyset$: $\langle\!\langle x, A, y+z\rangle\!\rangle \xrightarrow{\mathcal{R}_A} \langle\!\langle x\rangle\!\rangle$ iff$_{22}$ $A \neq \emptyset$ iff$_{22}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{\mathcal{R}_A} \langle\!\langle x\rangle\!\rangle$ and $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{R}_A} \langle\!\langle x\rangle\!\rangle$ iff$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{R}_A} \langle\!\langle x\rangle\!\rangle$. And note, $R(\langle\!\langle x\rangle\!\rangle, \langle\!\langle x\rangle\!\rangle)$.

Next, we look at the transitions of the left-hand side. Suppose $\langle\!\langle x, A, y + z\rangle\!\rangle \xrightarrow{\mathbf{a}} x'$. Then

- either$_{26}$ $y+z \xrightarrow{\mathcal{U}_B} \surd$, $\mathbf{a} = \mathcal{U}_\emptyset$ and $x' = \mathcal{C}_A$. But then$_4$ $y \xrightarrow{\mathcal{U}_B} \surd$ or $z \xrightarrow{\mathcal{U}_B} \surd$, thus$_{26}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \mathcal{C}_A$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \mathcal{C}_A$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \mathcal{C}_A$. Note that $R(\mathcal{C}_A, \mathcal{C}_A)$.
- or$_{27}$ $y + z \xrightarrow{\mathbf{a}_0} \surd$, $a \notin A$, $\mathbf{a} = a_0$ and $x' = \mathcal{C}_{A\cup\{a\}}$. But then$_4$ $y \xrightarrow{a_0} \surd$ or $z \xrightarrow{a_0} \surd$, thus$_{27}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$. Note that $R(\mathcal{C}_{A\cup\{a\}}, \mathcal{C}_{A\cup\{a\}})$.
- or$_{28}$ $y + z \xrightarrow{a_0} \surd$, $a \in A$, $\mathbf{a} = a$ and $x' = \mathcal{C}_A$. But then$_4$ $y \xrightarrow{a_0} \surd$ or $z \xrightarrow{a_0} \surd$, thus$_{28}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$. Note that $R(\mathcal{C}_A, \mathcal{C}_A)$.
- or$_{29}$ $y + z \xrightarrow{a} \surd$, $a \notin A$, $\mathbf{a} = a_0$ and $x' = \mathcal{C}_{A\cup\{a\}}$. But then$_4$ $y \xrightarrow{a} \surd$ or $z \xrightarrow{a} \surd$, thus$_{29}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \mathcal{C}_{A\cup\{a\}}$. Note that $R(\mathcal{C}_{A\cup\{a\}}, \mathcal{C}_{A\cup\{a\}})$.
- or$_{30}$ $y+z \xrightarrow{a} \surd$, $a \in A$, $\mathbf{a} = a$ and $x' = \mathcal{C}_A$. But then$_4$ $y \xrightarrow{a} \surd$ or $z \xrightarrow{a} \surd$, thus$_{28}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \mathcal{C}_A$. Note that $R(\mathcal{C}_A, \mathcal{C}_A)$.
- or$_{34}$ $y + z \xrightarrow{\mathcal{U}_B} y'$, $\mathbf{a} = \mathcal{U}_\emptyset$ and $x' = \langle\!\langle x, A, y'\rangle\!\rangle$. But then$_4$ $y \xrightarrow{\mathcal{U}_B} y'$ or $z \xrightarrow{\mathcal{U}_B} y'$, thus$_{26}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \langle\!\langle x, A, y'\rangle\!\rangle$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \langle\!\langle x, A, y'\rangle\!\rangle$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{\mathcal{U}_\emptyset} \langle\!\langle x, A, y'\rangle\!\rangle$. Note that $R(\langle\!\langle x, A, y'\rangle\!\rangle, \langle\!\langle x, A, y'\rangle\!\rangle)$.
- or$_{35}$ $y + z \xrightarrow{a_0} y'$, $a \notin A$, $\mathbf{a} = a_0$ and $x' = \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$. But then$_4$ $y \xrightarrow{a_0} y'$ or $z \xrightarrow{a_0} y'$, thus$_{35}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$. Note that $R(\langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle, \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle)$.
- or$_{36}$ $y + z \xrightarrow{a_0} y'$, $a \in A$, $\mathbf{a} = a$ and $x' = \langle\!\langle x, A, y'\rangle\!\rangle$. But then$_4$ $y \xrightarrow{a_0} y'$ or $z \xrightarrow{a_0} y'$, thus$_{36}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$. Note that $R(\langle\!\langle x, A, y'\rangle\!\rangle, \langle\!\langle x, A, y'\rangle\!\rangle)$.
- or$_{37}$ $y + z \xrightarrow{a} y'$, $a \notin A$, $\mathbf{a} = a_0$ and $x' = \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$. But then$_4$ $y \xrightarrow{a} y'$ or $z \xrightarrow{a} y'$, thus$_{37}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a_0} \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle$. Note that $R(\langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle, \langle\!\langle x, A \cup \{a\}, y'\rangle\!\rangle)$.
- or$_{38}$ $y + z \xrightarrow{a} y'$, $a \in A$, $\mathbf{a} = a$ and $x' = \langle\!\langle x, A, y'\rangle\!\rangle$. But then$_4$ $y \xrightarrow{a} y'$ or $z \xrightarrow{a} y'$, thus$_{38}$ $\langle\!\langle x, A, y\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$ or $\langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$ and thus$_5$ $\langle\!\langle x, A, y\rangle\!\rangle + \langle\!\langle x, A, z\rangle\!\rangle \xrightarrow{a} \langle\!\langle x, A, y'\rangle\!\rangle$. Note that $R(\langle\!\langle x, A, y'\rangle\!\rangle, \langle\!\langle x, A, y'\rangle\!\rangle)$.

The proof of the right-hand side is similar, again using a case distinction on the actions.

□

A.2 Proving elimination to BPArec

Theorem A.2 (Eliminiation to BPArec) *For every* PAtrans *term* t *there exists a guarded recursive specification* E *over* BPA *such that* t *is a solution of* E.

Proof This theorem is proven by induction on the general structure of t.

1. $t \equiv \mathbf{a}$ for $\mathbf{a} \in \mathbb{A} \cup \mathbb{L} \cup \mathbb{UL}$. Then t is a finite BPArec term.

2. $t \equiv t_1 + t_2$ for PAtrans terms t_1 and t_2. By induction, there are finite guarded BPArec terms s_1 and s_2 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = t_1$ and $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_2 = t_2$. But then also $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 + s_2 = t_1 + t_2$ and $s_1 + s_2$ is a finite BPArec term.

3. $t = t_1 \cdot t_2$ for PAtrans terms t_1 and t_2. This case is treated analogous to case 2.

4. $t \equiv [t_1]_b$ for PAtrans term t_1 and $b \in \mathbb{A}$. By induction, there exists a finite guarded BPArec term s_1 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = t_1$. Since all recursive specifications in PAtrans are guarded, there is a BPArec term r_1 which is in head normal form, such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models r_1 = s_1 = t_1$. We prove this case by induction on the structure of r_1:

   - $r_1 \equiv \mathcal{U}_A$ for $A \subseteq \mathbb{A}$. Then PAtrans $\vdash t = [\mathcal{U}_A]_b = \mathcal{U}_A$ and $\mathcal{U}_A$ is a finite BPArec term since $\mathcal{U}_A \in \mathbb{A}_{\text{BPArec}}$.
   - $r_1 \equiv a_n$ for $a \in \mathbb{A}$ and $n \in \mathbb{N}, n \geq 0$. Then, depending on whether $a = b$, PAtrans $\vdash t = [a_n]_a = a_{n+1}$ or, if $a \neq b$, PAtrans $\vdash t = [a_n]_b = a_n$. Both a_{n+1} and a_n are finite BPArec terms.
   - $r_1 \equiv a$ for $a \in \mathbb{A}$. Then PAtrans $\vdash t = [a]_b = a$ and a is a finite BPArec term.
   - $r_1 \equiv \mathbf{a} \cdot r_2$ for $\mathbf{a} \in \mathbb{A}_{\text{BPArec}}$ and BPArec term r_2. Then PAtrans $\vdash t = [\mathbf{a} \cdot r_2]_b = [\mathbf{a}]_b \cdot [r_2]_b$. By induction there exist finite guarded BPArec terms p_1 and p_2 such that PAtrans $\vdash p_1 = [\mathbf{a}]_b$ and PAtrans $\vdash p_2 = [r_2]_b$. Then also $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = [\mathbf{a}]_b \cdot [r_2]_b = p_1 \cdot p_2$ and $p_1 \cdot p_2$ is a finite linear BPArec term.
   - $r_1 \equiv r_2 + r_3$ for r_2 and r_3 closed BPArec terms. Then PAtrans $\vdash t = [r_2 + r_3]_b = [r_2]_b + [r_3]_b$. By induction there exist finite guarded BPArec terms p_2 and p_3 such that PAtrans $\vdash p_2 = [r_2]_b$ and PAtrans $\vdash p_3 = [r_3]_b$. Then also $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = [r_2]_b + [r_3]_b = p_1 + p_2$ and $p_1 + p_2$ is a finite guarded BPArec term.
   - $r_1 \equiv X$ for some recursion variable X. This case is not possible since r_1 should be guarded.

5. $t \equiv \lfloor t_1 \rfloor_A$ for PAtrans term t_1 and $A \subseteq \mathbb{A}$. This case is treated analogous to case 4.

6. $t \equiv t_1 \between t_2$ for PAtrans terms t_1 and t_2. By induction, there exists a finite guarded BPArec term s_1 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = t_1$. But since all recursive specifications in PAtrans are guarded, there is a BPArec term r_1 which is in head normal form, such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models r_1 = s_1 = t_1$. We prove this case by induction on the structure of r_1:

   - $r_1 \equiv \mathcal{U}_A$ for $A \subseteq \mathbb{A}$. Then PAtrans $\vdash t = \mathcal{U}_A \between t_2 = \mathcal{U}_A \lfloor t_2 \rfloor_A$. We proved (5) that there exists a finite guarded BPArec term s_2 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_2 = \lfloor t_2 \rfloor_A$. Since $\mathcal{U}_A$ is in $\mathbb{A}_{\text{BPArec}}$, there exists a finite guarded BPArec term s_3 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = \mathcal{U}_A \lfloor t_2 \rfloor_A = s_3$, viz. $s_3 = \mathcal{U}_A \cdot s_2$.
   - $r_1 \equiv a_n$ for $a \in \mathbb{A}$ and $n \in \mathbb{N}, n \geq 0$. Then PAtrans $\vdash t = a_n \between t_2 = a_n [t_2]_a$. We proved (4) that there exists a finite guarded BPArec term s_2 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_2 = [t_2]_a$. Since a_n is in $\mathbb{A}_{\text{BPArec}}$, there exists a finite guarded BPArec term s_3 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = a_n [t_2]_a = s_3$, viz. $s_3 = a_n \cdot s_2$.

- $r_1 \equiv a$ for $a \in \mathbb{A}$. Then PAtrans $\vdash t = a \between t_2 = a \cdot t_2$. By induction there exists a finite guarded BPArec term s_2 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_2 = t_2$. Since a is in $\mathbb{A}_{\text{BPArec}}$, there exists a finite guarded BPArec term s_3 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = a \cdot t_s = a \cdot s_2 = s_3$, viz. $s_3 = a \cdot s_2$.
- $r_1 \equiv \mathbf{a} \cdot r_2$ for $\mathbf{a} \in \mathbb{A}_{\text{BPArec}}$ and BPArec term r_2. Then PAtrans $\vdash t = (\mathbf{a} \cdot r_2) \between t_2$. Depending on the structure of $\mathbf{a}$, PAtrans $\vdash t = \mathcal{U}_A(r_2 \| \lfloor t_2 \rfloor_A)$, PAtrans $\vdash t = \mathbf{a}_n(r_2 \| \lceil t_2 \rceil_a)$ or PAtrans $\vdash t = a(r_2 \| t_2)$. So there is a function f such that PAtrans $\vdash t = (\mathbf{a} \cdot r_2) \between t_2 = \mathbf{a} \cdot (r_2 \| f(t_2))$ where $f(x) \in \{\lfloor x \rfloor_A, \lceil x \rceil_a, x\}$. We proved (4,5) that $f(t_2)$ is a finite guarded BPArec term. Furthermore, PAtrans $\vdash r_2 \| f(t_2) = r_2 \between f(t_2) + f(t_2) \between r_2$ and, by induction, both $r_2 \between f(t_2)$ and $f(t_2) \between r_2$ are finite guarded BPArec terms and thus $r_2 \| f(t_2)$ is a finite guarded BPArec term. Since $\mathbf{a}$ is in $\mathbb{A}_{\text{BPArec}}$, $\mathbf{a} \cdot (r_2 \| f(t_2))$ and thus $(\mathbf{a} \cdot r_2) \between t_2$ is a finite guarded BPArec term.
- $r_1 \equiv r_2 + r_3$ for r_2 and r_3 BPArec terms. Then PAtrans $\vdash t = (r_2 + r_3) \between t_2 = r_2 \between t_2 + r_3 \between t_2$. By induction there exist finite guarded BPArec terms s_2 and s_3 such that PAtrans $\vdash s_2 = r_2 \between t_2$ and PAtrans $\vdash s_3 = r_3 \between t_2$. Then also $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t = (r_2 + r_3) \between t_2 = r_2 \between t_2 + r_3 \between t_2 = s_2 + s_3$ and $s_2 + s_3$ is a finite guarded BPArec term.
- $r_1 \equiv X$ for some recursion variable X. This case is not possible since r_1 should be guarded.

7. $t \equiv t_1 \| t_2$ for PAtrans terms t_1 and t_2. PAtrans $\vdash t_1 \| t_2 = t_1 \| t_2 = t_1 \between t_2 + t_2 \between t_1$. We have proved (6) that there exist finite guarded BPArec terms s_1 and s_2 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = t_1 \between t_2$ and $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_2 = t_2 \between t_3$. But then, $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models t_1 \| t_2 = t_1 \between t_2 + t_2 \between t_1 = s_1 + s_2$ and $s_1 + s_2$ is a finite guarded BPArec term.

8. $t \equiv \langle\langle t_1 \rangle\rangle$ for closed PAtrans term t_1. PAtrans $\vdash \langle\langle t_1 \rangle\rangle = \langle\langle t_1, \emptyset, t_1 \rangle\rangle$. As will be proven in 9, there exists a finite guarded BPArec term s_1 such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = \langle\langle t_1, \emptyset, t_1 \rangle\rangle$ and thus $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s_1 = \langle\langle t_1 \rangle\rangle$.

9. $t \equiv \langle\langle t_1, A, t_2 \rangle\rangle$ for PAtrans terms t_1 and t_2 and $A \subseteq \mathbb{A}$. We give a set of recursive equations, E, and prove by induction on the structure of t_2 that for all terms $\langle\langle t_1, A, t_2 \rangle\rangle$ there exists a BPArec term s such that $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models s = \langle\langle t_1, A, t_2 \rangle\rangle$. Let E, the set of recursive equations, be defined as follows:

$$
\begin{aligned}
E = \{\ & X^{t,\emptyset}_{\mathcal{U}_B} = \mathcal{U}_\emptyset \cdot \mathcal{C}_\emptyset, && X^{t,A}_{\mathcal{U}_B} = \mathcal{U}_\emptyset \cdot \mathcal{C}_A + \mathcal{R}_A \cdot X^{t,\emptyset}_t,\\
& X^{t,A}_{c_n} = \mathcal{R}_A \cdot X^{t,\emptyset}_t && X^{t,\emptyset}_{b_0} = b_0 \cdot \mathcal{C}_{\{b\}},\\
& X^{t,A}_{b_0} = b_0 \cdot \mathcal{C}_{A\cup\{b\}} + \mathcal{R}_A \cdot X^{t,\emptyset}_t, && X^{t,A}_{a_0} = a \cdot \mathcal{C}_A + \mathcal{R}_A \cdot X^{t,\emptyset}_t,\\
& X^{t,\emptyset}_b = b_0 \cdot \mathcal{C}_{\{b\}}, && X^{t,A}_b = b_0 \cdot \mathcal{C}_{A\cup\{b\}} + \mathcal{R}_A \cdot X^{t,\emptyset}_t,\\
& X^{t,A}_a = a \cdot \mathcal{C}_A + \mathcal{R}_A \cdot X^{t,\emptyset}_t, && X^{t,\emptyset}_{\mathcal{U}_B \cdot t_1} = \mathcal{U}_\emptyset \cdot X^{t,\emptyset}_{t_1},\\
& X^{t,A}_{\mathcal{U}_B \cdot t_1} = \mathcal{U}_\emptyset \cdot X^{t,A}_{t_1} + \mathcal{R}_A \cdot X^{t,\emptyset}_t && X^{t,\emptyset}_{c_n \cdot t_1} = \mathcal{R}_\emptyset \cdot X^{t,\emptyset}_t,\\
& X^{t,\emptyset}_{c_0 \cdot t_1} = c_0 \cdot X^{t,\{c\}}_{t_1}, && X^{t,A}_{b_0 \cdot t_1} = b_0 \cdot X^{t,A\cup\{b\}}_{t_1} + \mathcal{R}_A \cdot X^{t,\emptyset}_t,\\
& X^{t,A}_{a_0 \cdot t_1} = a \cdot X^{t,A}_{t_1} + \mathcal{R}_A \cdot X^{t,\emptyset}_t, && X^{t,\emptyset}_{c \cdot t_1} = c_0 \cdot X^{t,\{c\}}_{t_1},\\
& X^{t,A}_{b \cdot t_1} = b_0 \cdot X^{t,A\cup\{b\}}_{t_1} + \mathcal{R}_A \cdot X^{t,\emptyset}_t, && X^{t,A}_{a \cdot t_1} = a \cdot X^{t,A}_{t_1} + \mathcal{R}_A \cdot X^{t,\emptyset}_t,\\
& X^{t,B}_{t_1 + t_2} = X^{t,B}_{t_1} + X^{t,B}_{t_2} &&\\
& \mid A \subseteq \mathbb{A},\ A \neq \emptyset,\ B \subseteq \mathbb{A},\ n \in \mathbb{N},\ n > 0,\ a \in A,\ b \in \mathbb{A},\ b \notin A,\ c \in \mathbb{A} \quad \}
\end{aligned}
$$

As can be easily seen by comparing axioms TR2–20 with the recursive equations in E, $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models \langle X^{t_1,A}_{t_2} | E \rangle = \langle\langle t_1, A, t_2 \rangle\rangle$ holds for all t_1, A and t_2. So there exists a BPArec term for every $\langle\langle t_1, A, t_2 \rangle\rangle$ in PAtrans, viz. $\langle X^{t_1,A}_{t_2} | E \rangle$. Furthermore, since PAtrans $\vdash \langle\langle t \rangle\rangle = \langle\langle t, \emptyset, t \rangle\rangle$, $T(\text{PAtrans}) + \text{RDP}^- + \text{RSP} \models \langle\langle t \rangle\rangle = \langle X^{t,\emptyset}_t | E \rangle$.

So for all PAtrans terms t there exists a finite guarded BPArec term s such that PAtrans $+ \text{RDP}^- + \text{RSP} \models s = t$. □