A Mobile Calculus with Data

Thomas Gehrke, Arend Rensink

Informatik-Bericht Nr. 99-04
Oktober 1999

Abteilung Programmierung
Technische Universität Braunschweig
Gaußstraße 11
D-38092 Braunschweig/Germany

A Mobile Calculus with Data

Thomas Gehrke, Arend Rensink
Institut für Software
Abteilung Programmierung
Technische Universität Braunschweig
Gaußstraße 11
D-38092 Braunschweig/Germany

A Mobile Calculus with Data

Thomas Gehrke*, Arend Rensink°

* Technische Universität Braunschweig, Institut für Software, Abteilung Programmierung
Postfach 3329, D-38023 Braunschweig, Germany
gehrke@ips.cs.tu-bs.de

° Universiteit Twente, Department of Computer Science
P.O. Box 217, NL-7500 AE Enschede, The Netherlands
rensink@cs.utwente.nl

Abstract. *Process algebras* are a widely used formalism for the specification of the structure and the behaviour of reactive systems. Some process algebras, e.g. the *pi-calculus* and the *fusion calculus*, allow for *mobility*, i.e. it is possible to change the system structure dynamically by sending channel names in communication actions. These calculi abstract from other data than channel names; therefore, the description of the data aspects of reactive systems is not supported.
In this paper we introduce a mobile calculus, in which the channel names are replaced by expressions of a typed data language; channels are realized as values of a specific datatype. In order to allow for appropiate data descriptions, we do not fix the data language; it is possible to define expressions in any language which satisfies a small set of conditions.
The semantics of our calculus is given by transition rules. We define *strong* and *weak bisimulation* as equivalence relations on terms. Furthermore, an axiomatic semantics for both relations is given. As an example for the integration of data, we combine the calculus with an extended typed lambda calculus and apply it in a case study.

Computer Science Report 99-04

Technical University of Braunschweig
Institute for Software
Programming Department

* This author was supported by the DFG grant Go 671/2-2 *EREAS* (Entwurf reaktiver Systeme).

1 Introduction

Reactive and distributed systems are of increasing importance in theory and practice of computer science. These systems can be described by three characteristics: structure, behaviour and data. For the specification of the first two aspects the formalism of *process algebras* [3, 18, 20, 21] is widely used. Process algebras provide a powerful theory on behavioural preorders and equivalences and allow for formal reasoning on correctness issues, but usually they are weaker on the treatment of data. In order to include the data aspect into system specifications, in the recent years languages like *Concurrent ML* [26], *Facile* [11], and *ProFun* [10] have been developed, which combine the paradigms of process algebras and functional programming languages. The semantic treatment of such concurrent functional languages is not obvious; some approaches are described in [7, 8, 29].

Some process algebras, e.g. the π-calculus [21] and the *fusion*-calculus [24], allow for the sending of channels names in communication actions. This feature, called *mobility*, allows for the specification of evolving system structures by creating new connections between system components. In these algebras, there is no difference between values and identifiers: *names* both represent the channel values used in communication actions and the identifiers to which channels can be bound. Therefore, these calculi do not explicitly provide other data types than channels. (However, other data types can be encoded using channels, as for instance in Pict [25].)

In this paper, we introduce a mobile calculus, which allows for the specification of the behavioural and data aspects of systems. In this calculus, the names known from the π-calculus are replaced by expressions of a typed data language; channels are represented as values of a specific polymorphic datatype *channel*. The data language can provide the whole range of traditional types as well, thus allowing an adequate specification of the data manipulations occuring during system runs. The calculus does not fix a special data language; any typed language which satisfies a small set of conditions can be used to formulate expressions. Furthermore, in contrast to the π-calculus, we strictly distinguish between binding operations and restriction of channel values. Binding operations are applied to identifiers, whereas restriction is used to define a scope for channel values.

In most process algebras, concurrency is realised by a binary operator $t|u$, which represents the parallel composition of the processes t and u. On the other hand, concurrent programming languages like *Java* [2] and the above mentioned languages such as Concurrent ML as well as concurrency libraries for sequential languages like *C++* [27, 28] rather rely on an unary operator to create or *spawn* a new process, which then runs concurrently to the remainder of the program. In order to allow specifications which reflect this model of concurrency, the calculus presented here makes use of a similar *spawn*-operation for process creation, inspired by [4, 17].

Process creation is used in combination with an operator for the *sequential composition* of subprograms, which again is in contrast to (in fact, a generalisation of) the *action prefix operator* seen in most process algebras. The sequential composition of terms allows to express complex systems in a more appropiate way than action prefixing, because it is not necessary to handle termination of subsystems explicitly in the specification (instead, it is dealt with implicitly by the semantics).

Independent interest in sequential composition exists from the area of *action refinement*, see e.g. [12, 14]. Action refinement allows for the stepwise construction of of reactive systems. Single communication actions are replaced by process terms, which describe the behaviour of these actions in more detail. The notion of action refinement, in its syntactic interpretation as substitution within terms, calls for sequential composition rather than action prefixing. For example, if in a term $a.b.\mathbf{0}$, the action a should be refined by a term t, there is no obvious way to denote the resulting behaviour $t.b.\mathbf{0}$ without resorting to sequential composition.

The remainder of the paper is structured as follows: First we introduce in section 2 the syntax and the type system of the process calculus. In section 3 we define its operational semantics. Furthermore, in 4 we define strong bisimulation as an equivalence relation on

process terms and consider weak bisimulation as an equivalence which abstracts from the internal steps of processes. As an example for the integration of a data language, we combine the process calculus in section 5 with a typed lambda calculus and apply the calculus in an example. Section 6 contains some concluding remarks and discusses related work.

For readability, the proofs of the theorems and propositions are omitted from the main papers; they can be found in the appendix.

2 The Process Calculus

In this section, we present the syntax of our calculus. We first discuss the constraints we impose on the data sub-language (without restricting ourselves to a particular language at this point), subsequently discuss the behavioural part of the language, and finally give a type system for the language.

Data sublanguage. The process calculus should be able to deal with several data languages. Therefore, instead of defining a concrete data language, any language can be used which fulfils the assumptions listed below. These are entirely standard (see, e.g., [22, 19]), with the exception of the *channel* types which we assume to exist.

We assume a set of variable identifiers VAR, ranged over by $x, y, \ldots$; furthermore, we assume that the data language gives rise to a set of *expressions* EXPR, ranged over by $E, F, \ldots$. Expressions may contain *free variables*, i.e., elements of VAR syntactically occurring within the expression and not bound by any encompassing abstraction. The free variables of E ($\in$ EXPR) are collected into $fv(E)$ ($\subseteq$ VAR). If $fv(E) = \emptyset$, we call E *closed*. Among the closed expressions are the *values*, collected in VALUE $\subseteq$ EXPR and ranged over by $v, v', \ldots$; values are "basic" expressions, intuitively denoting constants; e.g., denotations for the natural numbers.

We expect the data sub-language to be *typed*. That is, we assume a set of *data types* DTYPE, ranged over by T; DTYPE is assumed to contain at least the type *bool* for the truth values *true* and *false*, and for every type $T \in$ DTYPE a type T *channel* for data channels over which values of type T can be communicated. The typing of expressions is given, as usual, through type judgements of the form $\Delta \vdash E : T$, where Δ is a list of *type assumptions* of the form $x : T'$, expressing that variable x (which presumably occurs free in E) is assumed to have type T'; under these assumptions, E has type T. Not every expression has an associated type judgement; those that have one (i.e., can be typed) are called *well-typed*. The set of values of type T is denoted VALUE^T, and the set of all channel values is denoted CHAN, ranged over by $a, b, c, \ldots$. The channel values occurring syntactically in an expression E are collected into $fc(E)$.

Finally, we assume that the data language has an associated reduction semantics. That is, an arbitrary closed well-typed expression E can be reduced to a value $[\![E]\!] \in$ VALUE, which constitutes, in a sense, the meaning of E. If $\vdash E : T$ then $[\![E]\!] \in \text{VALUE}^T$, i.e., reduction preserves the expression type. Based on the reduction semantics, we also assume a notion of β-equivalence: $E =_\beta F$ for well-typed E and F expresses that E and F are essentially the same. In particular, $[\![E]\!] = [\![F]\!]$ implies $E =_\beta F$ (although the reverse is not necessarily true, especially for higher-order values). Moreover, we require that $E =_\beta F$ implies $fc(E) = fc(F)$; we need this requirement for our operational semantics to be consistent.

Vectors of variables, expressions and types are denoted $\mathbf{x}$, $\mathbf{E}$ and $\mathbf{T}$, respectively; we sometimes write $\mathbf{x} : \mathbf{T}$ to denote a list of typed variables. $|\mathbf{x}|$ denotes the length of vector $\mathbf{x}$, and $\{\mathbf{x}\}$ denotes the set of elements occurring in $\mathbf{x}$.

Behavioural sublanguage. We now come to the behavioural sub-language, denoted TERM. Apart from the concepts already introduced for the data language above such as expressions, channels and variables), we also use a set of *process names*, denoted PROC and ranged over by P. Each process name has an associated arity, which is a natural number indicating its number of parameters (see below). It is generated by the following grammar:

$$\begin{aligned} t ::= {} & \mathbf{0} \mid \mathbf{1} \mid \tau \mid [E] \mid E!F \mid E?x;t \mid \{\mathbf{x} \leftarrow \mathbf{E}\};t \mid t + t \mid t;t \\ & \mid \mathbf{ch}\ \mathbf{c}.\ t \mid \mathit{spawn}(t) \mid P(\mathbf{E}) \end{aligned}$$

- $\mathbf{0}$ denotes the inactive process.
- $\mathbf{1}$ denotes a successfully terminated term.
- τ is an internal action which cannot interact with other actions.
- $[E]$ is a guard or *match* operator: the guard is passed (i.e., terminates) if E reduces to *true*, thus $[E];t$ evolves to t if E reduces to *true*.

t	$fv(t)$	$fc(t)$
$\mathbf{0}$	$\emptyset$	$\emptyset$
$\mathbf{1}$	$\emptyset$	$\emptyset$
$[E]$	$fv(E)$	$fc(E)$
$E!F$	$fv(E) \cup fv(F)$	$fc(E) \cup fc(F)$
$E?x; u$	$fv(E) \cup (fv(u) \setminus \{x\})$	$fc(E) \cup fc(u)$
$\{\mathbf{x} \leftarrow \mathbf{E}\}; u$	$fv(\mathbf{E}) \cup (fv(u) \setminus \{\mathbf{x}\})$	$fc(\mathbf{E}) \cup fc(u)$
$u_1 + u_2$	$fv(u_1) \cup fv(u_2)$	$fc(u_1) \cup fc(u_2)$
$u_1; u_2$	$fv(u_1) \cup fv(u_2)$	$fc(u_1) \cup fc(u_2)$
$\mathbf{ch}\ \mathbf{c}.\ u$	$fv(u)$	$fc(u) \setminus \{\mathbf{c}\}$
$spawn(u)$	$fv(u)$	$fc(u)$
$P(\mathbf{E})$	$fv(\mathbf{E})$	$fc(\mathbf{E})$

Fig. 1. Free variables and free channels.

- $E!F$, with $E, F \in$ Expr, denotes an output operation on a channel: the expression E defines the channel, the expression F defines the value to be sent.
- $E?x; t$ describes an input action on a channel: after a value is received on the channel defined by E, the value is bound to the data variable x; the scope of x extends to t.
- The declaration operator $\{\mathbf{x} \leftarrow \mathbf{E}\}; t$ declares the vector $\mathbf{x}$ (with $\{\mathbf{x}\} \subseteq$ Var) of data variables in t and assigns the vector $\mathbf{E}$ of expressions to it.
- The choice operator $t + u$ performs either t or u. $t; u$ denotes the sequential composition of t and u, i.e. u can perform actions when t has terminated.
- $\mathbf{ch}\ \mathbf{c}.\ t$ defines a new vector $\mathbf{c}$ of distinct channel values which can be used in t. The channels are local in t; therefore, the actions in t are not allowed to use this channel for communication with partners not in t. (Note that there are indeed *values* rather than variables.) As we will see, the ordering of the elements of $\mathbf{c}$ is immaterial; therefore we will sometimes write a *set* of channels C rather than a vector.
- $spawn(t)$ creates a new process which performs t concurrently to the spawning process: thus, $spawn(t); u$ represents the concurrent execution of t and u.
- Finally, process names P ($\in$ Proc) are interpreted by a function $\Theta :$ Proc $\to$ (Var* $\times$ Term), called the *process environment*. For every $P \in$ Proc, $\Theta(P) = (\mathbf{x}, t)$ (usually denoted $P(\mathbf{x}) \mapsto t$) represents a process declaration with name P, formal parameters $\mathbf{x}$ and body t; $|\mathbf{x}|$ equals the arity of P. $P(\mathbf{E})$ denotes a process call of $\Theta(P)$ with actual parameters $\mathbf{E}$ (where $|\mathbf{E}| = |\mathbf{x}|$). (However, see below, where we extend Θ with typing information.)

Note that we re-use the operator ";" to denote, on the one hand, two prefix-like operators ($E?x; t$ and $\{\mathbf{x} \leftarrow \mathbf{E}\}; t$), and on the other hand, full sequential composition ($t; u$ for arbitrary $t \in$ Term). The difference between the concepts of prefix and sequential composition is very small and lies only in their technical treatment; hence this overlap in notation should not cause confusion. (For a further discussion of this issue see Section 3.)

For syntactical convenience we assume that ; has a higher priority than +, which has a higher priority than $\mathbf{ch}\ \mathbf{c}.\ t$. For instance, $\mathbf{ch}\ c.\ a!c; b!v + b!v'$ equals $\mathbf{ch}\ c.\ ((a!c; b!v) + b!v')$. Furthermore, we assume sequential composition to be right associative, i.e. $t_1; t_2; t_3$ equals $t_1; (t_2; t_3)$. We let γ range over $\{\tau, E!F, [E] \mid E, F \in$ Expr$\}$.

In Figure 1 we define the free variables of a term, in the usual way, and also the free channels, in an analogous way. In particular, an input action $E?x; t$ binds x in t and a declaration $\{\mathbf{x} \leftarrow \mathbf{E}\}; t$ binds $\{\mathbf{x}\}$ in t; moreover, a restriction $\mathbf{ch}\ \mathbf{c}.\ t$ binds $\{\mathbf{c}\}$ in t. Finally, as we will see below, in a process definition $P(\mathbf{x}) \mapsto t$, it is required that all free variables of t are bound by the formal parameters (i.e., $fv(t) \subseteq \{\mathbf{x}\}$) and that t has no free channels (i.e., $fc(t) = \emptyset$). This implies that, in order to determine the free variables and channels of a process invocation $P(\mathbf{E})$, we do not have to consider the body of P.

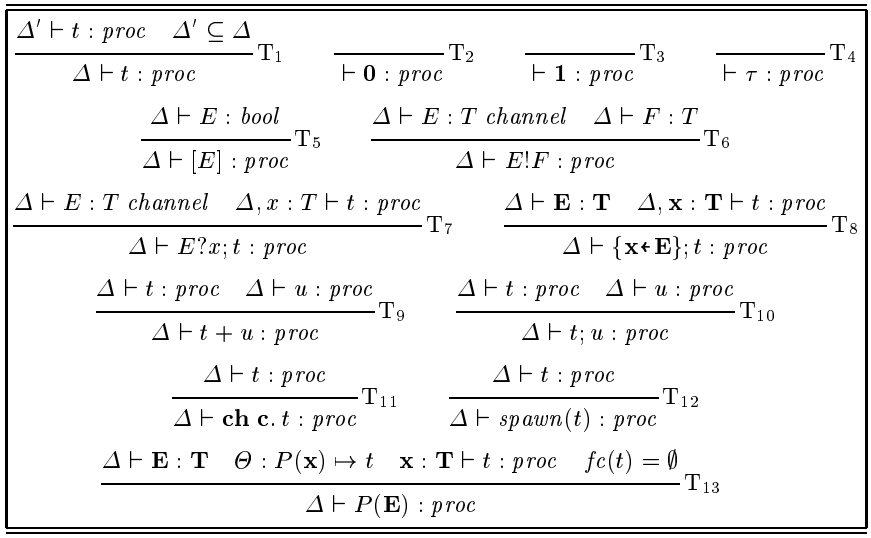


Fig. 2. Type system for TERM.

Typing. The type judgements for EXPR are extended to a type system for TERM. The only type for terms of TERM is *proc*; i.e., the type system only serves to characterise well-typed terms, without extracting any further information. The type system is given in Figure 2. Some comments are in order.

- T_5 constrains the expression E in a match $[E]$ to be of type *bool*, corresponding to the idea that this represents a guard which can either hold or fail to hold;
- T_6 expresses that the type of an output channel and the output value should fit together;
- T_7 and T_8 reflect the fact that input and declaration bind variables;
- T_{11} does not specify any constraints: the typing of the channel values in $\mathbf{c}$ is assumed to be pre-determined.
- T_{13} contains two constraints on the process environment Θ, already discussed above: each process definition must be typable by the formal parameters only (meaning that the body of the definition must be closed), where the types of the actual and formal parameters must coincide; and process definitions have no free channels.

As usual, well-typedness implies the absence of certain errors, for instance the application of a data operation on values for which that operation does not make sense (such as adding functions together, or sending an item over something that is not a channel). In the next section, we will see that well-typedness is preserved during the execution of a term.

In the presentation above, we have not provided explicit typings for the variables bound in $E?x;t$, $\{\mathbf{x} \leftarrow \mathbf{E}\};t$ or $P(\mathbf{x}) \mapsto t$, or the channels bound in $\mathbf{ch}\ \mathbf{c}.\ t$. In examples, however, we will often add such explicit types for readability and understandability; for instance, the definition of a process identifier P will often be given as $P(\mathbf{x} : \mathbf{T}) \mapsto t$.

3 Operational Semantics

Now we define the operational semantics of the process calculus. For this purpose, we use the notion of *labelled transition systems* [20].

Definition 1. A *labelled transition system* is a tuple $\langle \Omega, S, \rightarrow, q \rangle$ such that Ω is a set of labels, S is a set of *states*, $\rightarrow \subseteq S \times \Omega \times S$ is a *transition relation* and $q \in S$ is the initial state.

For our purpose, the transition labels (which reflect the observable behaviour of the system) can be one of the following:

- $c?v$ with $c \in \textsc{Chan}$ and $v \in \textsc{Value}$, representing the execution of an input action where the actual input value is v;
- $c!v, C$ with $c \in \textsc{Chan}$, $v \in \textsc{Value}$ and $C \subseteq fc(v)$, representing the execution of an output action, where C is a collection of fresh channels "announced" by the system (corresponding to the "bound output" of the π-calculus, see [21]), which is omitted when empty;
- τ, representing the occurrence of an internal action;
- $\checkmark$, representing the successful termination of a system (which, as we will see, does not imply that the system can display no more behaviour).

The set of all such labels is collected in Ω, ranged over by ω; moreover, we let α range over labels of the first three kinds, i.e., $\Omega \setminus \{\checkmark\}$.

To define the transitions of $\textsc{Term}$, we also need the concept of *substitution* of data variables by values. For this purpose, we use partial functions $\sigma \colon \textsc{Var} \rightarrow \textsc{Value}$. The application of σ to a term t involves the simultaneous substitution, within t, of all variables in the definition domain of σ by their σ-images; this is denoted $t\sigma$. Sometimes we explicitly give the substitutions as vectors $\langle \mathbf{v}/\mathbf{x} \rangle$ (where $\mathbf{v} \in \textsc{Expr}^*$ and $|\mathbf{v}| = |\mathbf{x}|$).

The operational semantics of terms in $\textsc{Term}$, then, is given by transitions of the form $t \xrightarrow{\omega} t'$. We impose a further well-formedness condition on such transitions: $t \xrightarrow{c!v,C} t'$ is well-formed only if $C \cap fc(t) = \emptyset$. This expresses that fresh channels announced during a transition may not occur free already before the transition. Figure 3 defines operational rules for $\textsc{Term}$ that give rise to a transition system semantics.

The main novelty in this semantics with respect to most other calculi with data (except [17]) is the treatment of termination and sequential composition (and, indirectly, of concurrent behaviour). First note that any term of the form $spawn(t)$ can always terminate (see R_{14}); on the other hand, it can also do any action that t can do (R_{15}). This implies that the term can be active even after it has terminated. The idea is that the behaviour that remains after termination runs *in parallel* to the rest of the system; in fact, as an independently spawned process.

The operational rules for sequential composition are adapted accordingly. First let us recall the standard rules (see, for example, [3]):

$$\frac{t \xrightarrow{\alpha} t'}{t; u \xrightarrow{\alpha} t'; u} \qquad \frac{t \xrightarrow{\checkmark} t' \quad u \xrightarrow{\alpha} u'}{t; u \xrightarrow{\alpha} u'}$$

In our setup, the first rule is fine, but the second one is not, since it discards the first operand. In the case where the first operand equals $spawn(t_0)$ for some t_0, this is not the desired behaviour; rather, $spawn(t_0)$ should remain in the target term, in order to execute the remainder of its behaviour (the independently spawned process). In general, if the first operand is terminated, the sequential composition behaves very much like standard *parallel* composition. This is indeed our intuition; in fact, we also allow *communication* between $spawn(t_0)$ and u in $spawn(t_0); u$ (see R_{11}).

Our approach to sequential composition is realised in R_9, R_{10} and R_{11} (in addition to the rules for *spawn*). Consider the following example:

$$\frac{}{\mathbf{1} \xrightarrow{\checkmark} \mathbf{1}}\mathrm{R}_1 \qquad \frac{}{\tau \xrightarrow{\tau} \mathbf{1}}\mathrm{R}_2 \qquad \frac{[\![E]\!] = true}{[E] \xrightarrow{\tau} \mathbf{1}}\mathrm{R}_3 \qquad \frac{}{E!F \xrightarrow{[\![E]\!]![\![F]\!]} \mathbf{1}}\mathrm{R}_4 \qquad \frac{}{E?x;t \xrightarrow{[\![E]\!]?v} t\langle v/x\rangle}\mathrm{R}_5$$

$$\frac{[\![\mathbf{E}]\!] = \mathbf{v} \quad t\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{\omega} t'}{\{\mathbf{x} \leftarrow \mathbf{E}\}; t \xrightarrow{\omega} t'}\mathrm{R}_6 \qquad \frac{t \xrightarrow{\omega} t'}{t + u \xrightarrow{\omega} t'}\mathrm{R}_7 \qquad \frac{u \xrightarrow{\omega} u'}{t + u \xrightarrow{\omega} u'}\mathrm{R}_8$$

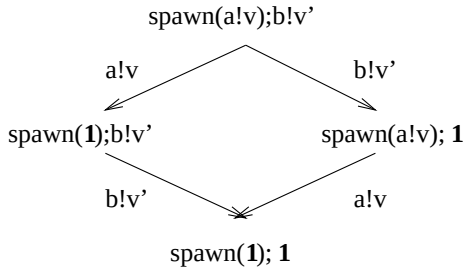
$$\frac{t \xrightarrow{\alpha} t'}{t; u \xrightarrow{\alpha} t'; u}\mathrm{R}_9 \qquad \frac{t \xrightarrow{\checkmark} t' \quad u \xrightarrow{\omega} u'}{t; u \xrightarrow{\omega} t'; u'}\mathrm{R}_{10}$$

$$\frac{t \xrightarrow{\checkmark} t' \quad t' \xrightarrow{\alpha} t'' \quad u \xrightarrow{\alpha'} u' \quad \{\alpha, \alpha'\} = \{c?v, (c!v, C)\} \quad C \cap fc(t; u) = \emptyset}{t; u \xrightarrow{\tau} \mathbf{ch}\ C.\, t''; u'}\mathrm{R}_{11}$$

$$\frac{t \xrightarrow{\omega} t' \quad fc(\omega) \cap \{\mathbf{c}\} = \emptyset}{\mathbf{ch}\ \mathbf{c}.\, t \xrightarrow{\omega} \mathbf{ch}\ \mathbf{c}.\, t'}\mathrm{R}_{12} \qquad \frac{t \xrightarrow{c!v, A} t' \quad c \notin \{\mathbf{c}\} \quad A \cap C = \emptyset \quad fc(v) \cap \{\mathbf{c}\} \neq \emptyset}{\mathbf{ch}\ \mathbf{c}.\, t \xrightarrow{c!v, A \cup (fc(v) \cap \{\mathbf{c}\})} \mathbf{ch}\ \mathbf{c} \setminus fc(v).\, t'}\mathrm{R}_{13}$$

$$\frac{}{spawn(t) \xrightarrow{\checkmark} spawn(t)}\mathrm{R}_{14} \qquad \frac{t \xrightarrow{\alpha} t'}{spawn(t) \xrightarrow{\alpha} spawn(t')}\mathrm{R}_{15} \qquad \frac{[\![\mathbf{E}]\!] = \mathbf{v} \quad \Theta : P(\mathbf{x} : \mathbf{T}) \mapsto t}{P(\mathbf{E}) \xrightarrow{\tau} t\langle \mathbf{v}/\mathbf{x}\rangle}\mathrm{R}_{16}$$

Fig. 3. Operational semantics (note that, in addition, $t \xrightarrow{c!v, C} t'$ only if $C \cap fc(t) = \emptyset$).



Some more comments:

- Rule R_6 describes the binding operator $\{\mathbf{x} \leftarrow \mathbf{E}\}; t$. If the vector $\mathbf{E}$ of expressions can be reduced to a vector $\mathbf{v}$ of values and $t\langle \mathbf{v}/\mathbf{x}\rangle$ can perform a transition ω to become t', then $\{\mathbf{x} \leftarrow \mathbf{E}\}; t$ can perform the same transition. Note that (just as with $E?x; t$), although notationally this operator is very similar to sequential composition, the first "operand" can be executed in a single transition, and therefore termination is not needed.
- Rule R_{11} expresses communication. If a process term t is terminated, and may also perform an action α, it is clear that α originates from a *spawn*-subterm. If u is able to perform the dual action α' such that $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, implying that α and α' specify input and output over the same channel, then communication is possible. Furthermore, the private channels C announced in the output action are restricted to the two communication partners.
- The rules R_{12} and R_{13} deal with the restriction of channels. If in a term $\mathbf{ch}\ C.\, t$ the subterm t performs an action ω and the set $fc(\omega)$ of the free channels in ω is disjoint with the set C of restricted channels, $\mathbf{ch}\ C.\, t$ can perform the same ω-transition. In case of an output action $(c!v, A)$ that annouces private channels $A \subseteq fc(v)$, $\mathbf{ch}\ C.\, t$ must also announce the values of $fc(v) \cap C$ as additional local channels. Furthermore, in the resulting term, the announced channels in $fc(v)$ must be removed from the set C. Finally, we have to ensure that $A \cap C = \emptyset$. Without this condition, we could for example derive a transition $\mathbf{ch}\ c.\, \mathbf{ch}\ c.\, a!c \xrightarrow{a!c, \{c\}} \mathbf{1}$, which contradicts the intuition that each channel restriction introduces a new channel value.
- Rule R_{16} defines the behaviour of a process call $P(\mathbf{E})$. If the vector $\mathbf{E}$ of actual parameters can be reduced to a vector of values $\mathbf{v}$, then the process call unfolds in a τ-step to

the process body t in which the formal parameters $\mathbf{x}$ are substituted by the values of $\mathbf{v}$.

In order to be able to meet the condition $A \cap C$ in R_{13}, as well as the well-formedness condition on transitions ($t \xrightarrow{c!v,C} t'$ only if $C \cap fc(t) = \emptyset$), we allow the α-conversion of bound channel values. We denote $t =_\alpha u$ if u can be obtained from t by such α-conversion; thus, $\mathbf{ch}\ c.t =_\alpha \mathbf{ch}\ d.t\langle c/d\rangle$ for arbitrary $c, d \in \textsc{Chan}$ and $t \in \textsc{Term}$. The transition relation is then interpreted up to α-conversion of its source term; that is, if $t =_\alpha t'$ and $t' \xrightarrow{\omega} t''$ then $t \xrightarrow{\omega} t''$ is also defined to hold. For instance, in order to derive an outgoing transition of the term $\mathbf{ch}\ c.\ \mathbf{ch}\ c.\ a!c$ above, one of the restricted channels should be α-converted to a fresh name: $\mathbf{ch}\ c.\ \mathbf{ch}\ c.\ a!c =_\alpha \mathbf{ch}\ d.\ \mathbf{ch}\ c.\ a!c \xrightarrow{a!c,\{c\}} \mathbf{ch}\ d.\ \mathbf{1}$.

The following example shows why transition well-formedness is important. Here, the channel name c is used twice for different local channels; the well-formedness condition ensures that the second channel is renamed by alpha-conversion before it can be extruded. Otherwise, there would be a name conflict with the first announcement of channel c.

$$\begin{aligned}
&\mathit{spawn}((\mathbf{ch}\ c.\ a!c); (\mathbf{ch}\ c.\ b!c)); a?x; b?y; y!x \\
&\quad \xrightarrow{\tau} \mathbf{ch}\ c.\ \mathit{spawn}(\mathbf{1}; (\mathbf{ch}\ c.\ b!c)); b?y; y!c \\
&\quad =_\alpha \mathbf{ch}\ c.\ \mathit{spawn}(\mathbf{1}; (\mathbf{ch}\ d.\ b!d)); b?y; y!c \\
&\quad \xrightarrow{\tau} \mathbf{ch}\ c, d.\ \mathit{spawn}(\mathbf{1}; \mathbf{1}); d!c
\end{aligned}$$

In all, the operational rules give rise to a transition system that constitutes the semantics of $\textsc{Term}$.

Definition 2. The *semantics* of a term $t \in \textsc{Term}$ is the transition system $\langle \Omega, \textsc{Term}, \rightarrow, t\rangle$, where the transition relation $\rightarrow$ is defined by the rules in Figure 3, augmented with

$$\frac{t =_\alpha t' \quad t' \xrightarrow{\omega} t''}{t \xrightarrow{\omega} t''}$$

The following theorem is analogous to the "subject reduction theorem" by Curry (see, for example, [5]). It states that, if a well-typed term performs a transition step, the resulting term is also well-typed. Therefore, it is ensured that all terms occuring in the evaluation of a well-typed term are also well-typed, and thus error-free in the sense mentioned in the previous section.

Theorem 3. *Assume* $\vdash t : \mathit{proc}$.

- *If* $t \xrightarrow{c?v} t'$ *then* $\vdash c : T$ *channel for some* T*; if moreover* $\vdash v : T$ *then* $\vdash t' : \mathit{proc}$.
- *If* $t \xrightarrow{c!v,C} t'$ *then* $\vdash c : T$ *channel and* $\vdash v : T$ *for some* T *and* $\vdash t' : \mathit{proc}$.
- *If* $t \xrightarrow{\omega} t'$ *and* $\omega = \tau$ *or* $\omega = \checkmark$*, then* $\vdash t' : \mathit{proc}$.

4 Bisimulation

In this section, we define two equivalence relations over TERM: *strong* and *weak* bisimilarity (see [20]). Both are based on the idea that two systems can be regarded as equivalent if they can perform equivalent transitions; the weak version differs from the strong in that "performing an equivalent transition" may involve intermediate internal steps.

Strong bisimulation. In the standard definition of [20], two terms t, u are bisimilar if every $t \xrightarrow{\omega} t'$ is matched by a $u \xrightarrow{\omega} u'$ such that the resulting terms t', u' are also bisimilar, and vice versa. However, due to our (parametric) incorporation of a data language, the demand for identical labels for t and u is too restrictive. For example, if we use a data language with functions based on the λ-calculus, the transition labels $a!(\lambda x.\lambda y.x + y)$ and $a!(\lambda x.\lambda y.y + x)$ would be distinguished, although the functions compute the same result in any application. Instead of demanding identity, we require that the labels are β-equivalent. First, we extend the notion of β-equivalence to transition labels.

Definition 4. Beta-equivalence on transition labels is defined as follows:

- $\tau =_\beta \tau$
- $\checkmark =_\beta \checkmark$
- $c?v =_\beta c?w$ if $v =_\beta w$.
- $(c!v, C) =_\beta (c!w, C)$ if $v =_\beta w$.

Furthermore, $\langle \mathbf{v}/\mathbf{x} \rangle =_\beta \langle \mathbf{v}'/\mathbf{x}' \rangle$ if $\mathbf{x} = \mathbf{x}'$ and $\mathbf{v} =_\beta \mathbf{v}'$.

Now we can define the bisimulation of terms.

Definition 5. Let $R \subseteq \text{TERM} \times \text{TERM}$ be a symmetrical relation. R is called a bisimulation relation if $(t, u) \in R$ implies that for all $t \xrightarrow{\omega} t'$:

- If $\omega = c?v$, then $\forall \omega', \omega =_\beta \omega', \exists u' : u \xrightarrow{\omega'} u'$ and $(t', u') \in R$.
- If $\omega \neq c?v$, then $\exists \omega', \omega =_\beta \omega', \exists u' : u \xrightarrow{\omega'} u'$ and $(t', u') \in R$.

Two closed terms t, u are called *bisimilar*, written $t \sim_\beta u$, if there exists some bisimulation relation R such that $(t, u) \in R$. If t and u are open terms, we write $t \sim_\beta u$ if $\forall \sigma : t\sigma \sim_\beta u\sigma$.

The relation $\sim_\beta$ corresponds to *early bisimulation* in the π-calculus [23]. For input transitions $t \xrightarrow{c?v} t'$, we require that u can perform a corresponding transition for any v' equivalent to v. For output transitions $t \xrightarrow{\omega} t'$, we only demand that at least one matching transition $u \xrightarrow{\omega'} u'$ exists.

By distinguishing variables and values from the start, we have avoided the problem of the π-calculus that bisimulation fails to be a congruence for input actions. The above definition of bisimilulation only applies to closed terms; it is extended to open terms in the standard way, namely by requiring that all their closed-term instantiations are equivalent. In the π-calculus, on the other hand, the notion of free variable does not exist, because every channel identifier can also be used to stand for a channel name. Therefore, every term can be executed without applying substitutions. Thus, substitution changes the behaviour of the term, e.g. by enabling new communication possibilities which were not present before the substitution has been applied.

The following theorem states that bisimulation is a congruence for all operators of TERM.

Theorem 6. *$\sim_\beta$ is a congruence for the operators of* TERM.

The proof of this theorem relies on the following lemma, which states that when β-equivalent substitutions are applied to the same term t, the resulting terms are bisimilar.

Lemma 7. *If $\sigma =_\beta \sigma'$, then $t\sigma \sim_\beta t\sigma'$.*

Axiomatisation. In Figure 4 we give a set $\mathcal{AX}_{\sim_\beta}$ of axioms for the axiomatisation of $\sim_\beta$. The axioms can be separated into the following groups:

- Axioms (1)–(12) concern the fragment of TERM without channel restriction or *spawn*. Except for the match and declaration operators and the treatment of input as a prefix operator, this fragment is identical to BPA, the fragment of ACP without parallel composition (see [3]). Hence, these axioms can be regarded as standard.
- Axioms (13)–(24) express how the restriction operator interacts with the other operators of TERM: for instance, if a restricted channel does not occur in a sub-term, the sub-term may be removed from under the restriction operator.
- Axiom (25) expresses that a process invocation corresponds to an internal step (reflecting the unfolding of the process definition) followed by the execution of the process' body (with the appropriate parameter substitution). Note that we require that the parameters have been reduced to values; otherwise it would be possible that the reduction of the parameter expressions does not terminate, which means that the left hand side of the equation would not be able to perform the process creation while the right hand side can always perform a τ-step.
- Axioms (26)–(29) express the effect of spawning a process; they are analogous to the equations developed in [4] for basically the same operator.
- Rules (30)–(35) concern the interplay of the data formalism, in particular as regards β-equivalence, with the process formalism: they basically express that if two data expressions are β-equivalent, they can be interchanged within a process expression.
- Axiom (36) is the expansion law for TERM. In order to be able to represent this concisely, we have used some notational conventions, discussed below.

As examples for derived equations, we have $spawn(\mathbf{1}) = \mathbf{1}$, $spawn(spawn(t)) = spawn(t)$ and $spawn(t_1; spawn(t_2)) = spawn(t_1; t_2)$.

For the purpose of representing the expansion law concisely, we define the notion of *derived output actions*, following the definition of *derived prefixes* in [21]. If $c \notin D$, then we can replace a term $\mathbf{ch}\ D.\ c!v; t$ by $(c!v, C); t'$ with $C = D \cap fc(v)$ and $t' = \mathbf{ch}\ D \setminus fc(v).\ t$. The sub-term $(c!v, C)$ is called a *derived output action*. By using derived output actions as an auxiliary notation, we can remove restriction operators from terms.

The expansion law has a side condition to avoid the capture of free variables. This is in order to disallow the derivation of equations like $spawn(a?x; \mathbf{1}); b!x = (a?x; spawn(\mathbf{1}); b!x) + (b!x; spawn(a?x; \mathbf{1}))$, where in the left hand term the occurence of x in $b!x$ is free, whereas in the right hand term it is bound by $a?x$.

We then have the following result:

Theorem 8. *The axiomatic theory $\mathcal{AX}_{\sim_\beta}$ is sound with respect to $\sim_\beta$.*

Weak bisimulation. As an equivalence relation on process terms, strong bisimilarity is often too restrictive, because it does not abstract from the internal behaviour of processes. Therefore, in this section we define the notion of *weak bisimulation* [20], which does abstract from internal behaviour: We require that each τ-transition can be matched by zero or more τ-moves.

Let $\Rightarrow$ be the reflexive and transitive closure of $\xrightarrow{\tau}$, and let $\stackrel{\omega}{\Longrightarrow}$ denote $\Rightarrow\xrightarrow{\omega}\Rightarrow$ for arbitrary $\tau \in \Omega$. Let $\stackrel{\widehat{\omega}}{\Longrightarrow}$ equal $\Rightarrow$ if $\omega = \tau$, and $\stackrel{\omega}{\Longrightarrow}$ otherwise.

Definition 9. Let $R \subseteq \text{TERM} \times \text{TERM}$ be a symmetrical relation. R is called a *weak bisimulation relation* if $(t, u) \in R$ implies that for all $t \xrightarrow{\omega} t'$:

- If $\omega = c?v$, then $\forall \omega', \omega =_\beta \omega', \exists u' : u \stackrel{\widehat{\omega'}}{\Longrightarrow} u'$ and $(t', u') \in R$.
- If $\omega \neq c?v$, then $\exists \omega', \omega =_\beta \omega', \exists u' : u \stackrel{\widehat{\omega'}}{\Longrightarrow} u'$ and $(t', u') \in R$.

$$[false] = \mathbf{0} \quad (1)$$
$$[true] = \tau \quad (2)$$
$$\mathbf{1}; t = t \quad (3)$$
$$t; \mathbf{1} = t \quad (4)$$
$$\mathbf{0}; t = \mathbf{0} \quad (5)$$
$$t_1; (t_2; t_3) = (t_1; t_2); t_3 \quad (6)$$
$$\{\mathbf{x} \leftarrow \mathbf{v}\}; t = t\langle \mathbf{v}/\mathbf{x} \rangle \quad (7)$$
$$t + \mathbf{0} = t \quad (8)$$
$$t + u = u + t \quad (9)$$
$$t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3 \quad (10)$$
$$t + t = t \quad (11)$$
$$(t_1 + t_2); t_3 = t_1; t_3 + t_2; t_3 \quad (12)$$

$$\mathbf{ch}\ C.\ \mathbf{0} = \mathbf{0} \quad (13)$$
$$\mathbf{ch}\ C.\ \mathbf{1} = \mathbf{1} \quad (14)$$
$$\mathbf{ch}\ C.\ spawn(t) = spawn(\mathbf{ch}\ C.\ t) \quad (15)$$
$$\mathbf{ch}\ C.\ t = t \quad \text{if } C \cap fc(t) = \emptyset \quad (16)$$
$$\mathbf{ch}\ C.\ \mathbf{ch}\ C'.\ t = \mathbf{ch}\ C \cup C'.\ t \quad (17)$$
$$\mathbf{ch}\ C.\ t + u = \mathbf{ch}\ C.\ t + \mathbf{ch}\ C.\ u \quad (18)$$
$$\mathbf{ch}\ C.\ c?x; t = \mathbf{0} \quad \text{if } c \in C \quad (19)$$
$$\mathbf{ch}\ C.\ c!v; t = \mathbf{0} \quad \text{if } c \in C \quad (20)$$
$$\mathbf{ch}\ C.\ \gamma; t = \gamma; \mathbf{ch}\ C.\ t \quad \text{if } C \cap fc(\gamma) = \emptyset \quad (21)$$
$$\mathbf{ch}\ C.\ c?x; t = c?x; \mathbf{ch}\ C.\ t \quad \text{if } c \notin C \quad (22)$$
$$(\mathbf{ch}\ C.\ t); u = \mathbf{ch}\ C.\ t; u \quad \text{if } C \cap fc(u) = \emptyset \quad (23)$$
$$t; \mathbf{ch}\ C.\ u = \mathbf{ch}\ C.\ t; u \quad \text{if } C \cap fc(t) = \emptyset \quad (24)$$
$$P(\mathbf{v}) = \tau; t\langle \mathbf{v}/\mathbf{x} \rangle \quad \text{if } \Theta : P(\mathbf{x}) \mapsto t \quad (25)$$

$$spawn(\mathbf{0}) = \mathbf{1} \quad (26)$$
$$spawn(t); spawn(u) = spawn(u); spawn(t) \quad (27)$$
$$spawn(t); spawn(u) = spawn(spawn(t); u) \quad (28)$$
$$spawn(t_1; spawn(t_2) + t_3) = spawn(t_1; t_2 + t_3) \quad (29)$$

$$E!F = E'!F' \quad \text{if } E =_\beta E', F =_\beta F' \quad (30)$$
$$E?x; t = E'?x; t \quad \text{if } E =_\beta E' \quad (31)$$
$$\{\mathbf{x} \leftarrow \mathbf{E}\}; t = \{\mathbf{x} \leftarrow \mathbf{E}'\}; t \quad \text{if } \mathbf{E} =_\beta \mathbf{E}' \quad (32)$$
$$[E] = [E'] \quad \text{if } E =_\beta E' \quad (33)$$
$$P(\mathbf{E}) = P(\mathbf{E}') \quad \text{if } \mathbf{E} =_\beta \mathbf{E}' \quad (34)$$
$$t\langle v/x \rangle = t \quad \text{if } x \notin fv(t) \quad (35)$$

if $t = \sum_{i \in \mathcal{I}} \delta_i; t_i + \sum_{j \in \mathcal{J}} E_j?x_j; t_j$ (δ ranges over $(c!v, C), [E], \tau$)
and $u = \sum_{k \in \mathcal{K}} \delta_k; t_k + \sum_{l \in \mathcal{L}} E_l?x_l; t_l$
and $\forall j \in \mathcal{J} : x_j \notin fv(u)$ and $\forall l \in \mathcal{L} : x_l \notin fv(t)$

$$\text{then } spawn(t); u = \sum_{i \in \mathcal{I}} \delta_i; spawn(t_i); u + \sum_{k \in \mathcal{K}} \delta_k; spawn(t); u_k \quad (36)$$
$$+ \sum_{j \in \mathcal{J}} E_j?x_j; spawn(t_j); u + \sum_{l \in \mathcal{L}} E_l?x_l; spawn(t); u_l$$
$$+ \sum_{\substack{i \in \mathcal{I}, l \in \mathcal{L} \\ \delta_i = (c!v, C), [\![E_l]\!] = c}} \tau; \mathbf{ch}\ C.\ spawn(t_i); u_l\langle v/x_l \rangle$$
$$+ \sum_{\substack{j \in \mathcal{J}, k \in \mathcal{K} \\ [\![E_j]\!] = c, \delta_k = (c!v, C)}} \tau; \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j \rangle); u_k$$

Fig. 4. Axioms for (strong) bisimulation.

Two closed terms t, u are called *weakly bisimilar*, written $t \approx_\beta u$, if there exists some weak bisimulation relation R such that $(t, u) \in R$. If t and u are open terms, we write $t \approx_\beta u$ if $\forall \sigma : t\sigma \approx_\beta u\sigma$.

The following theorem states that weak bisimilarity is a congruence for all operators of TERM except choice. The lack of congruence for choice is a problem well-known from CCS, with a well-known solution; see below.

Theorem 10. *$\approx_\beta$ is a congruence for spawn, variable declaration, input actions, channel restriction and sequential composition.*

Apart from the axioms in Figure 4, which are sound up to strong bisimilarity and hence certainly up to weak, there are some equations having specifically to do with internal moves that are satisfied by weak bisimilarity but not by strong.

Proposition 11. *For all $t \in$ TERM : $\tau; t \approx_\beta t$ and $spawn(\tau; t) \approx_\beta spawn(t)$. Furthermore, $\mathbf{1}; \tau \approx_\beta \mathbf{1}$.*

Weak bisimulation is not a congruence for choice (for example, $a!v \approx_\beta \tau; a!v$ but $c!v' + a!v \not\approx_\beta c!v' + \tau; a!v$); therefore, we define the notion of *weak congruence* (or *observation congruence*) [20].

Definition 12. Let $R \subseteq \text{TERM} \times \text{TERM}$ be a symmetrical relation. R is called a *rooted bisimulation relation* if $(t, u) \in R$ implies that for all $t \xrightarrow{\omega} t'$:

- If $\omega = c?v$, then $\forall \omega', \omega =_\beta \omega', \exists u' : u \stackrel{\omega'}{\Longrightarrow} u'$ and $t' \approx_\beta u'$.
- If $\omega \neq c?v$, then $\exists \omega', \omega =_\beta \omega', \exists u' : u \stackrel{\omega'}{\Longrightarrow} u'$ and $t' \approx_\beta u'$.

Two closed terms t, u are called *weakly congruent*, written $t \simeq_\beta u$, if there exists some rooted bisimulation relation R such that $(t, u) \in R$.

The following proposition describes the relationship between strong bisimulation, weak bisimulation and weak congruence.

Proposition 13. *$t \sim_\beta u$ implies $t \simeq_\beta u$, and $t \simeq_\beta u$ implies $t \approx_\beta u$.*

As usual, rooted bisimulation solves the congruence problem of weak bisimilarity.

Theorem 14. *$\simeq_\beta$ is the largest congruence for the operations of* TERM *contained in $\approx_\beta$.*

In Figure 5 we extend the axiomatic theory $\mathcal{AX}_{\sim_\beta}$ with axioms for $\simeq_\beta$. We denote the extended theory by $\mathcal{AX}_{\simeq_\beta}$. Note that the equations in Proposition 11 are *not* valid up to weak congruence: they are typical of the difference between $\simeq_\beta$ and $\approx_\beta$.

$\gamma; \tau = \gamma$ (37)	$t + \tau; t = \tau; t$ (39)
$spawn(\tau; t) = \tau; spawn(t)$ (38)	$E?x; \tau; t = E?x; t$ (40)

Fig. 5. Axioms for weak congruence.

Theorem 15. *The axiomatic theory $\mathcal{AX}_{\simeq_\beta}$ is sound with respect to $\simeq_\beta$.*

not	$: bool \to bool$
and, or	$: (bool \times bool) \to bool$
$+, -, *, /, mod$	$: (int \times int) \to int$
$<, >$	$: (int \times int) \to bool$
$=$	$: (ET \times ET) \to bool$

Fig. 6. Operator types.

5 Integration of a Functional Data Language

As an example for the integration of a data language into the process calculus, we introduce in this section an enriched typed lambda calculus for the description of data transformations. We define a type system for the calculus and give a set of predefined operations on the datatypes. Furthermore, we apply the new language to a case study in which a remote memory component is specified. By using the axiomatic theory, we prove that two versions of the case study show equivalent behaviour.

First, we define the set DTYPE of data types. It is generated by the following grammar:

$$\begin{aligned} ET &::= int \mid bool \mid T\ channel \mid ET \times \cdots \times ET \mid ET\ list \\ T &::= ET \mid T \times \cdots \times T \mid T\ list \mid T \to T \end{aligned}$$

ET defines the subset of type expressions for which the notion of equality is defined, i.e. type expressions which do not contain function types as subterms. T represents arbitrary type expressions. *int* and *bool* are basic types, with values given by $\text{VALUE}^{bool} = \{false, true\}$ and $\text{VALUE}^{int} = \{\ldots, -1, 0, 1, \ldots\}$. Values of type $T\ channel$ are channels for transmitting values of type T. Similarly, the values of $T\ list$ are lists that only have elements of type T; they are constructed by *nil* or *cons* $E\ L$ (where L is again of type $T\ list$). We abbreviate terms of the form $cons\ E_1(cons\ E_2 \cdots (cons\ E_n\ nil))$ by $[E_1, \ldots, E_n]$. $T \to T'$ denotes a function type, whose elements are written $\lambda x.E$ or (more usually) $\lambda x : T.E$. The type $T_1 \times \ldots \times T_n$ denotes the set of tuples with n (appropriately typed) components; elements are denoted $(E_1, \ldots, E_n)$.

The data language, i.e., the set EXPR of expressions, is defined by the following grammar:

$$\begin{aligned} E &::= V \mid op_{un}\ E \mid E\ op_{bin}\ E \mid E\ E \mid if\ E\ then\ E\ else\ E \\ &\quad \mid let\ x = E\ in\ E \mid cons\ E\ E \\ V &::= const \mid cons\ V\ V \mid (V, \ldots, V) \mid \lambda x.E \\ const &::= true \mid false \mid nil \mid n \mid channel\text{-}id \\ op_{un} &::= not \mid - \mid head \mid tail \mid Y \\ op_{bin} &::= + \mid - \mid * \mid / \mid mod \mid and \mid or \mid < \mid > \mid = \end{aligned}$$

Most of these constructs are standard. V defines the values of our language, i.e. the elements of VALUE. *const* defines the constants; these were discussed above; n stands for an arbitrary integer. op_{un} and op_{bin} represent the predefined unary and binary operators and functions, respectively. In Figure 6 the types of the operators are defined. We assume a set of functions $\#i$ to access the i'th component of a tuple. Furthermore, to define recursion, we use the Y-combinator [5]. The functions *head* and *tail* compute the head and the remainder of a list, respectively.

In Figure 7 the typing rules for the data language are given.

Definition 16. Let $\text{CEXPR} \subset \text{EXPR}$ be the set of closed expressions. The reduction relation $\hookrightarrow \subseteq \text{CEXPR} \times \text{CEXPR}$ is defined by the rules in Figure 8. Let $\hookrightarrow^*$ denote the reflexive and transitive closure of $\hookrightarrow$. A value v is the *reduction semantics* of an expression E, denoted $[\![E]\!] = v$, if $E \hookrightarrow^* v$.

We show that the subject reduction theorem is valid for the rules of the reduction semantics.

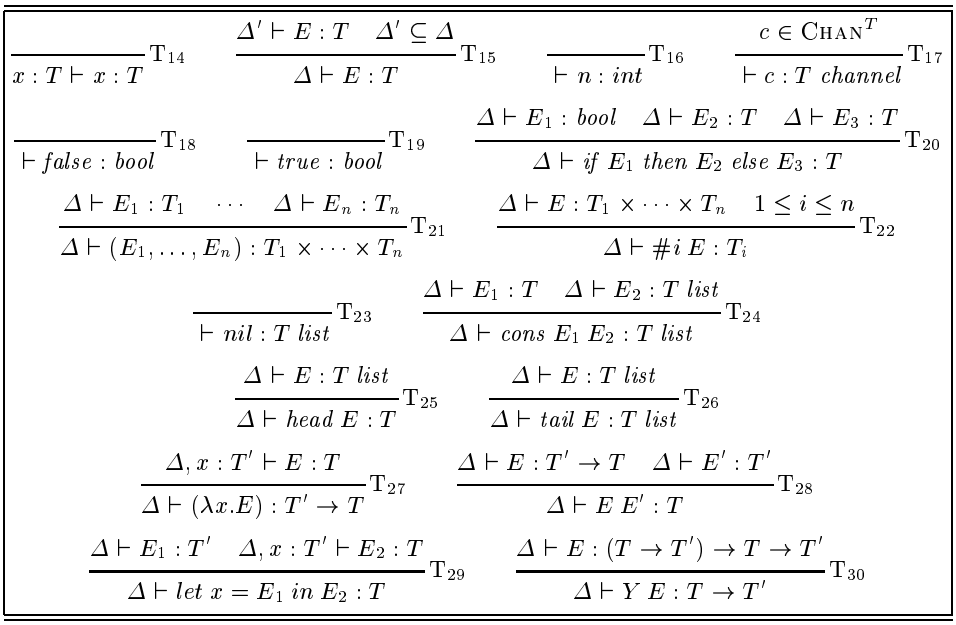


Fig. 7. Type rules for functional data language.

Theorem 17. *If* $\vdash E : T$ *and* $E \hookrightarrow E'$, *then* $\vdash E' : T$.

Definition 18. We define the partial function $fc : \textsc{Expr} \rightharpoonup 2^{\textsc{Chan}}$ which computes the set of free channels of an expression:

- $fc(const) = \emptyset$
- $fc(cons\ v_1\ v_2) = fc(v_1) \cup fc(v_2)$
- $fc(v_1, \ldots, v_n) = \bigcup_{1 \le i \le n} fc(v_i)$
- $fc(\lambda x.E) = \bigcup_v fc([\![\bar{E}\ \bar{v}]\!])$

The function fc is only defined for values, because it is not possible that unreduced expressions may occur in transition labels (see R_{12} and R_{13}).

Now we define the notion of beta-equivalence of expressions.

Definition 19. Let $=_\beta \subseteq \textsc{CExpr} \times \textsc{CExpr}$ be a relation. If E, E' are not lambda-abstractions, then $E =_\beta E'$ if $\exists v : [\![E]\!] = v = [\![E']\!]$. If E, E' are lambda-abstractions, then $E =_\beta E'$ if $\forall v' \exists v$: $[\![E\ v']\!] = v = [\![E'\ v']\!]$. Two open expressions E, E' are beta-equivalent if $\forall \sigma : E\sigma =_\beta E'\sigma$.

Note that the definitions of $=_\beta$ and fc fulfil the requirement $E =_\beta F \Rightarrow fc(E) = fc(F)$.

Example: FIFO queue. As an example, we specify a queue of numbers. The queue interacts with its environment via two channels: On channel *put* the environment can send values to the cell, on *get* the first element of the current queue can be read. If the cell is empty, communication on channel *get* should not be possible. In our language, the queue can be specified as follows:

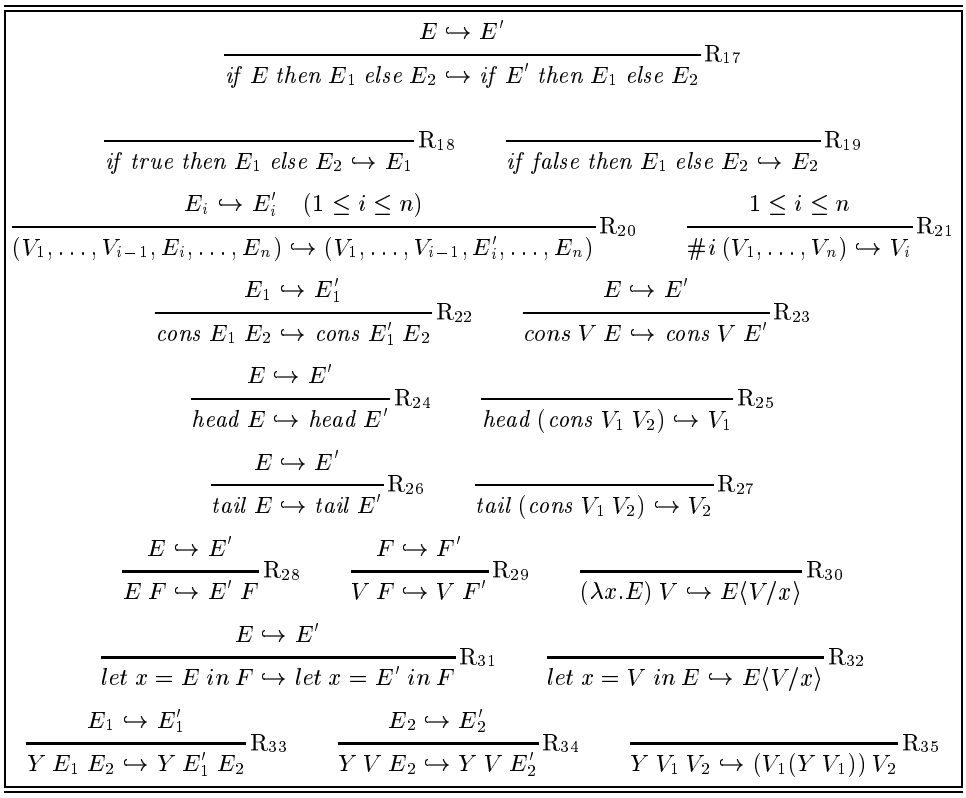


Fig. 8. Reduction semantics for functional data language.

$$
\begin{array}{l}
\mathit{FIFO\_Queue}\ (\mathit{queue} : \mathit{int\ list},\ \mathit{get} : \mathit{int\ channel},\ \mathit{put} : \mathit{int\ channel}) \mapsto \\
\quad \{\mathit{append} \leftarrow \\
\qquad \mathit{let\ append}_0 = \lambda f : (\mathit{int\ list} \to \mathit{int\ list} \to \mathit{int\ list}).\ \lambda l_1 : \mathit{int\ list}.\lambda l_2 : \mathit{int\ list}. \\
\qquad\qquad \mathit{if}\ l_1 = \mathit{nil}\ \mathit{then}\ l_2\ \mathit{else}\ \mathit{cons}\ (\mathit{head}\ l_1)\ (f\ (\mathit{tail}\ l_1)\ l_2) \\
\qquad \mathit{in}\ Y\ \mathit{append}_0\}; \\
\quad (\ \ [\mathit{queue} = \mathit{nil}]; \\
\qquad\quad \mathit{put}?\mathit{value};\ \mathit{FIFO\_Queue}([\mathit{value}], \mathit{get}, \mathit{put}) \\
\quad + [\mathit{not}\ (\mathit{queue} = \mathit{nil})]; \\
\qquad (\qquad \mathit{put}?\mathit{value};\ \mathit{FIFO\_Queue}(\mathit{append}\ \mathit{queue}\ [\mathit{value}], \mathit{get}, \mathit{put}) \\
\qquad\quad + \mathit{get}!\mathit{head}\ \mathit{queue};\ \mathit{FIFO\_Queue}(\mathit{tail}\ \mathit{queue}, \mathit{get}, \mathit{put})) \\
\quad)
\end{array}
$$

The process definition *FIFO_Queue* has three parameters. The list *queue* contains the current state. Interaction with the environment is possible via the channels *get* and *put*, as described above. First, a function for the concatenation of two lists is defined and bound to the identifier *append*. The function makes use of the Y-combinator to express recursion. For that purpose, a local function append_0 is defined, which uses an additional parameter f. After the definition of *append*, the current state of the queue is examined. If the queue is empty, only the sending of a new value to the cell is possible. After reading a value on *put*, the process calls itself recursively with a new queue containing the read value. If the queue is not empty, communication on both *get* and *put* is possible. If a value is sent to the cell, it is appended to the current queue. Otherwise, the first element of the queue removed from the queue and is sent on *get*. In both alternatives, the process calls itself recursively with the new state of the queue.

As an example for interaction with the queue, consider the following process term:

$$\textbf{ch}\ get, put.\ spawn(FIFO\_Queue(nil, get, put)); put!1; put!2; get?x; get?y; \mathbf{1}$$

This specifies that after an instance of the cell has been created, some values are sent to the cell and read from there. The restriction of the channels *get* and *put* enforces communication on these channels, and prevents any outside process to send or receive anything on them.

5.1 The RPC-Memory-Cell Case Study

As a second example, we recall a case study that was used as the basis for a workshop on system specification [6]. In this case study, a system composed of two components shall be specified. The first component is a memory for integer values. It consists of a set of memory cells which can be adressed individually for read and write access. The second component is a *remote procedure call* component, which accepts the requests for memory access from the users, delegates them to the memory component and also transmits the result of the memory access to the corresponding user.

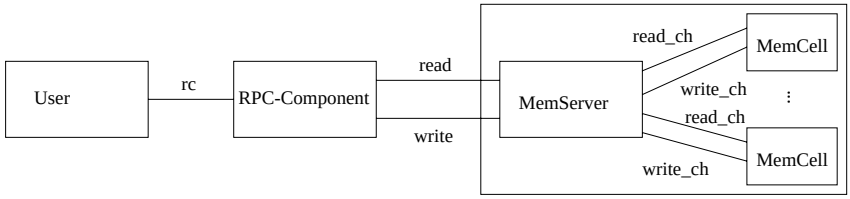


Fig. 9. Structure of the RPC-Memory-Cell Case Study.

In Figure 9 the structure of the system is given. The user sends the requests for memory access on a channel *rc* (for *remote call*) to the RPC-component. The component decodes the request and delegates it to the memory server within the memory by sending read requests on the *read* channel and write requests on the *write* channel. The memory server interacts with the adressed memory cell and delivers the received result via the RPC-component to the user.

The memory component. The memory can be specified as follows:

$$\begin{aligned}
&MemCell\ (value : int, readchan : int\ channel, writechan : int\ channel) \mapsto \\
&\qquad readchan!value; MemCell(value, readchan, writechan) \\
&\quad + \ writechan?newval; MemCell(newval, readchan, writechan)
\end{aligned}$$

$$\begin{aligned}
&CreateCells\ (count : int, readchans : int\ channel\ list, \\
&\qquad writechans : int\ channel\ list, exit : (int\ channel\ list \times int\ channel\ list)\ channel) \\
&\quad [count > 0]; \\
&\quad \textbf{ch}\ rc, wc.\ (spawn(MemCell(0, rc, wc)); \\
&\qquad CreateCells(count - 1, cons\ rc\ readchans, cons\ wc\ writechans, exit)) \\
&\quad + \ [count = 0]; exit\,!\,(readchans, writechans)
\end{aligned}$$

$$\begin{aligned}
&MemRead\ (read_ch : int\ channel, v : int, return_ch : int\ channel) \mapsto \\
&\quad read_ch?x; return_ch\,!\,x
\end{aligned}$$

$$\begin{aligned}
&MemWrite\ (write_ch : int\ channel, value : int, return_ch : int\ channel) \mapsto \\
&\quad writech\,!\,value; returnch\,!\,1
\end{aligned}$$

$$
\begin{array}{l}
\mathit{MemServer}\ (\mathit{readchans} : \mathit{int\ channel\ list}, \mathit{writechans} : \mathit{int\ channel\ list} \\
\qquad \mathit{read} : (\mathit{int} \times \mathit{int} \times \mathit{int\ channel})\ \mathit{channel}, \\
\qquad \mathit{write} : (\mathit{int} \times \mathit{int} \times \mathit{int\ channel})\ \mathit{channel}) \mapsto \\
\quad \{\mathit{nth} \leftarrow \\
\qquad \mathit{let}\ \mathit{nth}_0 = \lambda f : \mathit{int} \to \mathit{int\ list} \to \mathit{int}.\lambda n : \mathit{int}.\lambda l : \mathit{int\ list}. \\
\qquad\qquad \mathit{if}\ n = 0\ \mathit{then}\ \mathit{head}\ l\ \mathit{else}\ f\ (n-1)\ (\mathit{tail}\ l) \\
\qquad \mathit{in}\ Y\ \mathit{nth}_0\}. \\
\quad (\quad \mathit{read}?\mathit{req}; \\
\qquad \{\mathit{read\_ch} \leftarrow \mathit{nth}\ (\#1\ \mathit{req})\ \mathit{readchans}, \mathit{return\_ch} \leftarrow \#3\ \mathit{req}\}. \\
\qquad \mathit{spawn}(\mathit{MemRead}(\mathit{read\_ch}, 0, \mathit{return\_ch})) \\
\quad + \quad \mathit{write}?\mathit{req}; \\
\qquad \{\mathit{write\_ch} \leftarrow \mathit{nth}\ (\#1\ \mathit{req})\ \mathit{writechans}, v \leftarrow \#2\ \mathit{req}, \mathit{return\_ch} \leftarrow \#3\ \mathit{req}\}. \\
\qquad \mathit{spawn}(\mathit{MemWrite}(\mathit{write\_ch}, v, \mathit{return\_ch}))); \\
\quad \mathit{MemServer}(\mathit{readchans}, \mathit{writechans}, \mathit{read}, \mathit{write})
\end{array}
$$

$$
\begin{array}{l}
\mathit{Memory}\ (n : \mathit{int}, \mathit{read} : \mathit{int} \times \mathit{int} \times \mathit{int\ channel}, \mathit{write} : \mathit{int} \times \mathit{int} \times \mathit{int\ channel}) \mapsto \\
\quad \textbf{ch}\ \mathit{lists\_ch}. \\
\qquad \mathit{spawn}(\mathit{CreateCells}(n, \mathit{nil}, \mathit{nil}, \mathit{lists\_ch})); \\
\qquad \mathit{lists\_ch}?\mathit{lists}; \\
\qquad\quad \mathit{spawn}(\mathit{MemServer}(\#1\ \mathit{lists}, \#2\ \mathit{lists}, \mathit{read}, \mathit{write}))
\end{array}
$$

The process *MemCell* describes the behaviour of a single memory cell. It has three parameters, containing respectively the current value of the cell and the two channels for read- and write-access. The process *CreateCells* is used to create the initial state of the memory component. It creates *count* instances of *MemCell*, each of them with corresponding channels. These channels are collected in two channel lists, which are finally sent to the caller of *CreateCells* using a local channel given as a parameter. *MemRead* and *MemWrite* are auxiliary processes used for parallelizing access to the memory cells. *MemServer* specifies the memory server. At the beginning, it gets the two lists of channels for interaction with the memory cells and the two channels *read* and *write* as parameters. In order to compute the corresponding channel of an adressed cell, the recursive function *nth* is defined which allows access to the elements of a list (the head of the list is indexed with 0). Requests on *read* or *write* must be tuples (adr, v, c), where *adr* is the address of the memory cell to be used, v specifies the value to be written in the cell (v is ignored in a read request), and c is an integer channel on which the result of the access is sent to the caller. After receiving a request on *read* or *write*, the tuple is decomposed using an assignment. Then a *MemRead* or *MemWrite*, resp., is spawned to handle the interaction with the addressed memory cell. The process *Memory* initializes the memory component. First it creates the memory cells using *CreateCells*. Then it creates an instance of *MemServer* with the local channels received from *CreateCells*.

The RPC component. This component can be specified as follows:

$$
\begin{array}{l}
\mathit{RPC\_Server}\ (\mathit{remote\_call} : \\
\qquad\qquad ((\mathit{int} \times \mathit{int} \times \mathit{int\ channel})\ \mathit{channel} \times \mathit{int} \times \mathit{int} \times \mathit{int\ channel})\ \mathit{channel}) \mapsto \\
\quad \mathit{remote\_call}?\mathit{req}; \mathit{spawn}(\mathit{RPC\_Client}(\mathit{req})); \mathit{RPC\_Server}(\mathit{remote\_call})
\end{array}
$$

$$
\begin{array}{l}
\mathit{RPC\_Client}\ (\mathit{req} : (\mathit{int} \times \mathit{int} \times \mathit{int\ channel})\ \mathit{channel} \times \mathit{int} \times \mathit{int} \times \mathit{int\ channel}) \mapsto \\
\quad \{\mathit{proc\_ch} \leftarrow \#1\ \mathit{req}, \mathit{adr} \leftarrow \#2\ \mathit{req}, \mathit{value} \leftarrow \#3\ \mathit{req}, \mathit{return\_ch} \leftarrow \#4\ \mathit{req}\}. \\
\quad\ \textbf{ch}\ \mathit{local\_ch}. \\
\quad\ \mathit{proc\_ch}\,!\,(\mathit{adr}, \mathit{value}, \mathit{local\_ch}); \mathit{local\_ch}?x; \mathit{return\_ch}\,!\,x
\end{array}
$$

The RPC-component receives the requests on the channel *rc*. Requests must be tuples $(\mathit{proc\_ch}, \mathit{adr}, v, c)$, where *proc_ch* is either the channel *read* or the channel *write*, and *adr*, v, and c have the same meaning as described previously. The requests are received by the

RPC_Server process, which then creates an instance of *RPC_Client* to handle the request. This is also done to enable the concurrent execution of more than one request at the same time. The *RPC_Client* decomposes the request using a binding operator. Then it sends the request via *read* or *write*, resp., to the memory component. As a return channel, a local channel *local_ch* is used. After receiving the result of the memory access on *local_ch*, the result is delivered to the user on the channel specified in the remote procedure call.

Now we give an example of proving properties using the axiomatic theory of Figure 4 and Figure 5. We assume a second system without the RPC-component, meaning that the user directly interacts with the memory component. Then we want to show that the two systems show equivalent behaviour w.r.t observation congruence.

In order to provide a similar interface for the user, we add a process *RPC_Simulate* to model the system without a remote procedure call component. This process decomposes the requests received on the remote call channel analogously to an *RPC_Client*, but does not accept the reply messages from the memory component. Thus, the results are delivered to the user directly.

$$
\begin{array}{l}
RPC\_Simulate\ (remote\_call : \\
\qquad ((int \times int \times int\ channel)\ channel \times int \times int \times int\ channel)\ channel) \mapsto \\
\quad remote\_call?req. \\
\qquad \{proc\_ch \leftarrow \#1\ req, adr \leftarrow \#2\ req, value \leftarrow \#3\ req, return\_ch \leftarrow \#4\ req\}. \\
\qquad\quad spawn(proc\_ch\,!\,(adr, value, return\_ch)); RPC\_Simulate(remote\_call)
\end{array}
$$

The following theorem states that, under the assumption that the memory component is working correctly, the system with the remote procedure call component can simulate the system in which the user directly accesses the memory.

Theorem 20. *The systems*

$$\mathbf{ch}\ rc.\ spawn(RPC\_Simulate(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$$

and

$$\mathbf{ch}\ rc.\ spawn(RPC\_Server(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$$

show the same behaviour w.r.t. weak congruence.

6 Conclusions

In this paper, we introduced a process calculus with mobility which allows a smooth incorporation of a data formalism. In order to enable a problem-oriented data description, we did not fix a specific data description language, but allow to use any data language which has a reduction semantics for expressions and which satisfies a small set of conditions.

In our calculus, we decided to use an unary operation for process creation instead of adopting the commonly used binary operator for parallel composition. Therefore, our calculus becomes more suitable as a basis for the semantical analysis of concurrent programming languages and libraries including process creation. The interaction of process creation and sequential composition in the setting of process algebra has been studied before by Baeten and Vaandrager in [4] and by Havelund and Larsen in [16, 17]. Only the latter address higher order features as well, also through name passing in the π-calculus style. Therefore, their calculus does not support the description of data structures and computation on data. Furthermore, their technical handling of process creation leads to a more involved operational semantics using two transition systems for describing the behaviour of single processes and the global system behaviour, respectively.

There are several proposals for a integration of the description of the behavioural and data aspects of reactive systems. For example, in [8] a calculus similar to *Concurrent ML* is described. In this calculus, the process language and the data language are integrated *symmetrically*: It is possible to use communication and parallel execution within expressions, which means that the concurrency semantics is part of the reduction semantics for expressions. This symmetrical integration leads to a more involved semantics; it is necessary to use higher-order bisimulation as an appropiate equivalence relation on terms. In our calculus, the *asymmetrical* integration of data and behaviour language clearly separates the reduction of expressions from the semantics of the behaviour language. This separation allows for a two-step analysis of systems: First, the data part can be analysed without regarding the process part. Second, the results of this analysis can then be integrated in the analysis of the process part. We claim that this restricted interface between the two paradigms eases the analysis of systems by allowing to adopt existing verification techniques.

In future work, we intend to investigate the integration of other data languages in our process calculus. Of special interest is the integration of object-oriented data languages and the examination of the relationship between the object-oriented concepts, like encapsulation, and the process behaviour of systems.

As a first step in the area of semantics for concurrent programming languages, the semantics of an object-based language with intra-object-concurrency has been defined using a subset of the calculus without data [9]. The object-based language and its semantics will be extended by the introduction of data.

Furthermore, we are interested in applying *action refinement* [1, 12, 13, 14, 15] as a method for stepwise development of systems to our calculus. While the theory of action refinement is well-understood for calculi without data, we aim to develop a framework for a combination of action refinement and data refinement.

References

1. L. Aceto and M. C. Hennessy. Towards Action-Refinement in Process Algebras. *Information and Computation*, 103:204–269, 1993.
2. K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.
3. J. Baeten and W. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.
4. J. C. M. Baeten and F. W. Vaandrager. An Algebra for Process Creation. *Acta Inf.*, 29(4):303–334, 1992. Report version: CS-R8907, CWI, Amsterdam; also available in "J.W. de Bakker, 25 Jaar Semantiek — Liber Amicorum", Stichting Mathematisch Centrum, Amsterdam, 1989.
5. H. Barendregt. Functional Programming and Lambda Calculus. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, pages 321–363. Elsevier, 1990.
6. M. Broy, S. Merz, and K. Spies, editors. *Formal System Specification: The RPC-Memory Specification*, volume 1169 of *Lecture Notes in Computer Science*. Springer, 1996.
7. W. Ferreira and M. Hennessy. Towards a Semantic Theory of CML. Technical Report 2/95, University of Sussex, Feb. 1995.
8. W. Ferreira, M. Hennessy, and A. Jeffrey. A Theory of Weak Bisimulation for Core CML. Technical Report 05/95, University of Sussex, Sept. 1995.
9. T. Gehrke. An Algebraic Semantics for an Abstract Language with Intra-Object-Concurrency. In D. Pritchard and J. Reeve, editors, *Proceedings of the 4th International Euro-Par Conference on Parallel Processing (Euro-Par '98)*, volume 1470 of *Lecture Notes in Computer Science*, pages 733–737. Springer, 1998.
10. T. Gehrke and M. Huhn. ProFun – a Language for Executable Specifications. In H. Kuchen and S. Swierstra, editors, *Proceedings of the 8th International Symposium on Programming Languages: Implementations, Logics and Programs (PLILP '96)*, volume 1140 of *Lecture Notes in Computer Science*, pages 304–318. Springer, 1996.
11. A. Giacalone, P. Mishra, and S. Prasad. Facile: A symmetric integration of concurrent and functional programming. *International Journal of Parallel Programming*, 18(2), 1989.
12. U. Goltz, R. Gorrieri, and A. Rensink. Comparing Syntactic and Semantic Action Refinement. *Information and Computation*, 125:118–143, 1996.
13. U. Goltz and R. van Glabbeek. Refinement of Actions and Equivalence Notions for Concurrent Systems. Hildesheimer Informatik-Bericht 6/98, University of Hildesheim, 1998.
14. U. Goltz and R. J. van Glabbeek. Equivalence Notions for Concurrent Systems and Refinement of Actions. In *Proceedings of MFCS '89*, Lecture Notes in Computer Science, pages 237–248. Springer, 1989.
15. R. Gorrieri and A. Rensink. Action refinement. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 1999. To appear.
16. K. Havelund. *The Fork Calculus*. PhD thesis, DIKU, University of Kopenhagen, 1994.
17. K. Havelund and K. G. Larsen. The Fork-Calculus. *Nordic Journal of Computing*, 1:346–363, 1994.
18. C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
19. P. Hudak, S. Peyton Jones, P. Wadler (editors), B. Boutel, J. Fairbairn, J. Fasel, M. M. Guzmán, K. Hammond, J. Hughes, Y. Johnsson, D. Kieburtz, R. Nikhil, W. Partian, and J. Peterson. Haskell: A non-strict, purely functional language. *SIGPLAN Notices*, 27(5), May 1992.
20. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
21. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I+II. *Information and Computation*, 100, 1992.
22. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, 1990.
23. J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120:174–197, 1995.
24. J. Parrow and B. Victor. The Fusion Calculus. In *Proceedings of LICS '98*. IEEE Press, 1998.
25. B. C. Pierce and D. N. Turner. Pict: A Programming Language Based on the Pi-Calculus. CSCI Technical Report 476, Indiana University, 1997.
26. J. Reppy. Concurrent ML: Design, application and semantics. In *Programming, Concurrency, Simulation and Automated Reasoning*, volume 693 of *Lecture Notes in Computer Science*, pages 165–198. Springer, 1992.
27. B. Stroustrup. *The C++ Programming Language*. Addison–Wesley, 1997.
28. Sun Microsystems GmbH. *Multithreaded Programming Guide*, 1994.

29. B. Thomsen, L. Leth, and A. Giacalone. Some Issues in the Semantics of Facile Distributed Programming. In *Proceedings of REX Workshop "Semantics: Foundations and Applications"*, volume 666 of *Lecture Notes in Computer Science*. Springer, 1992.

A Proofs

With $v =_\beta v'$ we know that $fc(v) = fc(v')$. In order to prove Theorem 3, first we give an auxiliary lemma.

Lemma 21. *If* $\Delta, \mathbf{x} : \mathbf{T} \vdash t : \mathit{proc}$ *and* $\emptyset \vdash \mathbf{v} : \mathbf{T}$, *then* $\Delta \vdash t\langle \mathbf{v}/\mathbf{x} \rangle : \mathit{proc}$.

Proof. By induction over the term structure of t. Substitutions distribute over all operators of TERM. All free occurences of $\mathbf{x}$ in t are replaced by values of the same type. Therefore, type assumptions $\mathbf{x} : \mathbf{T}$ can be removed from Δ. Furthermore, if $t = \mathbf{ch}\ C.\ u$ and $fc(\mathbf{v}) \cap C \neq \emptyset$, we apply alpha-conversion to the channel names in C and u. □

Theorem 3. Assume $\vdash t : \mathit{proc}$.

- If $t \xrightarrow{c?v} t'$ then $\vdash c : T\ \mathit{channel}$ for some T; if $\vdash v : T$ then $\vdash t' : \mathit{proc}$.
- If $t \xrightarrow{c!v,C} t'$ then $\vdash c : T\ \mathit{channel}$ and $\vdash v : T$ for some T and $\vdash t' : \mathit{proc}$.
- If $t \xrightarrow{\omega} t'$ and $\omega = \tau$ or $\omega = \checkmark$, then $\vdash t' : \mathit{proc}$.

Proof. We assume $\vdash t : \mathit{proc}$, $t \xrightarrow{\omega} t'$ and prove the theorem for the rules of the operational semantics in Figure 3.

- Rule R_1. Then $t = \mathbf{1}$, $\omega = \checkmark$ and $t' = \mathbf{1}$. With T_3, T_1 we know that $\vdash \mathbf{1} : \mathit{proc}$. Therefore, we have $\vdash t' : \mathit{proc}$.
- Rule R_2. Then $t = \tau$, $\omega = \tau$ and $t' = \mathbf{1}$. With T_3, T_1 we know that $\vdash t' : \mathit{proc}$.
- Rule R_3. Then $t = [E]$, $[\![E]\!] = \mathit{true}$, $\omega = \tau$ and $t' = \mathbf{1}$. Futhermore, we know with T_3, T_1 that $\vdash t' : \mathit{proc}$.
- Rule R_4. Then $t = E!F$, $\omega = [\![E]\!]![\![F]\!]$ and $t' = \mathbf{1}$. With T_6 we know that $\vdash E : T\ \mathit{channel}$ and $\vdash F : T$. Furthermore, we know with T_3, T_1 that $\vdash t' : \mathit{proc}$.
- Rule R_5. Then $t = E?x; t$, $\omega = [\![E]\!]?v$ and $t' = t\langle v/x \rangle$. With T_7 we can deduce that $\vdash E : T\ \mathit{channel}$ and $x : T \vdash t : \mathit{proc}$. Then we know with Lemma 21 and $\vdash v : T$ that $\vdash t' : \mathit{proc}$.
- Rule R_{14}. Then $t = \mathit{spawn}(u)$, $\omega = \checkmark$ and $t' = \mathit{spawn}(u)$. With T_{12} we know that $\vdash t' : \mathit{proc}$.
- Rule R_{15}. Then $t = \mathit{spawn}(u)$, $u \xrightarrow{\alpha} u'$ and $t' = \mathit{spawn}(u')$. We make an appropiate distinction for α. With T_{12} and $\vdash \mathit{spawn}(u) : \mathit{proc}$ we know that $\vdash u : \mathit{proc}$. With the induction hypothesis we can deduce that $\vdash u' : \mathit{proc}$. Therefore, we know with T_{12} that $\vdash \mathit{spawn}(u') : \mathit{proc}$.
- Rule R_6. Then $t = \{\mathbf{x} \leftarrow \mathbf{E}\}; u$, $[\![\mathbf{E}]\!] = \mathbf{v}$, $u\langle \mathbf{v}/\mathbf{x} \rangle \xrightarrow{\omega} u'$ and $t' = u'$. We make an appropiate distinction for ω. With T_8 we can deduce that $\vdash \mathbf{E} : \mathbf{T}$ and $\mathbf{x} : \mathbf{T} \vdash u : \mathit{proc}$. Then we can deduce with the $[\![\mathbf{E}]\!] = \mathbf{v}$ and Lemma 21 that $\vdash u\langle \mathbf{v}/\mathbf{x} \rangle : \mathit{proc}$. Therefore, we can deduce with the induction hypothesis that $\vdash u' : \mathit{proc}$.
- Rule R_7. Then $t = t_1 + t_2$ and $t_1 \xrightarrow{\omega} t'$. We make an appropiate distinction for ω. With T_9 and $\vdash t_1 + t_2 : \mathit{proc}$ we can deduce that $\vdash t_1 : \mathit{proc}$. Therefore, we can deduce with the induction hypothesis that $\vdash t' : \mathit{proc}$.
- Rule R_8. Similarly to the proof of R_7.
- Rule R_{16}. Then $t = P(\mathbf{E})$, $[\![\mathbf{E}]\!] = \mathbf{v}$, $P(\mathbf{x} : \mathbf{T}) \mapsto u \in \Theta$, $\omega = \tau$ and $t' = u\langle \mathbf{v}/\mathbf{x} \rangle$. With T_{13} we know that $\vdash \mathbf{E} : \mathbf{T}$, $fc(t) = \emptyset$ and $\mathbf{x} : \mathbf{T} \vdash u : \mathit{proc}$. Then we can deduce with $[\![\mathbf{E}]\!] = \mathbf{v}$ and Lemma 21 that $\vdash u\langle \mathbf{v}/\mathbf{x} \rangle : \mathit{proc}$.
- Rule R_9. Then $t = t_1; t_2$, $t_1 \xrightarrow{\alpha} t_1'$, $\omega = \alpha$ and $t' = t_1'; t_2$. We make an appropiate distinction for α. With T_{10} we know that $\vdash t_1 : \mathit{proc}$ and $\vdash t_2 : \mathit{proc}$. Then we can deduce with the induction hypothesis that $\vdash t_1' : \mathit{proc}$. Therefore, we have $\vdash t_1'; t_2 : \mathit{proc}$, derived with T_{10}.
- Rule R_{10}. Then $t = t_1; t_2$, $t_1 \xrightarrow{\checkmark} t_1'$, $t_2 \xrightarrow{\omega} t_2'$ and $t' = t_1'; t_2'$. We make an appropiate distinction for ω. With T_{10} we know that $\vdash t_1 : \mathit{proc}$ and $\vdash t_2 : \mathit{proc}$. Then we can deduce with the induction hypothesis that $\vdash t_1' : \mathit{proc}$ and $\vdash t_2' : \mathit{proc}$. Therefore, we know with T_{10} that $\vdash t_1'; t_2' : \mathit{proc}$.

– Rule R_{11}: Then $t = t_1; t_2$, $t_1 \xrightarrow{\checkmark} t_1'$, $t_1' \xrightarrow{\alpha} t_1''$, $t_2 \xrightarrow{\alpha'} t_2'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $t' = \mathbf{ch}\ C.\ t_1''; t_2'$. With T_{10} we know that $\vdash t_1 : \mathit{proc}$ and $\vdash t_2 : \mathit{proc}$. Then we can deduce with the induction hypothesis that $\vdash t_1' : \mathit{proc}$, $\vdash t_1'' : \mathit{proc}$ and $\vdash t_2' : \mathit{proc}$. Therefore, we can deduce with T_{10} that $\vdash t_1''; t_2' : \mathit{proc}$. Then we can deduce with T_{11} that $\vdash \mathbf{ch}\ C.\ t_1''; t_2' : \mathit{proc}$.
– Rule R_{12}: Then $t = \mathbf{ch}\ C.\ u$, $u \xrightarrow{\omega} u'$, $\mathit{fc}(\omega) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.\ u'$. We make an appropiate distinction for ω. With T_{11} we know that $\vdash u : \mathit{proc}$. With the induction hypothesis we can deduce that $\vdash u' : \mathit{proc}$. Therefore, we know with T_{11} that $\vdash \mathbf{ch}\ C.u' : \mathit{proc}$.
– Rule R_{13}: Then $t = \mathbf{ch}\ C.\ u$, $u \xrightarrow{c!v,A} u'$, $c \notin C$, $A \cap C = \emptyset$, $\mathit{fc}(v) \cap C \neq \emptyset$, $\omega = (c!v, A \cup (\mathit{fc}(v) \cap C))$ and $t' = \mathbf{ch}\ C \setminus \mathit{fc}(v).\ u'$. With T_{11} we know that $\vdash u : \mathit{proc}$, $\vdash c : T\mathit{channel}$ and $\vdash v : T$. With the induction hypothesis we can deduce that $\vdash u' : \mathit{proc}$. Then we know that $\vdash \mathbf{ch}\ C \setminus \mathit{fc}(v).\ u' : \mathit{proc}$, derived with T_{11}. □

Theorem 6. $\sim_\beta$ is a congruence for the operations of TERM.

Proof. We prove the congruence property of $\sim_\beta$ for the operations of TERM. For each step $t \xrightarrow{c!v,C} t'$ we assume that $C \cap \mathit{fc}(t) = \emptyset$. First, we prove the cases for which Lemma 7 is not necessary. After the proof of Lemma 7, we prove the remaining cases.

– $\mathit{spawn}(t) \sim_\beta \mathit{spawn}(u)$ if $t \sim_\beta u$.
Let $R = \{(\mathit{spawn}(t_0); \mathit{spawn}(u_0)) \mid t_0 \sim_\beta u_0\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetric, we only have to show one direction.
We assume $\mathit{spawn}(t_0) \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $t_0 \xrightarrow{c?v} t_0'$, $\omega = c?v$ and $t' = \mathit{spawn}(t_0')$, derived with R_{15}. With $t_0 \sim_\beta u_0$ we know that $\forall v', v =_\beta v', \exists u_0' : u_0 \xrightarrow{c?v'} u_0'$ and $t_0' \sim_\beta u_0'$. Then we can deduce with R_{15} that $\mathit{spawn}(u_0) \xrightarrow{c?v'} \mathit{spawn}(u_0')$. Furthermore, $(\mathit{spawn}(t_0'), \mathit{spawn}(u_0')) \in R$.
  - $t_0 \xrightarrow{\alpha} t_0'$, $\alpha = \tau$ or $\alpha = c!v, C$, $\omega = \alpha$ and $t' = \mathit{spawn}(t_0')$. With $t_0 \sim_\beta u_0$ we know that $\exists \omega', \omega =_\beta \omega', \exists u_0' : u_0 \xrightarrow{\omega'} u_0'$ and $t_0' \sim_\beta u_0'$. Then we can derive with R_{15} that $\mathit{spawn}(u_0) \xrightarrow{\omega'} \mathit{spawn}(u_0')$. Furthermore, $(\mathit{spawn}(t_0'), \mathit{spawn}(u_0')) \in R$.
  - $\omega = \checkmark$ and $t' = \mathit{spawn}(t_0)$, derived with R_{14}. Similarly, we can derive with R_{14} that $\mathit{spawn}(u_0) \xrightarrow{\checkmark} \mathit{spawn}(u_0)$. Furthermore, $(\mathit{spawn}(t_0), \mathit{spawn}(u_0)) \in R$.
– $t_1 + u \sim_\beta t_2 + u$ if $t_1 \sim_\beta t_2$.
Let $R = \{(u_1 + u, u_2 + u) \mid u_1 \sim_\beta u_2\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetrc, we only have to show one direction.
We assume $u_1 + u \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
  - $u_1 \xrightarrow{c?v} u_1'$, $\omega = c?v$ and $u_{new} = u_1'$, derived with R_7. With $u_1 \sim_\beta u_2$ we know that $\forall v', v =_\beta v', \exists u_2' : u_2 \xrightarrow{c?v'} u_2'$ and $u_1' \sim_\beta u_2'$. Then we can deduce with R_7 that $u_2 + u \xrightarrow{c?v'} u_2'$. Furthermore, $(u_1', u_2') \in R$.
  - $u_1 \xrightarrow{\omega} u_1'$, $\omega \neq c?v$ and $u_{new} = u_1'$, derived with R_7. With $u_1 \sim_\beta u_2$ we know that $\exists \omega', \omega =_\beta \omega', \exists u_2' : u_2 \xrightarrow{\omega'} u_2'$ and $u_1' \sim_\beta u_2'$. Then we can deduce with with R_7 that $u_2 + u \xrightarrow{\omega'} u_2'$. Furthermore, $(u_1', u_2') \in R$.
  - $u \xrightarrow{\omega} u'$ and $u_{new} = u'$, derived with R_8. Similarly, we can derive with R_8 that $u_2 + u \xrightarrow{\omega} u'$. Furthermore, $(u', u') \in R$.
– $\mathbf{ch}\ C.\ t \sim_\beta \mathbf{ch}\ C.\ u$ if $t \sim_\beta u$.
Let $R = \{(\mathbf{ch}\ D.\ t_0, \mathbf{ch}\ D.\ u_0) \mid t_0 \sim_\beta u_0\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetric, we only have to show one direction.
We assume $\mathbf{ch}\ C.\ t_0 \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $t_0 \xrightarrow{c?v} t_0'$, $\omega = c?v$, $c \notin C$, $\mathit{fc}(v) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.\ t_0'$, derived with R_{12}. With $t_0 \sim_\beta u_0$ we know that $\forall v', v =_\beta v', \exists u_0' : u_0 \xrightarrow{c?v'} u_0'$ and $t_0' \sim_\beta u_0'$. Furthermore,

with $v =_\beta v'$ we know that $fc(v) = fc(v')$; therefore, we can deduce that $fc(v') \cap C = \emptyset$. Therefore, we know with R_{12} that $\mathbf{ch}\ u_0.\ \xrightarrow{c?v'}\ \mathbf{ch}\ C.\ u'_0$. Furthermore, $(\mathbf{ch}\ C.\ t'_0, \mathbf{ch}\ C.\ u'_0) \in R$.

- $t_0 \xrightarrow{\omega} t'_0$, $\omega \neq c?v$, $fc(\omega) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.\ t'_0$, derived with R_{12}. With $t_0 \sim_\beta u_0$ we know that $\exists \omega', \omega =_\beta \omega', \exists u'_0 : u_0 \xrightarrow{\omega'} u'_0$ and $t'_0 \sim_\beta u'_0$. Furthermore, with $\omega =_\beta \omega'$ we know that $fc(\omega) = fc(\omega')$; therefore, we can deduce that $fc(\omega') \cap C = \emptyset$. Therefore, we know with R_{12} that $\mathbf{ch}\ C.u_0 \xrightarrow{\omega'} \mathbf{ch}\ C.u'_0$. Furthermore, $(\mathbf{ch}\ C.t'_0, \mathbf{ch}\ C.u'_0) \in R$.
- $t_0 \xrightarrow{c!v,A} t'_0$, $A \cap C = \emptyset, fc(v) \cap C \neq \emptyset, \omega = (c!v, A \cup (C \cap fc(v)))$ and $t' = \mathbf{ch}\ C \backslash fc(v).t'_0$, derived with R_{13}. With $t_0 \sim_\beta u_0$ we know that $\exists v', v =_\beta v', \exists u'_0 : u_0 \xrightarrow{c!v',A} u'_0$ and $t'_0 \sim_\beta u'_0$. Furthermore, with $v =_\beta v'$ we know that $fc(v) = fc(v')$; therefore, we can derive that $fc(v') \cap C = fc(v) \cap C \neq \emptyset$. Therefore, we can deduce with R_{13} that $\mathbf{ch}\ C.\ u_0 \xrightarrow{c!v', A \cup (fc(v) \cap C)}\ \mathbf{ch}\ C \setminus fc(v).\ u'_0$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ t'_0, \mathbf{ch}\ C \setminus fc(v).\ u'_0) \in R$.

– $t_1; t_2 \sim_\beta u_1; u_2$ if $t_1 \sim_\beta u_1$ and $t_2 \sim_\beta u_2$.
Let $R = \{(\mathbf{ch}\ D.\ t_1; t_2, \mathbf{ch}\ D.\ u_1; u_2) \mid t_1 \sim_\beta u_1, t_2 \sim_\beta u_2\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetric, we only have to show one direction.
First, we assume $D = \emptyset$. For $D \neq \emptyset$ we can apply proof techniques similar to the congruence proof for channel creation. We assume $t_1; t_2 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:

- $t_1 \xrightarrow{c?v} t'_1$, $\omega = c?v$ and $t_{new} = t'_1; t_2$, derived with R_9. With $t_1 \sim_\beta u_1$ we know that $\forall v', v =_\beta v', \exists u'_1 : u_1 \xrightarrow{c?v'} u'_1$ and $t'_1 \sim_\beta u'_1$. Then we can deduce with R_9 that $u_1; u_2 \xrightarrow{c?v'} u'_1; u_2$. Furthermore, $(t'_1; t_2, u'_1; u_2) \in R$.
- $t_1 \xrightarrow{\alpha} t'_1$, $\alpha \neq c?v$, $\omega = \alpha$ and $t_{new} = t_1; t_2$, derived with R_9. With $t_1 \sim_\beta u_1$ we know that $\exists \omega', \omega =_\beta \omega', \exists u'_1 : u_1 \xrightarrow{\omega'} u'_1$ and $t'_1 \sim_\beta u'_1$. Then we can deduce with R_9 that $u_1; u_2 \xrightarrow{\omega'} u'_1; u_2$. Furthermore, $(t'_1; t_2, u'_1; u_2) \in R$.
- $t_1 \xrightarrow{\checkmark} t'_1$, $t_2 \xrightarrow{c?v} t'_2$, $\omega = c?v$ and $t_{new} = t'_1; t'_2$, derived with R_{10}. With $t_1 \sim_\beta u_1$ we know that $\exists u'_1 : u_1 \xrightarrow{\checkmark} u'_1$ and $t'_1 \sim_\beta u'_1$. With $t_2 \sim_\beta u_2$ we know that $\forall v', v =_\beta v', \exists u'_2 : u_2 \xrightarrow{c?v'} u'_2$ and $t'_2 \sim_\beta u'_2$. Therefore, we can deduce with R_{10} that $u_1; u_2 \xrightarrow{c?v'} u'_1; u'_2$. Furthermore, $(t'_1; t'_2, u'_1; u'_2) \in R$.
- $t_1 \xrightarrow{\checkmark} t'_1$, $t_2 \xrightarrow{\omega} t'_2$, $\omega \neq c?v$ and $t_{new} = t'_1; t'_2$, derived with R_{10}. With $t_1 \sim_\beta u_1$ we know that $\exists u'_1 : u_1 \xrightarrow{\checkmark} u'_1$ and $t'_1 \sim_\beta u'_1$. With $t_2 \sim_\beta u_2$ we know that $\exists \omega', \omega =_\beta \omega', \exists u'_2 : u_2 \xrightarrow{\omega'} u'_2$ and $t'_2 \sim_\beta u'_2$. Therefore, we can deduce with R_{10} that $u_1; u_2 \xrightarrow{\omega'} u'_1; u'_2$. Furthermore, $(t'_1; t'_2, u'_1; u'_2) \in R$.
- $t_1 \xrightarrow{\checkmark} t'_1$, $t'_1 \xrightarrow{c?v} t''_1$, $t_2 \xrightarrow{c!v,C} t'_2$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ t''_1; t'_2$, derived with R_{11}.With $t_2 \sim_\beta u_2$ we know that $\exists v', v =_\beta v', \exists u'_2 : u_2 \xrightarrow{c!v',C} u'_2$ and $t'_2 \sim_\beta u'_2$. With $t_1 \sim_\beta u_1$ we know that $\exists u'_1 : u_1 \xrightarrow{\checkmark} u'_1$, $t'_1 \sim_\beta u'_1$ and $\forall v'', v =_\beta v'', \exists u''_1 : u'_1 \xrightarrow{c?v''} u''_1$, $t''_1 \sim_\beta u''_1$. Then we know with $v =_\beta v'$ that $\exists u_v : u'_1 \xrightarrow{c?v'} u_v$ and $t''_1 \sim_\beta u_v$. Therefore, we can deduce with R_{11} that $u_1; u_2 \xrightarrow{\tau} \mathbf{ch}\ C.\ u_v; u'_2$. Furthermore, $(\mathbf{ch}\ C.\ t''_1; t'_2, \mathbf{ch}\ C.\ u_v; u'_2) \in R$.
- $t_1 \xrightarrow{\checkmark} t'_1$, $t'_1 \xrightarrow{c!v,C} t''_1$, $t_2 \xrightarrow{c?v} t'_2$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ t''_1; t'_2$, derived with R_{11}. With $t_1 \sim_\beta u_1$ we know that $\exists u'_1 : u_1 \xrightarrow{\checkmark} u'_1$, $t'_1 \sim_\beta u'_1$ and $\exists v', v =_\beta v', \exists u''_1 : u'_1 \xrightarrow{c!v',C} u''_1$, $t''_1 \sim_\beta u''_1$. With $t_2 \sim_\beta u_2$ we know that $\forall v'', v =_\beta v'', \exists u'_2 : u_2 \xrightarrow{c?v''} u'_2$ and $t'_2 \sim_\beta u'_2$. Then we can deduce with $v =_\beta v'$ that $\exists u_v : u_2 \xrightarrow{c?v'} u_v$ and $t'_2 \sim_\beta u_v$. Therefore, we can deduce with R_{11} that $u_1; u_2 \xrightarrow{\tau} \mathbf{ch}\ C.\ u''_1; u_v$. Furthermore, $(\mathbf{ch}\ C.\ t''_1; t'_2, \mathbf{ch}\ C.\ u''_1, u_v) \in R$. □

The proof for the remaining operators makes use of Lemma 7, therefore we first have to prove this lemma.

Lemma 7. If $\sigma =_\beta \sigma'$, then $t\sigma \sim_\beta t\sigma'$.

Proof. By induction over the term structure of t. Let $R = \{(t\sigma_1, t\sigma_1') \mid t \in \text{TERM}, \sigma =_\beta \sigma'\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions. For each step $t \xrightarrow{c!v,C} t'$ we assume that $C \cap fc(t) = \emptyset$. Since R is symmetric, we need to show only one direction of bisimulation.

- For $t = \mathbf{0}$, $t = \mathbf{1}$ and $t = \tau$ trivial.
- $t = [E]$. With $\sigma =_\beta \sigma'$ we know that $\exists E', E'', E' =_\beta E'' : t\sigma = [E']$ and $t\sigma' = [E'']$. With $E' =_\beta E''$ we know that $[\![E']\!] = [\![E'']\!]$; therefore, if $[E'] \xrightarrow{\tau} \mathbf{1}$, then $[E''] \xrightarrow{\tau} \mathbf{1}$, derived with R_3. Furthermore, $(\mathbf{1}, \mathbf{1}) \in R$.
- $t = E!F$. With $\sigma =_\beta \sigma'$ we know that $\exists E', E'', E' =_\beta E'', \exists F', F'', F' =_\beta F'' : t\sigma = E'!F'$ and $t\sigma' = E''!F''$. For channels beta-equivalence is identity; therefore $\exists c : [\![E']\!] = c = [\![E'']\!]$. With $F' =_\beta F''$ we know that $\exists v, v', v =_\beta v'$: $[\![F']\!] = v$ and $[\![F'']\!] = v'$. Then we can deduce with R_4 that $t\sigma \xrightarrow{c!v} \mathbf{1}$, $t\sigma' \xrightarrow{c!v'} \mathbf{1}$ and $c!v =_\beta c!v'$. Furthermore, $(\mathbf{1}, \mathbf{1}) \in R$.
- $t = E?x; u$. With $\sigma =_\beta \sigma'$ we know that $\exists E', E'', E' =_\beta E''$, $t\sigma = E'?x; u\sigma$ and $t\sigma' = E''?x; u\sigma'$. For channels beta-equivalence is identity; therefore $\exists c : [\![E']\!] = c = [\![E'']\!]$. Then we know with R_5 that $E'?x; u\sigma \xrightarrow{c?v} u\sigma\langle v/x\rangle$ and $\forall v', v =_\beta v' : E''?x; u\sigma' \xrightarrow{c?v'} u\sigma'\langle v'/x\rangle$. Furthermore, $(u\sigma\langle v/x\rangle, u\sigma'\langle v'/x\rangle) \in R$.
- $t = \{\mathbf{x} \leftarrow \mathbf{E}\}; u$. With $\sigma =_\beta \sigma'$ we know that $\exists \mathbf{E}', \mathbf{E}'', \mathbf{E}' =_\beta \mathbf{E}'' : t\sigma = \{\mathbf{x} \leftarrow \mathbf{E}'\}; u\sigma$ and $t\sigma' = \{\mathbf{x} \leftarrow \mathbf{E}''\}; u\sigma'$. Let $[\![\mathbf{E}']\!] = \mathbf{v}'$ and let $[\![\mathbf{E}'']\!] = \mathbf{v}''$. With $\mathbf{E}' =_\beta \mathbf{E}''$ we know that $\mathbf{v}' =_\beta \mathbf{v}''$. We assume $t\sigma \xrightarrow{\omega} u'$. It follows that $u\sigma\langle \mathbf{v}'/\mathbf{x}\rangle \xrightarrow{\omega} u'$. With the induction hypothesis we know that $u\sigma\langle \mathbf{v}'/\mathbf{x}\rangle \sim_\beta u\sigma'\langle \mathbf{v}''/\mathbf{x}\rangle$. Therefore, with an appropiate distiction for ω we know that $\exists u'' : u\sigma'\langle \mathbf{v}''/\mathbf{x}\rangle \xrightarrow{\omega'} u''$ and $u' \sim_\beta u''$. Therefore, we can deduce with R_6 that $t\sigma' \xrightarrow{\omega} u''$; furthermore, $(u', u'') \in R$.
- $t = \mathit{spawn}(u)$. Then $t\sigma = \mathit{spawn}(u\sigma)$ and $t\sigma' = \mathit{spawn}(u\sigma')$. With the induction hypothesis we know that $u\sigma \sim_\beta u\sigma'$. We assume $t\sigma \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $u\sigma \xrightarrow{c?v} u'$, $\omega = c?v$ and $t' = \mathit{spawn}(u')$, derived with R_{15}. With $u\sigma \sim_\beta u\sigma'$ we know that $\forall v', v =_\beta v', \exists u'' : u\sigma' \xrightarrow{c?v'} u''$ and $u' \sim_\beta u''$. Then we can deduce with R_{15} that $\mathit{spawn}(u\sigma') \xrightarrow{c?v'} \mathit{spawn}(u'')$. Furthermore, $(\mathit{spawn}(u'), \mathit{spawn}(u'')) \in R$ due to the congruence of $\sim_\beta$ with respect to *spawn* (see above).
  - $u\sigma \xrightarrow{\alpha} u'$, $\alpha \neq c?v$, $\omega = \alpha$ and $t' = \mathit{spawn}(u')$, derived with R_{15}. With $u\sigma \sim_\beta u\sigma'$ we know that $\exists \alpha', \alpha =_\beta \alpha', \exists u'' : u\sigma' \xrightarrow{\alpha'} u''$ and $u' \sim_\beta u''$. Then we can derive with R_{15} that $\mathit{spawn}(u\sigma') \xrightarrow{\alpha'} \mathit{spawn}(u'')$. Furthermore, $(\mathit{spawn}(u'), \mathit{spawn}(u'')) \in R$.
  - $\omega = \checkmark$ and $t' = \mathit{spawn}(u\sigma)$, derived with R_{14}. Similarly, we can derive with R_{14} that $\mathit{spawn}(u\sigma') \xrightarrow{\checkmark} \mathit{spawn}(u\sigma')$. Furthermore, $(\mathit{spawn}(u\sigma), \mathit{spawn}(u\sigma')) \in R$.
- $t = t_1 + t_2$. Then $t\sigma = (t_1 + t_2)\sigma = t_1\sigma + t_2\sigma$ and $t\sigma' = (t_1 + t_2)\sigma' = t_1\sigma' + t_2\sigma'$. With the induction hypothesis we know that $t_1\sigma \sim_\beta t_1\sigma'$ and $t_2\sigma \sim_\beta t_2\sigma'$. We assume $t\sigma \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $t_1\sigma \xrightarrow{c?v} t_1'$, $\omega = c?v$ and $t' = t_1'$, derived with R_7. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\forall v', v =_\beta v', \exists t_1'' : t_1\sigma' \xrightarrow{c?v'} t_1''$ and $t_1' \sim_\beta t_1''$. Then we can deduce with R_7 that $t_1\sigma' + t_2\sigma' \xrightarrow{c?v'} t_1''$. Furthermore, $(t_1', t_1'') \in R$.
  - $t_1\sigma \xrightarrow{\omega} t_1'$, $\omega \neq c?v$ and $t' = t_1'$, derived with R_7. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists \omega', \omega =_\beta \omega', \exists t_1'' : t_1\sigma' \xrightarrow{\omega'} t_1''$ and $t_1' \sim_\beta t_1''$. Then we can deduce with R_7 that $t_1\sigma' + t_2\sigma' \xrightarrow{\omega'} t_1''$. Furthermore, $(t_1', t_1'') \in R$.

  For the case $t_2\sigma \xrightarrow{\omega} t_2'$ the proof is analogous.
- $t = \mathbf{ch}\ C.\ u$. If $C \cap \mathrm{rng}(\sigma) \neq \emptyset$ or $C \cap \mathrm{rng}(\sigma') \neq \emptyset$, we apply alpha-conversion to t. We have $t\sigma = \mathbf{ch}\ C.\ u\sigma$ and $t\sigma' = \mathbf{ch}\ C.\ u\sigma'$. With the induction hypothesis we know that $u\sigma \sim_\beta u\sigma'$. We assume $t\sigma \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $u\sigma \xrightarrow{c?v} u'$, $c \notin C$, $fc(v) \cap C = \emptyset$, $\omega = c?v$ and $t' = \mathbf{ch}\ C.\ u'$, derived with R_{12}. With $u\sigma \sim_\beta u\sigma'$ we know that $\forall v', v =_\beta v', \exists u'' : u\sigma' \xrightarrow{c?v'} u''$ and $u' \sim_\beta u''$. With $v =_\beta v'$ we know that $fc(v) = fc(v')$. Then we can deduce with R_{12} that

$\mathbf{ch}\ C.u\sigma' \xrightarrow{c?v'} \mathbf{ch}\ C.u''$. Furthermore, $(\mathbf{ch}\ C.u', \mathbf{ch}\ C.u'') \in R$ due to the congruence of $\sim_\beta$ with respect to restriction (see above).

- $u\sigma \xrightarrow{\omega} u', \omega \neq c?v, fc(\omega) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.u'$, derived with R_{12}. With $u\sigma =_\beta u\sigma'$ we know that $\exists \omega', \omega' =_\beta \omega, \exists u'' : u\sigma' \xrightarrow{\omega'} u''$ and $u' \sim_\beta u''$. With $\omega =_\beta \omega'$ we know that $fc(\omega) = fc(\omega')$. Then we can derive with R_{12} that $\mathbf{ch}\ C.\ u\sigma' \xrightarrow{\omega'} \mathbf{ch}\ C.\ u''$. Furthermore, $(\mathbf{ch}\ C.\ u', \mathbf{ch}\ C.\ u'') \in R$.
- $u\sigma \xrightarrow{c!v,A} u'$, $c \notin C$, $A \cap C = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = A \cup (C \cap fc(v))$ and $t' = \mathbf{ch}\ C \setminus fc(v).\ u'$, derived with R_{13}. With $u\sigma \sim_\beta u\sigma'$ we know that $\exists v', v =_\beta v', \exists u'' : u\sigma' \xrightarrow{c!v',A} u''$ and $u' \sim_\beta u''$. With $v =_\beta v'$ we know that $fc(v) = fc(v')$. Then we can deduce with R_{13} that $\mathbf{ch}\ C.\ u\sigma' \xrightarrow{c!v',A\cup(C\cap fc(v))} \mathbf{ch}\ C \setminus fc(v).\ u''$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ u', \mathbf{ch}\ C \setminus fc(v).\ u'') \in R$.

– $t = P(\mathbf{E})$. Let $\Theta : P(\mathbf{x} : \mathbf{T}) \mapsto u$. With $\sigma =_\beta \sigma'$ we know that $\exists \mathbf{E}', \mathbf{E}'', \mathbf{E}' =_\beta \mathbf{E}'' : t\sigma = P(\mathbf{E}')$ and $t\sigma' = P(\mathbf{E}'')$. Let $\mathbf{v}' = [\![\mathbf{E}']\!]$ and let $\mathbf{v}'' = [\![\mathbf{E}'']\!]$. With $\mathbf{E}' =_\beta \mathbf{E}''$ we know that $\mathbf{v}' =_\beta \mathbf{v}''$. With R_{16} we can deduce that $P(\mathbf{E}') \xrightarrow{\tau} u\langle \mathbf{v}'/\mathbf{x} \rangle$ and $P(\mathbf{E}'') \xrightarrow{\tau} u\langle \mathbf{v}''/\mathbf{x} \rangle$. Furthermore, $(u\langle \mathbf{v}'/\mathbf{x} \rangle, u\langle \mathbf{v}''/\mathbf{x} \rangle) \in R$.

– $t = t_1; t_2$. Then $t\sigma = (t_1; t_2)\sigma = t_1\sigma; t_2\sigma$ and $t\sigma' = (t_1; t_2)\sigma' = t_1\sigma'; t_2\sigma'$. With the induction hypothesis we know that $t_1\sigma \sim_\beta t_1\sigma'$ and $t_2\sigma \sim_\beta t_2\sigma'$. We assume $t\sigma \xrightarrow{\omega} t'$ and distinguish the following cases:

- $t_1\sigma \xrightarrow{c?v} t_1'$, $\omega = c?v$ and $t' = t_1'; t_2\sigma$, derived with R_9. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\forall v', v =_\beta v', \exists t_1'' : t_1\sigma' \xrightarrow{c?v'} t_1''$ and $t_1' \sim_\beta t_1''$. Then we can deduce with R_9 that $t_1\sigma'; t_2\sigma' \xrightarrow{c?v'} t_1''; t_2\sigma$. Furthermore, $(t_1'; t_2\sigma, t_1''; t_2\sigma) \in R$ due to the congruence of $\sim_\beta$ with respect to ";" (see above).
- $t_1\sigma \xrightarrow{\alpha} t_1'$, $\alpha \neq c?v$, $\omega = \alpha$ and $t' = t_1'; t_2\sigma$, derived with R_9. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists \alpha', \alpha =_\beta \alpha', \exists t_1'' : t_1\sigma' \xrightarrow{\alpha'} t_1''$ and $t_1' \sim_\beta t_1''$. Then we can deduce with R_9 that $t_1\sigma'; t_2\sigma' \xrightarrow{\alpha'} t_1''; t_2\sigma$. Furthermore, $(t_1'; t_2\sigma, t_1''; t_2\sigma) \in R$.
- $t_1\sigma \xrightarrow{\checkmark} t_1'$, $t_2\sigma \xrightarrow{c?v} t_2'$, $\omega = c?v$ and $t' = t_1'; t_2'$. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists t_1'' : t_1\sigma' \xrightarrow{\checkmark} t_1''$ and $t_1' \sim_\beta t_1''$. With $t_2\sigma \sim_\beta t_2\sigma'$ we know that $\forall v', v =_\beta v', \exists t_2'' : t_2\sigma' \xrightarrow{c?v'} t_2''$ and $t_2' \sim_\beta t_2''$. Then we can deduce with R_{10} that $t_1\sigma'; t_2\sigma' \xrightarrow{c?v'} t_1''; t_2''$. Furthermore, $(t_1'; t_2', t_1''; t_2'') \in R$.
- $t_1\sigma \xrightarrow{\checkmark} t_1'$, $t_2\sigma \xrightarrow{\omega} t_2'$, $\omega \neq c?v$ and $t' = t_1'; t_2'$, derived with R_{10}. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists t_1'' : t_1\sigma' \xrightarrow{\checkmark} t_1''$ and $t_1' \sim_\beta t_1''$. With $t_2\sigma \sim_\beta t_2\sigma'$ we know that $\exists \omega', \omega =_\beta \omega', \exists t_2'' : t_2\sigma' \xrightarrow{\omega'} t_2''$ and $t_2' \sim_\beta t_2''$. Then we can deduce with R_{10} that $t_1\sigma'; t_2\sigma' \xrightarrow{\omega'} t_1''; t_2''$. Furthermore, $(t_1'; t_2', t_1''; t_2'') \in R$.
- $t_1\sigma \xrightarrow{\checkmark} t_1'$, $t_1' \xrightarrow{c?v} t_1''$, $t_2\sigma \xrightarrow{c!v,C} t_2'$, $\omega = \tau$ and $t' = \mathbf{ch}\ C.\ t_1''; t_2'$, derived with R_{11}. With $t_2\sigma \sim_\beta t_2\sigma'$ we know that $\exists v', v =_\beta v', \exists t_2'' : t_2\sigma' \xrightarrow{c!v',C} t_2''$ and $t_2' \sim_\beta t_2''$. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists t_3' : t_1\sigma' \xrightarrow{\checkmark} t_3'$, $t_1' \sim_\beta t_3'$ and $\forall v'', v =_\beta v'', \exists t_3'' : t_3' \xrightarrow{c?v''} t_3''$, $t_1'' \sim_\beta t_3''$. Then we know with $v' =_\beta v''$ that $\exists t_v : t_3' \xrightarrow{c?v'} t_v$ and $t_1'' \sim_\beta t_v$. Therefore, we can deduce with R_{11} that $t_1\sigma'; t_2\sigma' \xrightarrow{\tau} \mathbf{ch}\ C.\ t_v; t_2''$. Furthermore, $(\mathbf{ch}\ C.\ t_1''; t_2', \mathbf{ch}\ C.\ t_v; t_2'') \in R$.
- $t_1\sigma \xrightarrow{\checkmark} t_1'$, $t_1' \xrightarrow{c!v,C} t_1''$, $t_2\sigma \xrightarrow{c?v} t_2'$, $\omega = \tau$ and $t' = \mathbf{ch}\ C.\ t_1''; t_2'$, derived with R_{11}. With $t_1\sigma \sim_\beta t_1\sigma'$ we know that $\exists t_3' : t_1\sigma' \xrightarrow{\checkmark} t_3'$, $t_1' \sim_\beta t_3'$ and $\exists v', v =_\beta v', \exists t_3'' : t_3' \xrightarrow{c!v',C} t_3''$, $t_1'' \sim_\beta t_3''$. With $t_2\sigma \sim_\beta t_2\sigma'$ we know that $\forall v'', v =_\beta v'', \exists t_2'' : t_2\sigma' \xrightarrow{c?v''} t_2''$ and $t_2' \sim_\beta t_2''$. Then we know with $v' =_\beta v''$ that $\exists t_v : t_2\sigma' \xrightarrow{c?v'} t_v$ and $t_2' \sim_\beta t_v$. Therefore, we can deduce with R_{11} that $t_1\sigma'; t_2\sigma' \xrightarrow{\tau} \mathbf{ch}\ C.\ t_3''; t_v$. Furthermore, $(\mathbf{ch}\ C.\ t_1''; t_2', \mathbf{ch}\ C.\ t_3''; t_v) \in R$. □

Proof. (Theorem 6, continued) Now we can prove the theorem for the remaining operaters in TERM.

– $\{\mathbf{x} \leftarrow \mathbf{E}\}; t \sim_\beta \{\mathbf{x} \leftarrow \mathbf{E}'\}; u$ if $t \sim_\beta u$ and $\mathbf{E} =_\beta \mathbf{E}'$.

Let $R = \{(\{\mathbf{x} \leftarrow \mathbf{E}\}; t_0, \{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0) \mid t_0 \sim_\beta u_0, \mathbf{E} =_\beta \mathbf{E}'\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetric, we need to show only one direction.

We assume $\{\mathbf{x} \leftarrow \mathbf{E}\}; t_0 \xrightarrow{\omega} t'$ and distinguish the following cases:

- $[\![\mathbf{E}]\!] = \mathbf{v}$, $t_0\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{c?v} t'_0$ and $t' = t'_0$, derived with R_6. Let $\mathbf{v}' = [\![\mathbf{E}']\!]$. With $t_0 \sim_\beta u_0$ and Lemma 7 we can deduce that $t_0\langle \mathbf{v}/\mathbf{x}\rangle \sim_\beta u_0\langle \mathbf{v}'/\mathbf{x}\rangle$. Then we know that $\forall v', v' =_\beta v, \exists u'_0 : u_0\langle \mathbf{v}'/\mathbf{x}\rangle \xrightarrow{c?v'} t'_0$ and $t'_0 \sim_\beta u'_0$. Therefore, we can derive with R_6 that $\{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0 \xrightarrow{c?v'} u'_0$. Furthermore, $(t'_0, u'_0) \in R$.
- $[\![\mathbf{E}]\!] = \mathbf{v}$, $t_0\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{\omega} t'_0$, $\omega \neq c?v$ and $t' = t'_0$, derived with R_6. Let $\mathbf{v}' = [\![\mathbf{E}']\!]$. With $t_0 \sim_\beta u_0$ and Lemma 7 we can deduce that $t_0\langle \mathbf{v}/\mathbf{x}\rangle \sim_\beta u_0\langle \mathbf{v}'/\mathbf{x}\rangle$. Then we know that $\exists \omega', \omega =_\beta \omega', \exists u'_0 : u_0\langle \mathbf{v}'/\mathbf{x}\rangle \xrightarrow{\omega'} u'_0$ and $t'_0 \sim_\beta u'_0$. Therefore, we can derive with R_6 that $\{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0 \xrightarrow{\omega'} u'_0$. Furthermore, $(t'_0, u'_0) \in R$.

– $E?x; t \sim_\beta E'?x; u$ if $t \sim_\beta u$ and $E =_\beta E'$.
Let $R = \{(E?x; t_0, E'?x; u_0) \mid t_0 \sim_\beta u_0, E =_\beta E'\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Since R is symmetric, we need to show only one direction.
We assume $E?x; t_0 \xrightarrow{\omega} t'$. For channel values beta-equivalence is identity; therefore, $\exists c : [\![E]\!] = c = [\![E']\!]$. Then we know that $\omega = c?v$ and $t' = t_0\langle v/x\rangle$, derived with R_5. Let $v' =_\beta v$ be arbitrarly. Then we can derive with R_5 that $E'?x; u_0 \xrightarrow{c?v'} u_0\langle v'/x\rangle$. With $t_0 \sim_\beta u_0$ we know that $\forall \sigma : t_0\sigma \sim_\beta u_0\sigma$. Therefore, we can deduce with $v =_\beta v'$ and Lemma 7 that $t_0\langle v/x\rangle \sim_\beta u_0\langle v'/x\rangle$. Furthermore, $(t_0\langle v/x\rangle, u_0\langle v'/x\rangle) \in R$. □

Theorem 8. The axiomatic theory is sound with respect to $\sim_\beta$.

Proof. For each axiom of Figure 4 we define a relation and prove that it satisfies the bisimulation conditions of Definition 5. For each step $t \xrightarrow{c!v,C} t'$ we assume that $C \cap fc(t) = \emptyset$.

– Axiom (1): $[false] = \mathbf{0}$.
Let $R = \{([false], \mathbf{0})\}$ be a relation. With R_3 we know that $[false] \not\xrightarrow{}$. Furthermore, $\mathbf{0} \not\xrightarrow{}$. Therefore, R satisfies the bisimulation conditions of Definition 5.
– Axiom (2): $[true] = \tau$.
Let $R = \{([true], \tau)\} \cup \{(t, t) \mid t \in \textsc{Term}\}$ be a relation. With R_3 we know that $[true] \xrightarrow{\tau} \mathbf{1}$. Furthermore, with R_2 we know that $\tau \xrightarrow{\tau} \mathbf{1}$. Therefore, $(\mathbf{1}, \mathbf{1}) \in R$ and R satisfies the bisimulation conditions of Definition 5.
– Axiom (3): $\mathbf{1}; t = t$.
Let $R = \{(\mathbf{1}; u, u) \mid u \in \textsc{Term}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. We distinguish the following cases:
  - Assume $\mathbf{1}; u \xrightarrow{\omega} u_{new}$. Then we know with R_1, R_{10} that $u \xrightarrow{\omega} u'$ and $u_{new} = \mathbf{1}; u'$. Furthermore, $(\mathbf{1}; u', u') \in R$.
  - Assume $u \xrightarrow{\omega} u'$. Then we can deduce with R_1, R_{10} that $\mathbf{1}; u \xrightarrow{\omega} \mathbf{1}; u'$. Furthermore, $(\mathbf{1}; u', u') \in R$.
– Axiom (4): $t; \mathbf{1} = t$.
Let $R = \{(u; \mathbf{1}, u) \mid u \in \textsc{Term}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. We distinguish the following cases:
  - We assume $u; \mathbf{1} \xrightarrow{\omega} u_{new}$ ans distinguish the following cases:
    * $u \xrightarrow{\alpha} u'$, $\omega = \alpha$ and $u_{new} = u'; \mathbf{1}$, derived with R_9. Furthermore, $(u'; \mathbf{1}, u') \in R$.
    * $u \xrightarrow{\checkmark} u'$, $\omega = \checkmark$ and $u_{new} = u'; \mathbf{1}$, derived with R_1, R_{10}. Furthermore, $(u'; \mathbf{1}, u') \in R$.
  - We assume $u \xrightarrow{\omega} u'$ and distinguish the following cases:
    * $\omega = \alpha$. Then $u; \mathbf{1} \xrightarrow{\alpha} u'; \mathbf{1}$, derived with R_9. Furthermore, $(u'; \mathbf{1}, u') \in R$.
    * $\omega = \checkmark$. Then $u; \mathbf{1} \xrightarrow{\checkmark} u'; \mathbf{1}$, derived with R_1, R_{10}. Furthermore, $(u'; \mathbf{1}, u') \in R$.
– Axiom (5): $\mathbf{0}; t = \mathbf{0}$.
Let $R = \{(\mathbf{0}; u, \mathbf{0}) \mid u \in \textsc{Term}\}$ be a relation. With $\mathbf{0} \not\xrightarrow{}$ and R_9, R_{10} we know that $\mathbf{0}; u \not\xrightarrow{}$. Therefore, R satisfies the bisimulation conditions of Definition 5.

– Axiom (6): $t_1;(t_2;t_3) = (t_1;t_2);t_3$. Let

$$R_1 = \{(u_1;(u_2;u_3),(u_1;u_2);u_3) \mid u_1,u_2,u_3 \in \text{TERM}\}$$
$$R_2 = \{(\mathbf{ch}\ C.\ u_1;(u_2;u_3),(\mathbf{ch}\ C.\ u_1;u_2);u_3) \mid u_1,u_2,u_3 \in \text{TERM}, C \cap fc(u_3) = \emptyset\}$$
$$R_3 = \{(u_1;(\mathbf{ch}\ C.\ u_2;u_3),\mathbf{ch}\ C.\ (u_1;u_2);u_3) \mid u_1,u_2,u_3 \in \text{TERM}, C \cap fc(u_1) = \emptyset\}$$
$$R_4 = \{(\mathbf{ch}\ C.\ u_1;(u_2;u_3),\mathbf{ch}\ C.\ (u_1;u_2);u_3) \mid u_1,u_2,u_3 \in \text{TERM}\}$$

Let $R = R_1 \cup R_2 \cup R_3 \cup R_4$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. First, we consider R_1:

- We assume $u_1;(u_2;u_3) \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
  - $u_1 \xrightarrow{\alpha} u_1'$, $\omega = \alpha$ and $u_{new} = u_1';(u_2;u_3)$, derived with R_9. Then we can deduce with R_9 that $(u_1;u_2);u_3 \xrightarrow{\alpha} (u_1';u_2);u_3$. Furthermore, it is the case that $(u_1';(u_2;u_3),(u_1';u_2);u_3) \in R$.
  - $u_1 \xrightarrow{\checkmark} u_1'$, $u_2 \xrightarrow{\alpha} u_2'$, $\omega = \alpha$ and $u_{new} = u_1';(u_2';u_3)$, derived with R_9, R_{10}. Then we can derive with R_{10}, R_9 that $(u_1;u_2);u_3 \xrightarrow{\alpha} (u_1';u_2');u_3$. Furthermore, $(u_1';(u_2';u_3),(u_1';u_2');u_3) \in R$.
  - $u_1 \xrightarrow{\checkmark} u_1'$, $u_1' \xrightarrow{\alpha} u_1''$, $u_2 \xrightarrow{\alpha'} u_2'$, $\{\alpha,\alpha'\} = \{c?v,(c!v,C)\}$, $\omega = \tau$ and $u_{new} = \mathbf{ch}\ C.\ u_1'';(u_2';u_3)$, derived with R_9, R_{11}. Then we can derive with R_{11}, R_9 that $(u_1;u_2);u_3 \xrightarrow{\tau} (\mathbf{ch}\ C.u_1'';u_2);u_3$. Furthermore, with C restricted in u_1 or u_2 and $C \cap fc(u_1;(u_2;u_3)) = C \cap fc((u_1;u_2);u_3) = \emptyset$ we know that $fc(u_3) \cap C = \emptyset$. Therefore, $(\mathbf{ch}\ C.\ u_1'';(u_2';u_3),(\mathbf{ch}\ C.\ u_1'';u_2);u_3) \in R$.
  - $u_1 \xrightarrow{\checkmark} u_1'$, $u_2 \xrightarrow{\checkmark} u_2'$, $u_3 \xrightarrow{\omega} u_3'$ and $u_{new} = u_1';(u_2';u_3')$, derived with R_{10}. Then we can deduce with R_{10} that $(u_1;u_2);u_3 \xrightarrow{\omega} (u_1';u_2');u_3'$. Furthermore, $(u_1';(u_2';u_3'),(u_1';u_2');u_3') \in R$.
  - $u_1 \xrightarrow{\checkmark} u_1'$, $u_1' \xrightarrow{\alpha} u_1''$, $u_2 \xrightarrow{\checkmark} u_2'$, $u_3 \xrightarrow{\alpha'} u_3$, $\{\alpha,\alpha'\} = \{c?v,(c!v,C)\}$, $\omega = \tau$ and $u_{new} = \mathbf{ch}\ C.\ u_1'';(u_2';u_3')$, derived with R_9, R_{10}, R_{11}. Then we can derive with R_9 , R_{10} and R_{11} that $(u_1;u_2);u_3 \xrightarrow{\tau} \mathbf{ch}\ C.\ (u_1'';u_2');u_3$. Furthermore, $(\mathbf{ch}\ C.\ u_1'';(u_2';u_3'),\mathbf{ch}\ C.\ (u_1'';u_2');u_3) \in R$.
  - $u_1 \xrightarrow{\checkmark} u_1'$, $u_2 \xrightarrow{\checkmark} u_2'$, $u_2 \xrightarrow{\alpha} u_2''$, $u_3 \xrightarrow{\alpha'} u_3'$, $\{\alpha,\alpha'\} = \{c?v,(c!v,C)\}$, $\omega = \tau$ and $u_{new} = u_1';(\mathbf{ch}\ C.\ u_2'';u_3')$, derived with R_9, R_{10}, R_{11}. Then we can derive with R_{10}, R_9, R_{11} that $(u_1;u_2);u_3 \xrightarrow{\tau} \mathbf{ch}\ C.\ (u_1';u_2'');u_3'$. Furthermore, with C restricted in u_2 or u_3 and $fc(u_1;(u_2;u_3)) \cap C = fc((u_1;u_2);u_3) \cap C = \emptyset$ we know that $fc(u_1) \cap C = \emptyset$. Therefore, $(u_1';(\mathbf{ch}\ C.\ u_2'';u_3'),\mathbf{ch}\ C.\ (u_1';u_2'');u_3') \in R$.
- For the assumption $(u_1;u_2);u_3 \xrightarrow{\omega} u_{new}$ analogous.

The theorem can be proved analogously for R_4 in combination with the techniques used for the congruence proof of channel restriction in Theorem 6. For R_2 we know that $fc(u_3) \cap C = \emptyset$; therefore, the restriction of C does not affect the behaviour of u_3. Then we can apply the techniques for the congruence proof of channel restriction. The same holds for u_1 in R_3.

– Axiom (7): $\{\mathbf{x} \leftarrow \mathbf{v}\};t = t\langle \mathbf{v}/\mathbf{x}\rangle$.
Let $R = \{(\{\mathbf{x} \leftarrow \mathbf{v}\};u, u\langle \mathbf{v}/\mathbf{x}\rangle) \mid u \in \text{TERM}\} \cup \{(u,u) \mid u \in \text{TERM}\}$ be a relation. We distinguish the following cases:
- Assume that $\{\mathbf{x} \leftarrow \mathbf{v}\};u \xrightarrow{\omega} u'$. Then we know with R_6 that $u\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{\omega} u'$. Furthermore, $(u',u') \in R$.
- Assume $u\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{\omega} u'$. Then we can derive with R_6 that $\{\mathbf{x} \leftarrow \mathbf{v}\};u \xrightarrow{\omega} u'$. Furthermore, $(u',u') \in R$.

– Axiom (8): $t + \mathbf{0} = t$.
Let $R = \{(u+\mathbf{0},u) \mid u \in \text{TERM}\} \cup \{(u,u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
- Assume $u + \mathbf{0} \xrightarrow{\omega} u'$. With $\mathbf{0} \not\xrightarrow{}$ and R_7 we can deduce that $u \xrightarrow{\omega} u'$. Furthermore, $(u',u') \in R$.
- Assume $u \xrightarrow{\omega} u'$. Then we can derive with R_7 that $u + \mathbf{0} \xrightarrow{\omega} u'$. Furthermore, $(u',u') \in R$.

– Axiom (9): $t + u = u + t$.
Let $R = \{(t_0 + u_0, u_0 + t_0) \mid t_0, u_0 \in \text{TERM}\} \cup \{(t_0, t_0) \mid t_0 \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $t_0 + u_0 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
    * $t_0 \xrightarrow{\omega} t_0'$ and $t_{new} = t_0'$, derived with R_7. Then we can deduce with R_8 that $u_0 + t_0 \xrightarrow{\omega} t_0'$. Furthermore, $(t_0', t_0') \in R$.
    * $u_0 \xrightarrow{\omega} u_0'$ and $t_{new} = u_0'$, derived with R_8. Then we can deduce with R_7 that $u_0 + t_0 \xrightarrow{\omega} u_0'$. Furthermore, $(u_0', u_0') \in R$.
  - For the assumption $u_0 + t_0 \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (10): $t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3$.
Let $R = \{(u_1 + (u_2 + u_3), (u_1 + u_2) + u_3) \mid u_1, u_2, u_3 \in \text{TERM}\} \cup \{(u, u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions.
  - We assume $u_1 + (u_2 + u_3) \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
    * $u_1 \xrightarrow{\omega} u_1'$ and $u_{new} = u_1'$, derived with R_7. Then we can deduce with R_7 that $(u_1 + u_2) + u_3 \xrightarrow{\omega} u_1'$. Furthermore, $(u_1', u_1') \in R$.
    * $u_2 \xrightarrow{\omega} u_2'$ and $u_{new} = u_2'$, derived with R_7, R_8. Then we can derive with R_8, R_7 that $(u_1 + u_2) + u_3 \xrightarrow{\omega} u_2'$. Furthermore, $(u_2', u_2') \in R$.
    * $u_3 \xrightarrow{\omega} u_3'$ and $u_{new} = u_3'$, derived with R_8. Then we can deduce with R_8 that $(u_1 + u_2) + u_3 \xrightarrow{\omega} u_3'$. Furthermore, $(u_3', u_3') \in R$.
  - For the assumption $(u_1 + u_2) + u_3 \xrightarrow{\omega} u_{new}$ analogous.

– Axiom (11): $t + t = t$.
Let $R = \{(u + u, u) \mid u \in \text{TERM}\} \cup \{(u, u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - Assume $u + u \xrightarrow{\omega} u'$. Then we know with R_7 or R_8 that $u \xrightarrow{\omega} u'$. Furthermore, $(u', u') \in R$.
  - Assume $u \xrightarrow{\omega} u'$. Then we can deduce with R_7 or R_8 that $u + u \xrightarrow{\omega} u'$. Furthermore, $(u', u') \in R$.

– Axiom (12): $(t_1 + t_2); t_3 = t_1; t_3 + t_2; t_3$.
Let $R = \{((u_1 + u_2); u_3, u_1; u_3 + u_2; u_3) \mid u_1, u_2, u_3 \in \text{TERM}\} \cup \{(u, u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $(u_1 + u_2); u_3 \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
    * $u_1 \xrightarrow{\alpha} u_1'$, $\omega = \alpha$ and $u_{new} = u_1'; u_3$, derived with R_7, R_9. Then we can derive with R_9, R_7 that $u_1; u_3 + u_2; u_3 \xrightarrow{\alpha} u_1'; u_3$. Furthermore, $(u_1'; u_3, u_1'; u_3) \in R$.
    * $u_1 \xrightarrow{\checkmark} u_1'$, $u_3 \xrightarrow{\omega} u_3'$ and $u_{new} = u_1'; u_3'$, derived with R_7, R_{10}. Then we can derive with R_{10}, R_7 that $u_1; u_3 + u_2; u_3 \xrightarrow{\omega} u_1'; u_3'$. Furthermore, $(u_1'; u_3', u_1'; u_3') \in R$.
    * $u_1 \xrightarrow{\checkmark} u_1'$, $u_1' \xrightarrow{\alpha} u_1''$, $u_3 \xrightarrow{\alpha'} u_3'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $u_{new} = \mathbf{ch}\ C.\ u_1''; u_3'$, derived with R_7, R_{11}. Then we can deduce with R_{11}, R_7 that $u_1; u_3 + u_2; u_3 \xrightarrow{\tau} \mathbf{ch}\ C.\ u_1''; u_3'$. Furthermore, $(\mathbf{ch}\ C.\ u_1''; u_3', \mathbf{ch}\ C.\ u_1''; u_3') \in R$.
    * $u_2 \xrightarrow{\alpha} u_2'$, $\omega = \alpha$ and $u_{new} = u_2'; u_3$, derived with R_8, R_9. Then we can derive with R_9, R_8 that $u_1; u_3 + u_2; u_3 \xrightarrow{\alpha} u_2'; u_3$. Furthermore, $(u_2'; u_3, u_2'; u_3) \in R$.
    * $u_2 \xrightarrow{\checkmark} u_2'$, $u_3 \xrightarrow{\omega} u_3'$ and $u_{new} = u_2'; u_3'$, derived with R_8, R_{10}. Then we can derive with R_{10}, R_8 that $u_1; u_3 + u_2; u_3 \xrightarrow{\omega} u_2'; u_3'$. Furthermore, $(u_2'; u_3', u_2'; u_3') \in R$.
    * $u_2 \xrightarrow{\checkmark} u_2'$, $u_2' \xrightarrow{\alpha} u_2''$, $u_3 \xrightarrow{\alpha'} u_3'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $u_{new} = \mathbf{ch}\ C.\ u_2''; u_3'$, derived with R_8, R_{11}. Then we can deduce with R_{11}, R_8 that $u_1; u_3 + u_2; u_3 \xrightarrow{\tau} \mathbf{ch}\ C.\ u_2''; u_3'$. Furthermore, $(\mathbf{ch}\ C.\ u_2''; u_3', \mathbf{ch}\ C.\ u_2''; u_3') \in R$.
  - For the assumption $u_1; u_3 + u_2; u_3$ analogous.

– Axiom (13): $\mathbf{ch}\ C.\ \mathbf{0} = \mathbf{0}$.
Let $R = \{(\mathbf{ch}\ C.\ \mathbf{0}, \mathbf{0})\}$ be a relation. With $\mathbf{0} \not\xrightarrow{}$ and R_{12}, R_{13} we know that $\mathbf{ch}\ C.\mathbf{0} \not\xrightarrow{}$. Therefore, R satisfies the bisimulation conditions of Definition 5.

– Axiom (14): $\mathbf{ch}\ C.\ \mathbf{1} = \mathbf{1}$.
Let $R = \{(\mathbf{ch}\ C.\ \mathbf{1}, \mathbf{1})\}$ be a relation. With R_1 we know that $\mathbf{1} \xrightarrow{\checkmark} \mathbf{1}$. Then we can deduce with R_{12} that $\mathbf{ch}\ C.\ \mathbf{1} \xrightarrow{\checkmark} \mathbf{ch}\ C.\ \mathbf{1}$. Furthermore, $(\mathbf{ch}\ C.\ \mathbf{1}, \mathbf{1}) \in R$ and the bisimulation conditions of Definition 5 are satisfied.

– Axiom (15): $\mathbf{ch}\ C.\ spawn(t) = spawn(\mathbf{ch}\ C.\ t)$.
Let $R = \{(\mathbf{ch}\ D.\ spawn(u), spawn(\mathbf{ch}\ D.\ u)) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $\mathbf{ch}\ C.\ spawn(u) \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
    * $u \xrightarrow{\alpha} u'$, $fc(\alpha) \cap C = \emptyset$, $\omega = \alpha$ and $u_{new} = \mathbf{ch}\ C.\ spawn(u')$, derived with R_{15}, R_{12}. Then we can deduce with R_{12}, R_{15} that $spawn(\mathbf{ch}\ C.u) \xrightarrow{\alpha} spawn(\mathbf{ch}\ C.u')$. Furthermore, $(\mathbf{ch}\ C.\ spawn(u'), spawn(\mathbf{ch}\ C.\ u')) \in R$.
    * $u \xrightarrow{c!v,A} u'$, $c \notin C$, $A \cap C = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = A \cup (C \cap fc(\omega))$ and $u_{new} = \mathbf{ch}\ C \setminus fc(v).\ spawn(u')$, derived with R_{15}, R_{13}. Then we can deduce with R_{13}, R_{15} that $spawn(\mathbf{ch}\ C.\ u) \xrightarrow{c!v,A\cup(C\cap fc(v))} spawn(\mathbf{ch}\ C \setminus fc(v).\ u')$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ spawn(u'), spawn(\mathbf{ch}\ C \setminus fc(v).\ u')) \in R$.
    * $spawn(u) \xrightarrow{\checkmark} spawn(u)$, $\omega = \checkmark$ and $u_{new} = \mathbf{ch}\ C.\ spawn(u)$, derived with R_{14}, R_{12}. Then we can derive with R_{14} that $spawn(\mathbf{ch}\ C.\ u) \xrightarrow{\checkmark} spawn(\mathbf{ch}\ C.\ u)$. Furthermore, $(\mathbf{ch}\ C.\ spawn(u), spawn(\mathbf{ch}\ C.\ u)) \in R$.
  - For the assumption $spawn(\mathbf{ch}\ C.\ u) \xrightarrow{\omega} u_{new}$ analogous.

– Axiom (16): $\mathbf{ch}\ C.\ t = t$ if $C \cap fc(t) = \emptyset$.
Let $R = \{(\mathbf{ch}\ C.\ u, u) \mid u \in \text{TERM}, C \cap fc(u) = \emptyset\}$ be a relation. We assume $u \xrightarrow{\omega} u'$ and $C \cap fc(u) = \emptyset$. Then we can deduce with R_7 that $\mathbf{ch}\ C.\ u \xrightarrow{\omega} \mathbf{ch}\ C.\ u'$. Furthermore, $(\mathbf{ch}\ C.\ u', u') \in R$ and the bisimulation conditions of Definition 5 are satisfied.

– Axiom (17): $\mathbf{ch}\ C.\ \mathbf{ch}\ C'.\ t = \mathbf{ch}\ C'.\ \mathbf{ch}\ C.\ t$.
Let $R = \{(\mathbf{ch}\ D.\ \mathbf{ch}\ D'.\ u, \mathbf{ch}\ D'.\ \mathbf{ch}\ D.\ u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. We assume $C \cap C' = \emptyset$; otherwise, we apply alpha-conversion.
  - We assume $\mathbf{ch}\ C.\ \mathbf{ch}\ C'.\ u \xrightarrow{\omega} u_{new}$ and distinguish the following cases.
    * $u \xrightarrow{\omega} u'$, $fc(\omega) \cap (C \cup C') = \emptyset$ and $u_{new} = \mathbf{ch}\ C.\mathbf{ch}\ C'.u'$, derived with R_{12}. Then we can deduce with R_{12} that $\mathbf{ch}\ C'.\ \mathbf{ch}\ C.\ u \xrightarrow{\omega} \mathbf{ch}\ C'.\ \mathbf{ch}\ C.\ u'$. Furthermore, $(\mathbf{ch}\ C.\ \mathbf{ch}\ C'.\ u', \mathbf{ch}\ C'.\ \mathbf{ch}\ C.\ u') \in R$.
    * $u \xrightarrow{c!v,A} u'$, $c \notin (C \cup C')$, $A \cap (C \cup C') = \emptyset$, $fc(v) \cap (C \cup C') \neq \emptyset$, $\omega = (c!v, A \cup ((C \cup C') \cap fc(v)))$ and $u_{new} = \mathbf{ch}\ C \setminus fc(v).\ \mathbf{ch}\ C' \setminus fc(v).\ u'$, derived with R_{13}. Then we can deduce with R_{13} that $\mathbf{ch}\ C'.\mathbf{ch}\ C.u \xrightarrow{c!v,A\cup((C\cup C')\cap fc(v))} \mathbf{ch}\ C' \setminus fc(v).\mathbf{ch}\ C \setminus fc(v).u'$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\mathbf{ch}\ C' \setminus fc(v).u', \mathbf{ch}\ C' \setminus fc(v).\ \mathbf{ch}\ C \setminus fc(v).\ u') \in R$. (Note that $fc(v) \cap C \neq \emptyset \neq fc(v) \cap C'$ is possible.)
  - For the assumption $\mathbf{ch}\ C'.\ \mathbf{ch}\ C.\ u \xrightarrow{\omega} u_{new}$ analogous.

– Axiom (18): $\mathbf{ch}\ C.\ t + u = \mathbf{ch}\ C.\ t + \mathbf{ch}\ C.\ u$.
Let $R = \{(\mathbf{ch}\ D.\ t_0 + u_0, \mathbf{ch}\ D.\ t_0 + \mathbf{ch}\ D.\ u_0) \mid t_0, u_0 \in \text{TERM}\} \cup \{(t_0, t_0) \mid t_0 \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $\mathbf{ch}\ C.\ t_0 + u_0 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
    * $t_0 \xrightarrow{\omega} t_0'$, $fc(\omega) \cap C = \emptyset$ and $t_{new} = \mathbf{ch}\ C.\ t_0'$, derived with R_7, R_{12}. Then we can deduce with R_{12}, R_7 that $\mathbf{ch}\ C.\ t_0 + \mathbf{ch}\ C.\ u_0 \xrightarrow{\omega} \mathbf{ch}\ C.\ t_0'$. Furthermore, $(\mathbf{ch}\ C.\ t_0', \mathbf{ch}\ C.\ t_0') \in R$.
    * $u_0 \xrightarrow{\omega} u_0'$, $fc(\omega) \cap C = \emptyset$ and $t_{new} = \mathbf{ch}\ C.\ u_0'$, derived with R_8, R_{12}. Then we can derive with R_{12}, R_8 that $\mathbf{ch}\ C.\ t_0 + \mathbf{ch}\ C.\ u_0 \xrightarrow{\omega} \mathbf{ch}\ C.\ u_0'$. Furthermore, $(\mathbf{ch}\ C.\ u_0', \mathbf{ch}\ C.\ u_0') \in R$.
    * $t_0 \xrightarrow{c!v,A} t_0'$, $A \cap C = \emptyset$, $c \notin C$, $fc(v) \cap C \neq \emptyset$, $\omega = c!v, A \cup (C \cap fc(v))$ and $t_{new} = \mathbf{ch}\ C \setminus fc(v).\ t_0'$, derived with R_7, R_{13}. Then we can derive with R_{13}, R_7 that $\mathbf{ch}\ C.\ t_0 + \mathbf{ch}\ C.\ u_0 \xrightarrow{c!v,A\cup(C\cap fc(v))} \mathbf{ch}\ C \setminus fc(v).\ t_0'$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ t_0', \mathbf{ch}\ C \setminus fc(v).\ t_0') \in R$.
    * $u_0 \xrightarrow{c!v,A} u_0'$, $A \cap C = \emptyset$, $c \notin C$, $fc(v) \cap C \neq \emptyset$, $\omega = c!v, A \cup (C \cap fc(v))$ and $t_{new} = \mathbf{ch}\ C \setminus fc(v).\ u_0'$, derived with R_8, R_{13}. Then we can derive with R_{13}, R_8 that $\mathbf{ch}\ C.\ t_0 + \mathbf{ch}\ C.\ u_0 \xrightarrow{c!v,A\cup(C\cap fc(v))} \mathbf{ch}\ C \setminus fc(v).\ u_0'$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ u_0', \mathbf{ch}\ C \setminus fc(v).\ u_0') \in R$.
  - For the assumption $\mathbf{ch}\ C.\ t_0 + \mathbf{ch}\ C.\ u_0 \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (19): $\mathbf{ch}\ C.\ c?x; t = \mathbf{0}$ if $c \in C$.
Let $R = \{(\mathbf{ch}\ C.\ c?x; u, \mathbf{0}) \mid u \in \text{TERM}\}$ be a relation. We know that $\mathbf{0} \not\xrightarrow{}$. Furthermore, we can deduce with $c \in C$ and R_5, R_{12}, R_{13} that $\mathbf{ch}\ C.\ c?x; u \not\xrightarrow{}$. Therefore, R satisfies the bisimulation conditions of Definition 5.
– Axiom (20): $\mathbf{ch}\ C.\ c!v; t = \mathbf{0}$ if $c \in C$.
Let $R = \{(\mathbf{ch}\ C.\ c!v; u, \mathbf{0}) \mid u \in \text{TERM}, c \in C\}$ be a relation. We know that $\mathbf{0} \not\xrightarrow{}$. Furthermore, we know with R_4, R_{12}, R_{13} that $\mathbf{ch}\ C.\ c!v; u \not\xrightarrow{}$. Therefore, R satisfies the bisimulation conditions of Definition 5.
– Axiom (21): $\mathbf{ch}\ C.\ \gamma; t = \gamma; \mathbf{ch}\ C.\ t$ if $C \cap fc(\gamma) = \emptyset$.
Let $R = \{(\mathbf{ch}\ C.\ \gamma; u, \gamma; \mathbf{ch}\ C.\ u) \mid u \in \text{TERM}, C \cap fc(\gamma) = \emptyset\} \cup$
$\{(\mathbf{ch}\ C.\ \mathbf{1}; u, \mathbf{1}; \mathbf{ch}\ C.\ u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. With R_9, R_{12} and $C \cap fc(\gamma) = \emptyset$ we can deduce that $\mathbf{ch}\ C.\gamma; u \xrightarrow{\gamma} \mathbf{ch}\ C.\mathbf{1}; u$. Similarly, we can derive with R_9 that $\gamma; \mathbf{ch}\ C.u \xrightarrow{\gamma} \mathbf{1}; \mathbf{ch}\ C.u$. If $\gamma = [E]$ and $[\![E]\!] = true$, we can deduce with $fc(\gamma) \cap C = \emptyset$, R_3, R_9, R_{12} that $\mathbf{ch}\ C.\gamma; u \xrightarrow{\tau} \mathbf{ch}\ C.\ \mathbf{1}; u$ and $\gamma; \mathbf{ch}\ C.\ u \xrightarrow{\tau} \mathbf{1}; \mathbf{ch}\ C.\ u$. Furthermore, $(\mathbf{ch}\ C.\ \mathbf{1}; u, \mathbf{1}; \mathbf{ch}\ C.\ u) \in R$. The proof for $\{(\mathbf{ch}\ C.\ \mathbf{1}; u, \mathbf{1}; \mathbf{ch}\ C.\ u) \mid u \in \text{TERM}\}$ is similar to proof for axiom (3).
– Axiom (22): $\mathbf{ch}\ C.\ c?x; t = c?x; \mathbf{ch}\ C.\ t$ if $c \notin C$.
Let $R = \{(\mathbf{ch}\ C.\ c?x; u, c?x; \mathbf{ch}\ C.\ u) \mid u \in \text{TERM}, c \notin C\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $\mathbf{ch}\ C.\ c?x; u \xrightarrow{\omega} u_{new}$. Then we have $\omega = c?v$, $fc(v) \cap C = \emptyset$ and $u_{new} = \mathbf{ch}\ C.\ u\langle v/x\rangle$, derived with R_5, R_{12}. Let $v' =_\beta v$ be arbitrarly. Then we can derive with R_5 that $c?x; \mathbf{ch}\ C.\ u \xrightarrow{c?v'} (\mathbf{ch}\ C.\ u)\langle v'/x\rangle$. With $v' =_\beta v$ we know that $fc(v') = fc(v)$. Therefore, we have $fc(v') \cap C = \emptyset$ and $(\mathbf{ch}\ C.u)\langle v'/x\rangle = \mathbf{ch}\ C.u\langle v'/x\rangle$. With Lemma 7 and Theorem 6 we can deduce that $\mathbf{ch}\ C.\ u\langle v/x\rangle \sim_\beta \mathbf{ch}\ C.\ u\langle v'/x\rangle$. Therefore, $(\mathbf{ch}\ C.\ u\langle v/x\rangle, \mathbf{ch}\ C.\ u\langle v'/x\rangle) \in R$.
  - We assume $c?x; \mathbf{ch}\ C.\ u \xrightarrow{\omega} u_{new}$. Then we have $\omega = c?v$ and $u_{new} = (\mathbf{ch}\ C.u)\langle v/x\rangle$, derived with R_5. We assume $fc(v) \cap C = \emptyset$; otherwise, we apply alpha-conversion. Then we can deduce that $(\mathbf{ch}\ C.\ u)\langle v/x\rangle = \mathbf{ch}\ C.\ u\langle v/x\rangle$. Let $v' =_\beta v$ be arbitrarly. Then we know that $fc(v) = fc(v')$. Therefore, we have $fc(v') \cap C = \emptyset$ and we can deduce with R_5, R_{12} that $\mathbf{ch}\ C.\ c?x; u \xrightarrow{c?v'} \mathbf{ch}\ C.\ u\langle v'/x\rangle$. Then we can derive with Lemma 7 and Theorem 6 that $\mathbf{ch}\ C.\ u\langle v/x\rangle \sim_\beta \mathbf{ch}\ C.\ u\langle v'/x\rangle$. Therefore, $(\mathbf{ch}\ C.\ u\langle v/x\rangle, \mathbf{ch}\ C.\ u\langle v'/x\rangle) \in R$.
– Axiom (23): $(\mathbf{ch}\ C.\ t); u = \mathbf{ch}\ C.\ t; u$ if $C \cap fc(u) = \emptyset$.
Let $R = \{(\mathbf{ch}\ E.\ (\mathbf{ch}\ D.\ t_0); u_0, \mathbf{ch}\ D \cup E.\ t_0; u_0)\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. First, we assume $E = \emptyset$. For $E \neq \emptyset$ we can apply the proof techniques used for the congruence proof for restriction.
  - We assume $(\mathbf{ch}\ C.\ t_0); u_0 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
    * $t_0 \xrightarrow{\omega} t_0'$, $fc(\omega) \cap C = \emptyset$ and $t_{new} = (\mathbf{ch}\ C.\ t_0'); u_0$, derived with R_{12}, R_9. Then we can deduce with R_9, R_{12} that $\mathbf{ch}\ C.\ t_0; u_0 \xrightarrow{\omega} \mathbf{ch}\ C.\ t_0'; u_0$. Furthermore, $((\mathbf{ch}\ C.\ t_0'); u_0, \mathbf{ch}\ C.\ t_0'; u_0) \in R$.
    * $t_0 \xrightarrow{c!v,A} t_0'$, $c \notin C$, $C \cap A = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = A \cup (C \cap fc(v))$ and $t_{new} = \mathbf{ch}\ (.C \setminus fc(v)t_0'); u_0$, derived with R_{13}, R_9. Then we can deduce with R_9, R_{13} that $\mathbf{ch}\ C.t_0; u_0 \xrightarrow{c!v,A\cup(C\cap fc(v))} \mathbf{ch}\ C \setminus fc(v).\ t_0'; u_0$. Furthermore, $(\mathbf{ch}\ (.C \setminus fc(v)t_0'); u_0, \mathbf{ch}\ C \setminus fc(v).\ t_0'; u_0) \in R$.
    * $t_0 \xrightarrow{\checkmark} t_0'$, $u_0 \xrightarrow{\omega} u_0'$ and $t_{new} = (\mathbf{ch}\ C.\ t_0); u_0'$, derived with R_{12}, R_{10}. Then we can deduce with R_{10}, R_{12} and $C \cap fc(u_0) = \emptyset$ that $\mathbf{ch}\ C.\ t_0; u_0 \xrightarrow{\omega} \mathbf{ch}\ C.\ t_0'; u_0'$. Furthermore, $((\mathbf{ch}\ C.\ t_0); u_0', \mathbf{ch}\ C.\ t_0'; u_0') \in R$.
    * $t_0 \xrightarrow{\checkmark} t_0'$, $t_0' \xrightarrow{\alpha} t_0''$, $u_0 \xrightarrow{\alpha}{}' u_0'$, $fc(\alpha) \cap C = \emptyset$, $\{\alpha, \alpha'\} = \{c?v, (c!v, A)\}$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ A.\ (\mathbf{ch}\ C.\ t_0''); u_0'$, derived with R_{12}, R_{11}. Then we can derive with R_{11}, R_{12} that $\mathbf{ch}\ C.\ t_0, u_0 \xrightarrow{\tau} \mathbf{ch}\ C \cup A.\ t_0''; u_0'$. Furthermore, it is the case that $(\mathbf{ch}\ A.\ (\mathbf{ch}\ C.\ t_0''); u_0', \mathbf{ch}\ C \cup A.\ t_0''; u_0') \in R$.
    * $t_0 \xrightarrow{\checkmark} t_0'$, $t_0' \xrightarrow{c!v,A} t_0''$, $u_0 \xrightarrow{c?v} u_0'$, $c \notin C$, $A \cap C = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ A \cup (C \cap fc(v)).\ (\mathbf{ch}\ C \setminus fc(v).\ t_0''; u_0')$. Then we can derive with R_{11}, R_{12}

and $(C \setminus fc(v)) \cup (A \cup (C \cap fc(v))) = (A \cup C)$ that $\mathbf{ch}\ C.\, t_0; u_0 \xrightarrow{\tau} \mathbf{ch}\ A \cup C.\, t_0''; u_0'$. Furthermore, $(\mathbf{ch}\ A \cup (C \cap fc(v)).\, (\mathbf{ch}\ C \setminus fc(v).\, t_0''); u_0', \mathbf{ch}\ A \cup C.\, t_0''; u_0') \in R$.

- We assume $\mathbf{ch}\ C.\, t_0; u_0 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  - $t_0 \xrightarrow{\omega} t_0'$, $fc(\omega) \cap C = \emptyset$ and $t_{new} = \mathbf{ch}\ C.\, t_0'; u_0$, derived with R_9, R_{12}. Then we can derive with R_{12}, R_9 that $(\mathbf{ch}\ C.\, t_0); u_0 \xrightarrow{\omega} (\mathbf{ch}\ C.\, t_0'); u_0$. Furthermore, $((\mathbf{ch}\ C.\, t_0'); u_0, \mathbf{ch}\ C.\, t_0'; u_0) \in R$.
  - $t_0 \xrightarrow{c!v, A} t_0'$, $c \notin C$, $A \cap C = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = c!v, A \cup (C \cap fc(v))$ and $t_{new} = \mathbf{ch}\ C \setminus fc(v).\, t_0'; u_0$, derived with R_9, R_{13}. Then we can deduce with R_{13}, R_9 that $(\mathbf{ch}\ C.\, t_0); u_0 \xrightarrow{c!v, A \cup (C \cap fc(v))} (\mathbf{ch}\ C \setminus fc(v).\, t_0'); u_0$. Furthermore, $((\mathbf{ch}\ C \setminus fc(v).\, t_0'); u_0, \mathbf{ch}\ C \setminus fc(v).\, t_0'; u_0) \in R$.
  - $t_0 \xrightarrow{\checkmark} t_0'$, $u_0 \xrightarrow{\omega} u_0'$ and $t_{new} = \mathbf{ch}\ C.\, t_0'; u_0'$, derived with R_{10}, R_{12} and $C \cap fc(u) = \emptyset$. Then we can derive with R_{12}, R_{10} that $(\mathbf{ch}\ C.t_0); u_0 \xrightarrow{\omega} (\mathbf{ch}\ C.t_0'); u_0'$. Furthermore, $((\mathbf{ch}\ C.\, t_0'); u_0', \mathbf{ch}\ C.\, t_0'; u_0') \in R$.
  - $t_0 \xrightarrow{\checkmark} t_0'$, $t_0' \xrightarrow{\alpha} t_0''$, $fc(\alpha) \cap C = \emptyset$, $\{\alpha, \alpha'\} = \{c?v, (c!v, A)\}$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ A \cup C.\, t_0''; u_0'$, derived with R_{11}, R_{12}. Then we can deduce with R_{12}, R_{11} that $(\mathbf{ch}\ C.\, t_0); u_0 \xrightarrow{\tau} \mathbf{ch}\ A.\, (\mathbf{ch}\ C.\, t_0''); u_0'$. Furthermore, $(\mathbf{ch}\ A.\, (\mathbf{ch}\ C.\, t_0''); u_0', \mathbf{ch}\ A \cup C.\, t_0''; u_0') \in R$.
  - $t_0 \xrightarrow{\checkmark} t_0'$, $t_0' \xrightarrow{c!v, A} t_0''$, $c \notin C$, $A \cap C = \emptyset$, $fc(v) \cap C \neq \emptyset$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C \cup A.\, t_0''; u_0'$, derived with R_{11}, R_{13} and $(C \setminus fc(v)) \cup (A \cup (C \cap fc(v))) = C \cup A$. Then we can derive with R_{13}, R_{11} that $(\mathbf{ch}\ C.t_0); u_0 \xrightarrow{\tau} \mathbf{ch}\ A \cup (C \cap fc(v)).(\mathbf{ch}\ C \setminus fc(v).\, t_0''); u_0'$. Furthermore, $(\mathbf{ch}\ A \cup (C \cap fc(v)).\, (\mathbf{ch}\ C \setminus fc(v).\, t_0''); u_0', \mathbf{ch}\ C \cup A.\, t_0''; u_0') \in R$.

– Axiom (24): $t; \mathbf{ch}\ C.\, u = \mathbf{ch}\ C.\, t; u$ if $C \cap fc(t) = \emptyset$.
Let $R = \{(\mathbf{ch}\ E.\, t_0; \mathbf{ch}\ D.\, u_0, \mathbf{ch}\ E \cup D.\, t_0; u_0)\}$ be a relation. The proof for R being a bisimulation is similar to the proof of axiom (23).

– Axiom (25): $P(\mathbf{v}) = \tau; t\langle \mathbf{v}/\mathbf{x} \rangle$ if $\Theta : P(\mathbf{x} : \mathbf{T}) \mapsto t$.
Let $R = \{(P(\mathbf{v}), \tau; u\langle \mathbf{v}/\mathbf{x} \rangle) \mid u \in \textsc{Term}, \Theta : P(\mathbf{x} : \mathbf{T}) \mapsto u\} \cup \{(u, \mathbf{1}; u) \mid u \in \textsc{Term}\}$ be a relation. With R_{16} we know that $P(\mathbf{v}) \xrightarrow{\tau} u\langle \mathbf{v}/\mathbf{x} \rangle$. With R_2, R_9 we know that $\tau; u\langle \mathbf{v}/\mathbf{x} \rangle \xrightarrow{\tau} \mathbf{1}; u\langle \mathbf{v}/\mathbf{x} \rangle$. Furthermore, $(u\langle \mathbf{v}/\mathbf{x} \rangle, \mathbf{1}; u\langle \mathbf{v}/\mathbf{x} \rangle) \in R$. The proof for $\{(u, \mathbf{1}; u) \mid u \in \textsc{Term}\}$ is similar to the proof for axiom (3). Therefore, R satisfies the bisimulation conditions of Definition 5.

– Axiom (26): $spawn(\mathbf{0}) = \mathbf{1}$.
Let $R = \{(spawn(\mathbf{0}), \mathbf{1})\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
- Assume $spawn(\mathbf{0}) \xrightarrow{\omega} t'$. Then we know with $\mathbf{0} \not\xrightarrow{}$ and R_{14} that $\omega = \checkmark$ and $t' = spawn(\mathbf{0})$. Similarly, we know with R_1 that $\mathbf{1} \xrightarrow{\checkmark} \mathbf{1}$. Furthermore, $(spawn(\mathbf{0}), \mathbf{1}) \in R$.
- For the assumption $\mathbf{1} \xrightarrow{\omega} t'$ analogous.

– Axiom (27): $spawn(t); spawn(u) = spawn(u); spawn(t)$. Let

$$R = \{(\mathbf{ch}\ D.\, spawn(t_0); spawn(u_0), \mathbf{ch}\ D.\, spawn(u_0); spawn(t_0)) \mid t_0; u_0 \in \textsc{Term}\}\ .$$

We show that R satisfies the bisimulation conditions of Definition 5. First, we assume $D = \emptyset$. For $D \neq \emptyset$ we can apply techniques used in the congruence proof for channel creation.

- We assume $spawn(t_0); spawn(u_0) \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  - $t_0 \xrightarrow{\alpha} t_0'$, $\omega = \alpha$ and $t_{new} = spawn(t_0'); spawn(u_0')$, derived with R_{15}, R_9. Then we can derive with R_{14}, R_{15}, R_{10} that $spawn(u_0); spawn(t_0) \xrightarrow{\alpha} spawn(u_0); spawn(t_0')$. Furthermore, $(spawn(t_0'); spawn(u_0), spawn(u_0); spawn(t_0')) \in R$.
  - $u_0 \xrightarrow{\alpha} u_0'$, $\omega = \alpha$ and $t_{new} = spawn(t_0); spawn(u_0')$, derived with R_{14}, R_{15}, R_{10}. Then we know with R_{15}, R_9 that $spawn(u_0); spawn(t_0) \xrightarrow{\alpha} spawn(u_0'); spawn(t_0)$. Furthermore, $(spawn(t_0); spawn(u_0'), spawn(u_0'); spawn(t_0)) \in R$.
  - $t_0 \xrightarrow{\alpha} t_0'$, $u_0 \xrightarrow{\alpha'} u_0'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $t_{new} =$ $\mathbf{ch}\ C.\, spawn(t_0'); spawn(u_0')$, derived with R_{14}, R_{15}, R_{11}. Then we can derive with R_{14}, R_{15}, R_{11} that $spawn(u_0); spawn(t_0) \xrightarrow{\tau} \mathbf{ch}\ C.\, spawn(u_0'); spawn(t_0')$. Furthermore, $(\mathbf{ch}\ C.\, spawn(t_0'); spawn(u_0'), \mathbf{ch}\ C.\, spawn(u_0'); spawn(t_0')) \in R$.

* $\omega = \checkmark$ and $t_{new} = spawn(t_0); spawn(u_0)$, derived with R_{14}, R_{10}. Then we can deduce with R_{14}, R_{10} that $spawn(u_0); spawn(t_0) \xrightarrow{\checkmark} spawn(u_0); spawn(t_0)$. Furthermore, $(spawn(t_0); spawn(u_0), spawn(u_0); spawn(t_0)) \in R$.

- For the assumption $spawn(u_0); spawn(t_0) \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (28): $spawn(t); spawn(u) = spawn(spawn(t); u)$. Let

$$R = \{(\mathbf{ch}\ D.\ spawn(t_0); spawn(u_0), spawn(\mathbf{ch}\ D.\ spawn(t_0); u_0)) \mid t_0, u_0 \in \text{TERM}\}\ .$$

We show that R satisfies the bisimulation conditions. First, we assume $D = \emptyset$. For $D \neq \emptyset$ the proof is similar to the proof for axiom (15).

- We assume $spawn(t_0); spawn(u_0) \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  * $t_0 \xrightarrow{\alpha} t_0'$, $\omega = \alpha$ and $t_{new} = spawn(t_0'); spawn(u_0)$, derived with R_{15}, R_9. Then we can deduce with R_{15}, R_9 that $spawn(spawn(t_0); u_0) \xrightarrow{\alpha} spawn(spawn(t_0'); u_0)$. Furthermore, $(spawn(t_0'); spawn(u_0), spawn(spawn(t_0'); u_0)) \in R$.
  * $u_0 \xrightarrow{\alpha} u_0'$, $\omega = \alpha$ and $t_{new} = spawn(t_0); spawn(u_0')$, derived with R_{14}, R_{15}, R_{10}. Then we can deduce with R_{14}, R_{10}, R_{15} that there is a transition

    $$spawn(spawn(t_0); u_0) \xrightarrow{\alpha} spawn(spawn(t_0); u_0')\ .$$

    Furthermore, $(spawn(t_0); spawn(u_0'), spawn(spawn(t_0); u_0')) \in R$.
  * $t_0 \xrightarrow{\alpha} t_0'$, $u_0 \xrightarrow{\alpha'} u_0'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $t_{new} =$ $\mathbf{ch}\ C.\ spawn(t_0'); spawn(u_0')$, derived with R_{14}, R_{15}, R_{11}. Then we can deduce with R_{14}, R_{15}, R_{11} that $spawn(spawn(t_0); u_0) \xrightarrow{\tau} spawn(\mathbf{ch}\ C.\ spawn(t_0'); u_0')$. Furthermore, $(\mathbf{ch}\ C.\ spawn(t_0'); spawn(u_0'), spawn(\mathbf{ch}\ C.\ spawn(t_0'); u_0')) \in R$.
  * $\omega = \checkmark$ and $t_{new} = spawn(t_0); spawn(u_0)$, derived with R_{14}, R_{10}. Similarly, we can derive with R_{14} that $spawn(spawn(t_0); u_0) \xrightarrow{\checkmark} spawn(spawn(t_0); u_0)$. Furthermore, $(spawn(t_0); spawn(u_0), spawn(spawn(t_0); u_0)) \in R$.
- For the assumption $spawn(spawn(t_0); u_0) \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (29): $spawn(t_1; spawn(t_2) + t_3) = spawn(t_1; t_2 + t_3)$. Let

$$\begin{aligned}
R_1 &= \{(spawn(u_1; spawn(u_2) + u_3), spawn(u_1; u_2 + u_3)) \mid u_1, u_2, u_3 \in \text{TERM}\} \\
R_2 &= \{(spawn(\mathbf{ch}\ C.\ u_1; spawn(u_2)), spawn(\mathbf{ch}\ C.\ u_1; u_2)) \mid u_1, u_2 \in \text{TERM}\} \\
R_3 &= \{(u, u) \mid u \in \text{TERM}\}\ .
\end{aligned}$$

Let $R = R_1 \cup R_2 \cup R_3$. We show that it satisfies the bisimulation conditions of Definition 5. First, we consider R_1.

- We assume $spawn(u_1; spawn(u_2) + u_3) \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  * $u_1 \xrightarrow{\alpha} u_1'$, $\omega = \alpha$ and $t_{new} = spawn(u_1'; spawn(u_2))$, derived with R_9, R_7, R_{15}. Then we can derive with R_9, R_7, R_{15} that $spawn(u_1; u_2 + u_3) \xrightarrow{\alpha} spawn(u_1'; u_2)$. Furthermore, $(spawn(u_1'; spawn(u_2)), spawn(u_1'; u_2)) \in R$.
  * $u_1 \xrightarrow{\checkmark} u_1'$, $u_2 \xrightarrow{\alpha} u_2'$, $\omega = \alpha$ and $t_{new} = spawn(u_1'; spawn(u_2'))$, derived with R_{15}, R_{10}, R_7. Then we can deduce with R_{10}, R_7, R_{15} that $spawn(u_1; u_2 + u_3) \xrightarrow{\alpha} spawn(u_1'; u_2')$. Furthermore, $(spawn(u_1'; spawn(u_2')), spawn(u_1'; u_2')) \in R$.
  * $u_1 \xrightarrow{\checkmark} u_1'$, $u_1' \xrightarrow{\alpha} u_1''$, $u_2 \xrightarrow{\alpha'} u_2'$, $\{\alpha, \alpha'\} = \{c?v, (c!v, C)\}$, $\omega = \tau$ and $t_{new} = spawn(\mathbf{ch}\ C.\ u_1''; spawn(u_2'))$, derived with R_{15}, R_{11}, R_7. Then we can deduce with R_{11}, R_7, R_{15} that $spawn(u_1; u_2 + u_3) \xrightarrow{\tau} spawn(\mathbf{ch}\ C.\ u_1''; u_2')$. Furthermore, $(spawn(\mathbf{ch}\ C.\ u_1''; spawn(u_2')), spawn(\mathbf{ch}\ C.\ u_1''; u_2')) \in R$.
  * $u_3 \xrightarrow{\alpha} u_3'$, $\omega = \alpha$ and $t_{new} = spawn(u_3')$, derived with R_8, R_{15}. Then we can derive with R_8, R_{15} that $spawn(u_1; u_2 + u_3) \xrightarrow{\alpha} spawn(u_3')$. Furthermore, $(spawn(u_3), spawn(u_3)) \in R$.
  * $\omega = \checkmark$ and $t_{new} = spawn(u_1; spawn(u_2) + u_3)$, derived with R_{14}. Similarly, we can derive with R_{14} that $spawn(u_1; u_2 + u_3) \xrightarrow{\checkmark} spawn(u_1; u_2 + u_3)$. Furthermore, $(spawn(u_1; spawn(u_2) + u_3), spawn(u_1; u_2 + u_3)) \in R$.
- For the assumption $spawn(u_1; u_2 + u_3) \xrightarrow{\omega} t_{new}$ analogous.

The proof for R_2 is similar to the proof for R_1 with $u_3 = \mathbf{0}$. Furthermore, we can apply techniques used for the congruence proof for channel restriction.

– Axiom (30): $E!F = E'!F'$ if $E =_\beta E'$ and $F =_\beta F'$.
Let $R = \{(E!F, E'!F') \mid E =_\beta E', F =_\beta F'\} \cup \{(t,t) \mid t \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. With $E =_\beta E'$ and $E : T$ *channel*, $E' : T$ *channel* we know that $\exists c : [\![E]\!] = c = [\![E']\!]$.
  - We assume $E!F \xrightarrow{\omega} t_{new}$. Then we know with R_4 that $\exists v : [\![F]\!] = v$, $\omega = c!v$ and $t_{new} = \mathbf{1}$. With $F =_\beta F'$ we know that $\exists v', v =_\beta v' : [\![F']\!] = v'$. Then we can deduce with R_4 that $E'!F' \xrightarrow{c!v'} \mathbf{1}$. Furthermore, $(\mathbf{1}, \mathbf{1}) \in R$.
  - For the assumption $E'!F' \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (31): $E?x;t = E'?x;t$ if $E =_\beta E'$.
Let $R = \{(E?x;u, E'?x;u) \mid u \in \text{TERM}, E =_\beta E'\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $E?x;u \xrightarrow{\omega} u_{new}$. Then we know with R_5 that $\exists c : [\![E]\!] = c$, $\omega = c?v$ and $u_{new} = u\langle v/x\rangle$. With $E : T$ *channel*, $E' : T$ *channel* and $E =_\beta E'$ we know that $[\![E']\!] = c$. Let $v' =_\beta v$ be arbitrarly. Then we can derive with R_5 that $E'?x;u \xrightarrow{c?v'} u\langle v'/x\rangle$. With Lemma 7 we can deduce that $u\langle v/x\rangle \sim_\beta u\langle v'/x\rangle$. Therefore, $(u\langle v/x\rangle, u\langle v'/x\rangle) \in R$.
  - For the assumption $E'?x;u \xrightarrow{\omega} u_{new}$ analogous.

– Axiom (32): $\{\mathbf{x} \leftarrow \mathbf{E}\};t = \{\mathbf{x} \leftarrow \mathbf{E}'\};t$ if $\mathbf{E} =_\beta \mathbf{E}'$.
Let $R = \{(\{\mathbf{x} \leftarrow \mathbf{E}\};u, \{\mathbf{x} \leftarrow \mathbf{E}'\};u) \mid u \in \text{TERM}, \mathbf{E} =_\beta \mathbf{E}'\}$ be a relation. We prove that it satisfies the bisimulation conditions of Definition 5.
  - We assume $\{\mathbf{x} \leftarrow \mathbf{E}\};u \xrightarrow{\omega} u_{new}$. Then we know with R_6 that $\exists \mathbf{v} : [\![\mathbf{E}]\!] = \mathbf{v}$, $\omega = \tau$ and $u_{new} = u\langle \mathbf{v}/\mathbf{x}\rangle$. With $\mathbf{E} =_\beta \mathbf{E}'$ we can deduce that $\exists \mathbf{v}', \mathbf{v} =_\beta \mathbf{v}' : [\![\mathbf{E}']\!] = \mathbf{v}'$. Then we can deduce with R_6 that $\{\mathbf{x} \leftarrow \mathbf{E}'\};u \xrightarrow{\tau} u\langle \mathbf{v}'/\mathbf{x}\rangle$. With Lemma 7 we can deduce that $u\langle \mathbf{v}/\mathbf{x}\rangle \sim_\beta u\langle \mathbf{v}'/\mathbf{x}\rangle$. Therefore, $(u\langle \mathbf{v}/\mathbf{x}\rangle, u\langle \mathbf{v}'/\mathbf{x}\rangle) \in R$.
  - For the assumption $\{\mathbf{x} \leftarrow \mathbf{E}'\};u \xrightarrow{\omega} u_{new}$ analogous.

– Axiom (33): $[E] = [E']$ if $E =_\beta E'$.
Let $R = \{([E], [E']) \mid E =_\beta E'\} \cup \{(t,t) \mid t \in \text{TERM}\}$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5.
  - We assume $[E] \xrightarrow{\omega} t_{new}$. Then we know with R_3 that $[\![E]\!] = \mathit{true}$, $\omega = \tau$ and $t_{new} = \mathbf{1}$. With $E =_\beta E'$ we can deduce that $[\![E']\!] = \mathit{true}$. Then we can derive with R_3 that $[E'] \xrightarrow{\tau} \mathbf{1}$. Furthermore, $(\mathbf{1}, \mathbf{1}) \in R$.
  - For the assumption $[E'] \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (34): $P(\mathbf{E}) = P(\mathbf{E}')$ if $\mathbf{E} =_\beta \mathbf{E}'$.
Let $R = \{(P(\mathbf{E}), P(\mathbf{E}')) \mid \mathbf{E} =_\beta \mathbf{E}'\} \cup \sim_\beta$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. Let $\Theta : P(\mathbf{x} : \mathbf{T}) \mapsto t$.
  - We assume $P(\mathbf{E}) \xrightarrow{\omega} t_{new}$. Then we know with R_{16} that $\exists \mathbf{v} : [\![\mathbf{E}]\!] = \mathbf{v}$, $\omega = \tau$ and $t_{new} = t\langle \mathbf{v}/\mathbf{x}\rangle$. With $\mathbf{E} =_\beta \mathbf{E}'$ we can deduce that $\exists \mathbf{v}', \mathbf{v} =_\beta \mathbf{v}' : [\![\mathbf{E}']\!] = \mathbf{v}'$. Then we can derive with R_{16} that $P(\mathbf{E}') \xrightarrow{\tau} t\langle \mathbf{v}'/\mathbf{x}\rangle$. With Lemma 7 we know that $t\langle \mathbf{v}/\mathbf{x}\rangle \sim_\beta t\langle \mathbf{v}'/\mathbf{x}\rangle$. Therefore, $(t\langle \mathbf{v}/\mathbf{x}\rangle, t\langle \mathbf{v}'/\mathbf{x}\rangle) \in R$.
  - For the assumption $P(\mathbf{E}') \xrightarrow{\omega} t_{new}$ analogous.

– Axiom (36): Expansion law.
Let $t_0 = \sum_{i\in\mathcal{I}} \delta_i; t_i + \sum_{j\in\mathcal{J}} E_j?x_j; t_j$, let $u_0 = \sum_{k\in\mathcal{K}} \delta_k; u_k + \sum_{l\in\mathcal{L}} E_l?x_l; u_l$ Let

$$R_1 = \left\{ \begin{array}{l} (\mathit{spawn}(t_0); u_0, \\ \sum_{i\in\mathcal{I}} \delta_i; \mathit{spawn}(t_i); u_0 + \sum_{k\in\mathcal{K}} \delta_k; \mathit{spawn}(t_0); u_k + \\ \sum_{j\in\mathcal{J}} E_j?x_j; \mathit{spawn}(t_j); u_0 + \sum_{l\in\mathcal{L}} E_l?x_l; \mathit{spawn}(t_0); u_l + \\ \sum_{\delta_i = (c!v, C), [\![E_l]\!] = c}^{i\in\mathcal{I}, l\in cL} \tau; \mathbf{ch}\ C.\ \mathit{spawn}(t_i); u_l\langle v/x_l\rangle + \\ \sum_{[\![E_j]\!] = c, \delta_k = c!v, C}^{j\in\mathcal{J}, k\in\mathcal{K}} \tau; \mathbf{ch}\ C.\ \mathit{spawn}(t_j\langle v/x_j\rangle); u_k) \mid \\ \forall j \in \mathcal{J} : x_j \notin \mathit{fv}(u_0), \forall l \in \mathcal{L} : x_l \notin \mathit{fv}(t) \end{array} \right\}$$

$$R_2 = \{(\mathbf{ch}\ C.\ \mathit{spawn}(\mathbf{1}; t'); u', \mathbf{1}; \mathbf{ch}\ C.\ \mathit{spawn}(t'); u') \mid t', u' \in \text{TERM}\}$$
$$R_3 = \{(\mathbf{ch}\ C.\ \mathit{spawn}(t'); \mathbf{1}; u', \mathbf{1}; \mathbf{ch}\ C.\ \mathit{spawn}(t'); u') \mid t', u' \in \text{TERM}\}$$
$$R_4 = \{(t', t') \mid t' \in \text{TERM}\}$$

Let $R = R_1 \cup R_2 \cup R_3 \cup R_4$ be a relation. We show that it satisfies the bisimulation conditions of Definition 5. We denote the left hand elements of the pairs in R by t_L and the right hand elements by t_R.

- We assume $t_L \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  - $\exists i \in \mathcal{I} : \delta_i \xrightarrow{\delta_i} \mathbf{1}$, $\omega = \delta_i$ and $t_{new} = spawn(\mathbf{1}; t_i); u_0$, derived with R_9, R_{15} and the choice rules. If $\delta_i = [E]$ then $\omega = \tau$. Then we can derive with R_9 and the choice rules that $t_R \xrightarrow{\delta_i} \mathbf{1}; spawn(t_i); u_0$. Furthermore, $(spawn(\mathbf{1}; t_i); u_0, \mathbf{1}; spawn(t_i); u_0) \in R$.
  - $\exists k \in \mathcal{K} : \delta_k \xrightarrow{\delta_k} \mathbf{1}$, $\omega = \delta_k$ and $t_{new} = spawn(t_0); \mathbf{1}; u_k$, derived with R_{14}, R_9, R_{10} and the choice rules. If $\delta_k = [E]$ then $\omega = \tau$. Then we can derive with R_9 and the choice rules that $t_R \xrightarrow{\omega} \mathbf{1}; spawn(t_0); u_k$. Furthermore, $(spawn(t_0); \mathbf{1}; u_k, \mathbf{1}; spawn(t_0); u_k) \in R$.
  - $\exists j \in \mathcal{J} : E_j?x_j; t_j \xrightarrow{c?v} t_j\langle v/x_j\rangle$, $\omega = c?v$ and $t_{new} = spawn(t_j\langle v/x_j\rangle); u_0$, derived with R_5, R_{15}, R_9 and the choice rules. Then we can derive with R_5 and the choice rules that $t_R \xrightarrow{c?v} (spawn(t_j); u_0)\langle v/x_j\rangle$. With $x_j \notin fv(u_0)$ we can deduce that $(spawn(t_j); u_0)\langle v/x_j\rangle = spawn(t_j\langle v/x_j\rangle); u_0$. Furthermore, $(spawn(t_j\langle v/x_j\rangle); u_0, spawn(t_j\langle v/x_j\rangle); u_0) \in R$.
  - $\exists l \in \mathcal{L} : E_l?x_l; u_l \xrightarrow{c?v} u_l\langle v/x_l\rangle$, $\omega = c?v$ and $t_{new} = spawn(t_0); u_l\langle v/x_l\rangle$, derived with R_{14}, R_5, R_{10} and the choice rules. Then we can derive with R_5 and the choice rules that $t_R \xrightarrow{c?v} (spawn(t_0); u_l)\langle v/x_l\rangle$. With $x_l \notin fv(t_0)$ we can deduce that $(spawn(t_0); u_l)\langle v/x_l\rangle = spawn(t_0); u_l\langle v/x_l\rangle$. Furthermore, it is the case that $(spawn(t_0); u_l\langle v/x_l\rangle, spawn(t_0); u_l\langle v/x_l\rangle) \in R$.
  - $\exists i \in \mathcal{I} : \delta_i \xrightarrow{c!v,C} \mathbf{1}$, $\exists l \in \mathcal{L} : E_l?x_l; u_l \xrightarrow{c?v} u_l\langle v/x_l\rangle$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ spawn(\mathbf{1}; t_i); u_l\langle v/x_l\rangle$, derived with R_{11} and the choice rules. Then we can derive with R_2, R_9 and the choice rules that $t_R \xrightarrow{\tau} \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_i); u_l\langle v/x_l\rangle$. Furthermore, $(\mathbf{ch}\ C.\ spawn(\mathbf{1}; t_i); u_l\langle v/x_l\rangle, \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_i); u_l\langle v/x_l\rangle) \in R$.
  - $\exists j \in \mathcal{J} : E_j?x_j; t_j \xrightarrow{c?v} t_j\langle v/x_j\rangle$, $\exists k \in \mathcal{K} : \delta_k \xrightarrow{c!v,C} \mathbf{1}$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); \mathbf{1}; u_k$, derived with R_{11} and the choice rules. Then we can derive with R_2, R_9 and the choice rules that $t_R \xrightarrow{\tau} \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); u_k$. Furthermore, $(\mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); \mathbf{1}; u_k, \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); u_k) \in R$.
- We assume $t_R \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
  - $\exists i \in \mathcal{I} : \delta_i \xrightarrow{\delta_i} \mathbf{1}$, $\omega = \delta_i$ and $t_{new} = \mathbf{1}; spawn(t_i); u_0$, derived with R_9 and the choice rules. If $\delta_i = [E]$ then $\omega = \tau$. Then we can derive with R_9, R_{15} and the choice rules that $t_L \xrightarrow{\omega} spawn(\mathbf{1}; t_i); u_0$. Furthermore, it is the case that $(spawn(\mathbf{1}; t_i); u_0, \mathbf{1}; spawn(t_i); u_0) \in R$.
  - $\exists k \in \mathcal{K} : \delta_k \xrightarrow{\delta_k} \mathbf{1}$, $\omega = \delta_k$ and $t_{new} = \mathbf{1}; spawn(t_0); u_k$, derived with R_9 and the choice rules. If $\delta_k = [E]$ then $\omega = \tau$. Then we can derive with R_{14}, R_9, R_{10} and the choice rules that $t_L \xrightarrow{\omega} spawn(t_0); \mathbf{1}; u_k$. Furthermore, it is the case that $(spawn(t_0); \mathbf{1}; u_k, \mathbf{1}; spawn(t_0); u_k) \in R$.
  - $\exists j \in \mathcal{J} : E_j?x_j; t_j \xrightarrow{c?v} t_j\langle v/x_j\rangle$, $\omega = c?v$ and $t_{new} = (spawn(t_j); u_0)\langle v/x_j\rangle$, derived with R_5 and the choice rules. Due to $x_j \notin fv(u_0)$, it is the case that $(spawn(t_j); u_0)\langle v/x_j\rangle = spawn(t_j\langle v/x_j\rangle); u_0$. With R_5, R_{15}, R_9 and the choice rules we can derive that $t_L \xrightarrow{c?v} spawn(t_j\langle v/x_j\rangle); u_0$. Furthermore, it is the case that $(spawn(t_j\langle v/x_j\rangle); u_0, spawn(t_j\langle v/x_j\rangle); u_0) \in R$.
  - $\exists l \in \mathcal{L} : E_l?x_l; u_l \xrightarrow{c?v} u_l\langle v/x_l\rangle$, $\omega = c?v$ and $t_{new} = (spawn(t_0); u_l)\langle v/x_l\rangle$, derived with R_5 and the choice rules. Due to $x_l \notin fv(t_0)$, we can deduce that $(spawn(t_0); u_l)\langle v/x_l\rangle = spawn(t_0); u_l\langle v/x_l\rangle$. With R_5, R_{14}, R_{10} and the choice rules we can deduce that $t_L \xrightarrow{c?v} spawn(t_0); u_l\langle v/x_l\rangle$. Furthermore, we know that $(spawn(t_0); u_l\langle v/x_l\rangle, spawn(t_0); u_l\langle v/x_l\rangle) \in R$.
  - $\exists i \in \mathcal{I} : \delta_i \xrightarrow{c!v,C} \mathbf{1}$, $\exists l \in \mathcal{L} : E_l?x_l; u_l \xrightarrow{c?v} u_l\langle v/x_l\rangle$, $\omega = \tau$ and $t_{new} = \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_i); u_l\langle v/x_l\rangle$, derived with R_2, R_9 and the choice rules. Then we can derive with R_{11} and the choice rules that $t_L \xrightarrow{\tau} \mathbf{ch}\ C.\ spawn(\mathbf{1}; t_i); u_l\langle v/x_l\rangle$. Furthermore, $(\mathbf{ch}\ C.\ spawn(\mathbf{1}; t_i); u_l\langle v/x_l\rangle, \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_i); u_l\langle v/x_l\rangle) \in R$.

* $\exists j \in \mathcal{J} : E_j?x_j; t_j \xrightarrow{c?v} t_j\langle v/x_j\rangle$, $\exists k \in \mathcal{K} : \delta_k \xrightarrow{c!v,C} \mathbf{1}$, $\omega = \tau$ and $t_{new} = \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); u_k$, derived with R_2, R_9 and the choice rules. Then we can deduce with R_{11} and the choice rules that $t_L \xrightarrow{\tau} \mathbf{ch}\ C.spawn(t_j\langle v/x_j\rangle); \mathbf{1}; u_k$. Furthermore, $(\mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); \mathbf{1}; u_k, \mathbf{1}; \mathbf{ch}\ C.\ spawn(t_j\langle v/x_j\rangle); u_k) \in R$.

For R_2 and R_3 we can use the axioms (3), (4) and the congruence of channel restriction. □

Lemma 22. *If $\sigma =_\beta \sigma'$, then $t\sigma \approx_\beta t\sigma'$.*

Proof. Follows directly from Lemma 7 and $\sim_\beta \subset \approx_\beta$. □

Theorem 10. $\approx_\beta$ is a congruence for *spawn*, variable declaration, input actions, channel restriction and sequential composition.

Proof. We prove the congruence property of $\simeq_\beta$ for the mentioned operators. For each step $t \xrightarrow{c!v,C} t'$ we assume that $C \cap fc(t) = \emptyset$.

- $spawn(t) \approx_\beta spawn(u)$ if $t \approx_\beta u$.
  Let $R = \{(spawn(t_0), spawn(u_0)) \mid t_0 \approx_\beta u_0\}$ be a relation. We show that it satisfies the conditions of Definition 9. Since R is symmetric, we need to show only one direction.
  We assume $spawn(t_0) \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $t_0 \xrightarrow{c?v} t_0'$, $\omega = c?v$ and $t' = spawn(t_0')$, derived with R_{15}. With $t_0 \approx_\beta u_0$ we know that $\forall v', v =_\beta v', \exists u_0' : u_0 \xRightarrow{\widehat{c?v'}} u_0'$ and $t_0' \approx_\beta u_0'$. Then we can deduce by the repeated application of R_{15} that $spawn(u_0) \xRightarrow{\widehat{c?v'}} spawn(u_0')$. Furthermore, $(spawn(t_0'), spawn(u_0')) \in R$.
  - $t_0 \xrightarrow{\alpha} t_0'$, $\alpha = \tau$ or $\alpha = (c!v, C)$, $\omega = \alpha$ and $t' = spawn(t_0')$, derived with R_{15}. With $t_0 \approx_\beta u_0$ we know that $\exists \alpha', \alpha =_\beta \alpha', \exists u_0' : u_0 \xRightarrow{\widehat{\alpha'}} u_0'$ and $t_0' \approx_\beta u_0'$. Then we can derive with the repeated application of R_{15} that $spawn(u_0) \xRightarrow{\widehat{\alpha'}} spawn(u_0')$. Furthermore, $(spawn(t_0'), spawn(u_0')) \in R$.
  - $\omega = \checkmark$ and $t' = spawn(t_0)$, derived with R_{14}. Similarly, we can derive with R_{14} that $spawn(u_0) \xrightarrow{\checkmark} spawn(u_0)$ and, therefore, $spawn(u_0) \xRightarrow{\widehat{\checkmark}} spawn(u_0)$. Furthermore, $(spawn(t_0), spawn(u_0)) \in R$.
- $\{\mathbf{x} \leftarrow \mathbf{E}\}; t \approx_\beta \{\mathbf{x} \leftarrow \mathbf{E}'\}; u$ if $t \approx_\beta u$ and $\mathbf{E} =_\beta \mathbf{E}'$.
  Let $R = \{(\{\mathbf{x} \leftarrow \mathbf{E}\}; t_0, \{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0) \mid t_0 \approx_\beta u_0, \mathbf{E} =_\beta \mathbf{E}'\} \cup \approx_\beta$ be a relation. We show that it satisfies the conditions of Definition 9. Since R is symmetric, we need to show only one direction.
  We assume $\{\mathbf{x} \leftarrow \mathbf{E}\}; t_0 \xrightarrow{\omega} t'$ and distinguish the following cases:
  - $[\![\mathbf{E}]\!] = \mathbf{v}$, $t_0\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{c?v} t_0'$ and $t' = t_0'$, derived with R_6. Let $\mathbf{v}' = [\![\mathbf{E}']\!]$. With $t_0 \approx_\beta u_0$ and Lemma 22 we can deduce that $t_0\langle \mathbf{v}/\mathbf{x}\rangle \approx_\beta u_0\langle \mathbf{v}'/\mathbf{x}\rangle$. Then we know that $\forall v', v =_\beta v', \exists u_0' : u_0\langle \mathbf{v}'/\mathbf{x}\rangle \xRightarrow{\widehat{c?v'}} u_0'$ and $t_0' \approx_\beta u_0'$. Therefore, we can deduce with R_6 that $\{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0 \xRightarrow{\widehat{c?v'}} u_0'$. Furthermore, $(t_0', u_0') \in R$.
  - $[\![\mathbf{E}]\!] = \mathbf{v}$, $t_0\langle \mathbf{v}/\mathbf{x}\rangle \xrightarrow{\omega} t_0'$, $\omega \neq c?v$ and $t' = t_0'$, derived with R_6. Let $\mathbf{v}' = [\![\mathbf{E}']\!]$. With $t_0 \approx_\beta u_0$ and Lemma 22 we know that $t_0\langle \mathbf{v}/\mathbf{x}\rangle \approx_\beta u_0\langle \mathbf{v}'/\mathbf{x}\rangle$. Then we know that $\exists \omega', \omega =_\beta \omega', \exists u_0' : u_0\langle \mathbf{v}'/\mathbf{x}\rangle \xRightarrow{\widehat{\omega'}} u_0'$ and $t_0' \approx_\beta u_0'$. Then we can deduce with R_6 that $\{\mathbf{x} \leftarrow \mathbf{E}'\}; u_0 \xRightarrow{\widehat{\omega'}} u_0'$. Furthermore, $(t_0', u_0') \in R$.
- $E?x; t \approx_\beta E'?x; u$ if $t \approx_\beta u$ and $E =_\beta E'$.
  Let $R = \{(E?x; t_0, E'?x; u_0) \mid t_0 \approx_\beta u_0, E =_\beta E'\} \cup \approx_\beta$ be a relation. We show that it satisfies the conditions of Definition 9. Since R is symmetric, we need to show only one direction.
  We assume $E?x; t_0 \xrightarrow{\omega} t'$. For channel values beta-equivalence is identity; therefore $\exists c : [\![E]\!] = c = [\![E']\!]$. Then we know that $\omega = c?v$ and $t' = t_0\langle v/x\rangle$, derived with R_5.

Let $v' =_\beta v$ be arbitrarly. Then we can derive with R_5 that $E'?x; u_0 \xrightarrow{c?v'} u_0\langle v'/x\rangle$. Therefore, we have $E'?x; u_0 \xRightarrow{\widehat{c?v'}} u_0\langle v'/x\rangle$. With $t_0 \approx_\beta u_0$ we know that $\forall \sigma : t_0\sigma \approx_\beta u_0\sigma$. Therefore, we can deduce with Lemma 22 that $t_0\langle v/x\rangle \approx_\beta u_0\langle v'/x\rangle$. Furthermore, $(t_0\langle v/x\rangle, u_0\langle v'/x\rangle) \in R$.

– $\mathbf{ch}\ C.\ t \approx_\beta \mathbf{ch}\ C.\ u$ if $t \approx_\beta u$.
Let $\{(\mathbf{ch}\ D.\ t_0, \mathbf{ch}\ D.\ u_0) \mid t_0 \approx_\beta u_0\}$ be a relation. We show that it satisfies the conditions of Definition 9. Since R is symmetric, we need to show only one direction.
We assume $\mathbf{ch}\ C.\ t_0 \xrightarrow{\omega} t'$ and distinguish the following cases:
- $t_0 \xrightarrow{c?v} t_0'$, $\omega = c?v$, $c \notin C$, $fv(c) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.\ t_0'$, derived with R_{12}. With $t_0 \approx_\beta u_0$ we know that $\forall v', v =_\beta v', \exists u_0' : u_0 \xRightarrow{\widehat{c?v'}} u_0'$ and $t_0' \approx_\beta u_0'$. Furthermore, with $v =_\beta v'$ we know that $fc(v) = fc(v')$; therefore, we can deduce that $fc(v') \cap C = \emptyset$. Therefore, we know with the repeated application of R_{12} that $\mathbf{ch}\ C.\ u_0 \xRightarrow{\widehat{c?v'}} \mathbf{ch}\ C.\ u_0'$. Furthermore, $(\mathbf{ch}\ C.\ t_0', \mathbf{ch}\ C.\ u_0') \in R$.
- $t_0 \xrightarrow{\omega} t_0'$, $\omega \neq c?v$, $fc(\omega) \cap C = \emptyset$ and $t' = \mathbf{ch}\ C.\ t_0'$, derived with R_{12}. With $t_0 \approx_\beta u_0$ we know that $\exists \omega', \omega =_\beta \omega', \exists u_0' : u_0 \xRightarrow{\widehat{\omega'}} u_0'$ and $t_0' \approx_\beta u_0'$. Furthermore, with $\omega =_\beta \omega'$ we know that $fc(\omega) = fc(\omega')$; therefore, we know that $fc(\omega') \cap C = \emptyset$. Therefore, we can deduce with the repeated application of R_{12} that $\mathbf{ch}\ C.\ u_0 \xRightarrow{\widehat{\omega'}} \mathbf{ch}\ C.\ u_0'$. Furthermore, $(\mathbf{ch}\ C.\ t_0', \mathbf{ch}\ C.\ u_0') \in R$.
- $t_0 \xrightarrow{c!v, A} t_0'$, $A \cap C = \emptyset, fc(v) \cap C = \emptyset, \omega = (c!v, A \cup (C \cap fc(v)))$ and $t' = \mathbf{ch}\ C \backslash fc(v).t_0'$, derived with R_{13}. With $t_0 \approx_\beta u_0$ we know that $\exists v', v =_\beta v', \exists u_0' : u_0 \xRightarrow{\widehat{c!v', A}} u_0'$ and $t_0' \approx_\beta u_0'$. Furthermore, with $v =_\beta v'$ we know that $fc(v) = fc(v')$; therefore, we know that $fc(v') \cap C = fc(v) \cap C \neq \emptyset$. Therefore, we can deduce with R_{13} and the repeated application of R_{12} that $\mathbf{ch}\ C.\ u_0 \xRightarrow{c!v, A \cup \widehat{(C \cap fc(v'))}} \mathbf{ch}\ C \setminus fc(v').\ u_0'$. Furthermore, $(\mathbf{ch}\ C \setminus fc(v).\ t_0', \mathbf{ch}\ C \setminus fc(v').\ u_0') \in R$.

– $t_1; t_2 \approx_\beta u_1; u_2$ if $t_1 \approx_\beta u_1$ and $t_2 \approx_\beta u_2$.
Let $R = \{(\mathbf{ch}\ D.\ t_1; t_2, \mathbf{ch}\ D.\ u_1; u_2) \mid t_1 \approx_\beta u_1, t_2 \approx_\beta u_2\}$ be a relation. We show that it satisfies the conditions of Definition 9. Since R is symmetric, we need to show only one direction.
First, we assume $D = \emptyset$. For $D \neq \emptyset$ we can apply proof techniques similar to the congruence proof for channel restriction. We assume $t_1; t_2 \xrightarrow{\omega} t_{new}$ and distinguish the following cases:
- $t_1 \xrightarrow{c?v} t_1'$, $\omega = c?v$ and $t_{new} = t_1'; t_2$, derived with R_9. With $t_1 \approx_\beta u_1$ we know that $\forall v', v =_\beta v', \exists u_1' : u_1 \xRightarrow{\widehat{c?v'}} u_1'$ and $t_1' \approx_\beta u_1'$. Then we can deduce with the repeated application of R_9 that $u_1; u_2 \xRightarrow{\widehat{c?v'}} u_1'; u_2$. Furthermore, $(t_1'; t_2, u_1'; u_2) \in R$.
- $t_1 \xrightarrow{\alpha} t_1'$, $\alpha \neq c?v$ and $t_{new} = t_1'; t_2$, derived with R_9. With $t_1 \approx_\beta u_1$ we know that $\exists \alpha', \alpha =_\beta \alpha', \exists u_1' : u_1 \xRightarrow{\widehat{\alpha'}} u_1'$ and $t_1' \approx_\beta u_1'$. Then we can deduce with the repeated application of R_9 that $u_1; u_2 \xRightarrow{\widehat{\alpha'}} u_1'; u_2$. Furthermore, $(t_1'; t_2, u_1'; u_2) \in R$.
- $t_1 \xrightarrow{\checkmark} t_1'$, $t_2 \xrightarrow{c?v} t_2'$, $\omega = c?v$ and $t_{new} = t_1'; t_2'$, derived with R_{10}. With $t_1 \approx_\beta u_1$ we know that $\exists u_1' : u_1 \xRightarrow{\widehat{\checkmark}} u_1'$ and $t_1' \approx_\beta u_1'$. With $t_2 \approx_\beta u_2$ we know that $\forall v', v =_\beta v', \exists u_2' : u_2 \xRightarrow{\widehat{c?v'}} u_2'$ and $t_2' \approx_\beta u_2'$. Therefore, we can deduce with the repeated application of R_9 and R_{10} that $u_1; u_2 \xRightarrow{\widehat{c?v'}} u_1'; u_2'$. Furthermore, $(t_1'; t_2', u_1'; u_2') \in R$.
- $t_1 \xrightarrow{\checkmark} t_1'$, $t_2 \xrightarrow{\omega} t_2'$, $\omega \neq c?v$ and $t_{new} = t_1'; t_2'$, derived with R_{10}. With $t_1 \approx_\beta u_1$ we know that $\exists u_1' : u_1 \xRightarrow{\widehat{\checkmark}} u_1'$ and $t_1' \approx_\beta u_1'$. With $t_2 \approx_\beta u_2$ we know that $\exists \omega', \omega =_\beta \omega', \exists u_2' : u_2 \xRightarrow{\widehat{\omega'}} u_2'$ and $t_2' \approx_\beta u_2'$. Therefore, we can deduce with the repeated application of R_9 and R_{10} that $u_1; u_2 \xRightarrow{\widehat{\omega'}} u_1'; u_2'$. Furthermore, $(t_1'; t_2', u_1'; u_2') \in R$.
- $t_1 \xrightarrow{\checkmark} t_1'$, $t_1' \xrightarrow{c?v} t_1''$, $t_2 \xrightarrow{c!v, C} t_2'$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ t_1''; t_2'$, derived with

R_{11}. With $t_2 \approx_\beta u_2$ we know that $\exists v', v =_\beta v', \exists u_2' : u_2 \xrightarrow{\widehat{c!v',C}} u_2'$ and $t_2' \approx_\beta u_2'$. With $t_1 \approx_\beta u_1$ we know that $\exists u_1' : u_1 \stackrel{\widehat{\checkmark}}{\Longrightarrow} u_1'$, $t_1' \approx_\beta u_1'$ and $\forall v'', v =_\beta v'', \exists u_1'' : u_1' \xrightarrow{\widehat{c?v''}} u_1''$, $t_1'' \approx_\beta u_1''$. Then we can deduce with $v' =_\beta v''$ that $\exists u_v : u_1' \xrightarrow{\widehat{c?v'}} u_v$ and $t_1'' \approx_\beta u_v$. Therefore, we can deduce with R_{11} and the repeated application of R_9 and R_{10} that $u_1; u_2 \stackrel{\tau}{\Longrightarrow} \mathbf{ch}\ C.\ u_v; u_2'$. Furthermore, $(\mathbf{ch}\ C.\ t_1''; t_2', \mathbf{ch}\ C.\ u_v; u_2') \in R$.

- $t_1 \xrightarrow{\checkmark} t_1'$, $t_1' \xrightarrow{c!v,C} t_1''$, $t_2 \xrightarrow{c?v} t_2'$, $\omega = \tau$ and $t_{new} = \mathbf{ch}\ C.\ t_1''; t_2'$, derived with R_{11}. With $t_1 \approx_\beta u_1$ we know that $\exists u_1' : u_1 \stackrel{\widehat{\checkmark}}{\Longrightarrow} u_1'$, $t_1' \approx_\beta u_1'$ and $\exists v', v =_\beta v', \exists u_1'' : u_1' \xrightarrow{\widehat{c!v',C}} u_1''$, $t_1'' \approx_\beta u_1''$. With $t_2 \approx_\beta u_2$ we know that $\forall v'', v =_\beta v'', \exists u_2' : u_2 \xrightarrow{\widehat{c?v''}} u_2'$ and $t_2' \approx_\beta u_2'$. Then we can deduce with $v' =_\beta v''$ that $\exists u_v : u_2 \xrightarrow{\widehat{c?v'}} u_v$ and $t_2' \approx_\beta u_v$. Therefore, we can deduce with R_{11} and the repeated application of R_9 and R_{10} that $u_1; u_2 \stackrel{\tau}{\Longrightarrow} \mathbf{ch}\ C.\ u_1''; u_v$. Furthermore, $(\mathbf{ch}\ C.\ t_1''; t_2', \mathbf{ch}\ C.\ u_1''; u_v) \in R$. □

Proposition 11. For all $t \in \text{TERM} : \tau; t \approx_\beta t$ and $spawn(\tau; t) \approx_\beta spawn(t)$. Furthermore, $\mathbf{1}; \tau \approx_\beta \mathbf{1}$.

Proof. – $\tau; t \approx_\beta t$.

Let $R = \{(\tau; u, u) \mid u \in \text{TERM}\} \cup \sim_\beta$ be a relation. We show that it satisfies the conditions of Definition 9. From Theorem 8 we know that $\forall u \in \text{TERM} : \mathbf{1}; u \sim_\beta u$.

- We assume $\tau; u \xrightarrow{\omega} u_{new}$. Then $\omega = \tau$ and $u_{new} = \mathbf{1}; u$, derived with R_2, R_9. We know by definition of $\Rightarrow$ that $u \stackrel{\widehat{\tau}}{\Longrightarrow} u$. Furthermore, with $\mathbf{1}; u \sim_\beta u$ we know that $(\mathbf{1}; u, u) \in R$.
- We assume $u \xrightarrow{\omega} u'$. With R_2, R_9 we can deduce that $\tau; u \xrightarrow{\tau} \mathbf{1}; u$. With $u \sim_\beta \mathbf{1}; u$ we know that $(\mathbf{1}; u, u) \in R$. Furthermore, we can derive with R_{10} that $\mathbf{1}; u \xrightarrow{\omega} \mathbf{1}; u'$ and $(\mathbf{1}; u', u') \in R$. Therefore, we have $\tau; u \stackrel{\widehat{\omega}}{\Longrightarrow} \mathbf{1}; u'$ and $(\mathbf{1}; u', u') \in R$.

– $spawn(\tau; t) \approx_\beta spawn(t)$.

Let $R = \{(spawn(\tau; u), spawn(u)) \mid u \in \text{TERM}\} \cup \sim_\beta$ be a relation. We show that it satisfies the conditions of Definition 9. From Theorem 8 we know that $\forall u \in \text{TERM} : spawn(\mathbf{1}; u) \sim_\beta spawn(u)$.

- We assume $spawn(\tau; u) \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
  - $\tau; u \xrightarrow{\tau} \mathbf{1}; u$. Then $\omega = \tau$ and $u_{new} = spawn(\mathbf{1}; u)$, derived with R_2, R_9, R_{15}. We know by definition of $\Rightarrow$ that $u \stackrel{\widehat{\tau}}{\Longrightarrow} u$ and, therefore, $spawn(u) \stackrel{\widehat{\tau}}{\Longrightarrow} spawn(u)$. Furthermore, due to $spawn(\mathbf{1}; u) \sim_\beta spawn(u)$ we can deduce that $(spawn(\mathbf{1}; u), spawn(u)) \in R$.
  - $\omega = \checkmark$ and $u_{new} = spawn(\tau; u)$, derived with R_{14}. Similarly, we can deduce with R_{14} that $spawn(u) \xrightarrow{\checkmark} spawn(u)$ and, therefore, $spawn(u) \stackrel{\widehat{\checkmark}}{\Longrightarrow} spawn(u)$.. Furthermore, $(spawn(\tau; u), spawn(u)) \in R$.
- We assume $spawn(u) \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
  - $u \xrightarrow{\alpha} u'$, $\omega = \alpha$ and $u_{new} = spawn(u')$, derived with R_{15}. With R_2, R_9, R_{15} we can deduce that $spawn(\tau; u) \xrightarrow{\tau} spawn(\mathbf{1}; u)$. With $spawn(\mathbf{1}; u) \sim_\beta spawn(u)$ we know that $(spawn(\mathbf{1}; u), spawn(u)) \in R$. Furthermore, we can derive with R_{10} that $spawn(\mathbf{1}; u) \xrightarrow{\omega} spawn(\mathbf{1}; u')$ and $(spawn(\mathbf{1}; u'), spawn(u')) \in R$. Therefore, we have $spawn(\tau; u) \stackrel{\widehat{\omega}}{\Longrightarrow} spawn(\mathbf{1}; u')$ and $spawn(\mathbf{1}; u', u') \in R$.
  - $\omega = \checkmark$ and $u_{new} = spawn(u)$, derived with R_{14}. Similarly, we can derive with R_{14} that $spawn(\tau; u) \xrightarrow{\checkmark} spawn(\tau; u)$ and, therefore, $spawn(\tau; u) \stackrel{\widehat{\checkmark}}{\Longrightarrow} spawn(\tau; u)$. Furthermore, $(spawn(\tau; u), spawn(u)) \in R$.

– $\mathbf{1}; \tau \approx_\beta \mathbf{1}$.

Let $R = \{(\mathbf{1}; \tau, \mathbf{1})\} \cup \sim_\beta$ be a relation. We show that it satisfies the conditions of Definition 9.

- We assume $\mathbf{1};\tau \xrightarrow{\omega} t'$. Then $\omega = \tau$ and $t' = \mathbf{1};\mathbf{1}$, derived with R_2, R_{10}. With the definition of $\Rightarrow$ we know that $\mathbf{1} \stackrel{\widehat{\tau}}{\Longrightarrow} \mathbf{1}$. Furthermore, with $\mathbf{1};\mathbf{1} \sim_\beta \mathbf{1}$ we know that $(\mathbf{1};\mathbf{1},\mathbf{1}) \in R$.
- We assume $\mathbf{1} \xrightarrow{\omega} t'$. Then $\omega = \checkmark$ and $t' = \mathbf{1}$, derived with R_1. With R_2, R_{10} we can deduce that $\mathbf{1};\tau \xrightarrow{\tau} \mathbf{1};\mathbf{1}$. Then we know with $\mathbf{1};\mathbf{1} \sim_\beta \mathbf{1}$ that $(\mathbf{1};\mathbf{1},\mathbf{1}) \in R$. Furthermore, we can derive with R_1, R_{10} that $\mathbf{1};\mathbf{1} \xrightarrow{\checkmark} \mathbf{1};\mathbf{1}$. Therefore, we know that $\mathbf{1};\tau \stackrel{\widehat{\checkmark}}{\Longrightarrow} \mathbf{1};\mathbf{1}$ and $(\mathbf{1};\mathbf{1},\mathbf{1}) \in R$. □

Proposition 13. $t \sim_\beta u$ implies $t \simeq_\beta u$, and $t \simeq_\beta u$ implies $t \approx_\beta u$.

Proof. From Definition 5 and Definition 9 follows directly that $t \sim_\beta u$ implies $t \approx_\beta u$. For $t \sim_\beta u$ implies $t \simeq_\beta u$ we can use Definition 12 and the definition of $\Rightarrow$: For all $\omega \in \Omega$: If $t \xrightarrow{\omega} t'$ then $t \stackrel{\omega}{\Longrightarrow} t'$. For $t \simeq_\beta u$ implies $t \approx_\beta u$ we can use the congruence property of $\simeq_\beta$: With $t \simeq_\beta u$ we know that $t + \mathbf{0} \simeq_\beta u + \mathbf{0}$. Furthermore, we can deduce with $t + \mathbf{0} \sim_\beta t$ and $u + \mathbf{0} \sim_\beta u$ that $t + \mathbf{0} \approx_\beta t$ and $u + \mathbf{0} \approx_\beta u$. With $\mathbf{0} \not\xrightarrow{}$ and $t \simeq_\beta u$ we can deduce that $t \approx_\beta u$. □

Lemma 23. *If $\sigma =_\beta \sigma'$, then $t\sigma \simeq_\beta t\sigma'$.*

Proof. Follows directly from Lemma 7 and $\sim_\beta \subset \simeq_\beta$ (Proposition 13). □

Theorem 14. $\simeq_\beta$ is a congruence for the operators of TERM.

Proof. (of Theorem 14)

- $spawn(t) \simeq_\beta spawn(u)$ if $t \simeq_\beta u$.
  Let $R = \{(spawn(t_0), spawn(u_0)) \mid t_0 \simeq_\beta u_0\}$. The proof for R satisfying the conditions of Definition 12 is similar to the proof for Theorem 10. We can use Theorem 10 to show that the residuals after performing the initial transitions are weakly bisimilar.
- $\{\mathbf{x} \leftarrow \mathbf{E}\};t \simeq_\beta \{\mathbf{x} \leftarrow \mathbf{E}'\};u$ if $t \simeq_\beta u$ and $\mathbf{E} =_\beta \mathbf{E}'$.
  Let $R = \{(\{\mathbf{x} \leftarrow \mathbf{E}\};t_0, \{\mathbf{x} \leftarrow \mathbf{E}'\};u_0) \mid t_0 \simeq_\beta u_0, \mathbf{E} =_\beta \mathbf{E}'\}$. The proof for R satisfying the conditions of Definition 12 is similar to the proof for Theorem 10. We can use Theorem 10 to show that the residuals after performing the initial transitions are weakly bisimilar. Furthermore, we can use Lemma 23 to show that the premises of R_6 are satisfied.
- $E?x;t \simeq_\beta E'?x;u$ if $t \simeq_\beta u$ and $E =_\beta E'$.
  Let $R = \{(E?x;t_0, E'?x;u_0) \mid t_0 \simeq_\beta u_0, E =_\beta E'\}$. The proof for R satisfying the conditions of Definition 12 is similar to the proof for Theorem 10. We can use Theorem 10 and Lemma 23 to show that the residuals after performing the initial transitions are weakly bisimilar.
- $\mathbf{ch}\ C.\ t \simeq_\beta \mathbf{ch}\ C.\ u$ if $t \simeq_\beta u$.
  Let $R = \{(\mathbf{ch}\ D.\ t_0, \mathbf{ch}\ D.\ u_0) \mid t_0 \simeq_\beta u_0\}$. The proof for R satisfying the conditions of Definition 12 is similar to the proof for Theorem 10. We can use Theorem 10 to show that the residuals after performing the initial transitions are weakly bisimilar.
- $t_1;t_2 \simeq_\beta u_1;u_2$ if $t_1 \simeq_\beta u_1$ and $t_2 \simeq_\beta u_2$.
  Let $R = \{(t_1;t_2, u_1;u_2) \mid t_1 \simeq_\beta u_1, t_2 \simeq_\beta u_2\}$. The proof for R satisfying the conditions of Definition 12 is similar to the proof for Theorem 10. We can use Theorem 10 to show that the residuals after performing the initial transitions are weakly bisimilar.
- $t_1 + t \simeq_\beta t_2 + t$ if $t_1 \simeq_\beta t_2$.
  Let $R = \{(u_1 + u, u_2 + u) \mid u_1 \simeq_\beta u_2\}$ be a relation. We show that is satisfies the conditions of Definition 12. Since R is symmetric, we need to show only one direction.
  We assume $u_1 + u \xrightarrow{\omega} u_{new}$ and distinguish the following cases:
  - $u_1 \xrightarrow{c?v} u_1'$, $\omega = c?v$ and $u_{new} = u_1'$, derived with R_7. With $u_1 \simeq_\beta u_2$ we know that $\forall v', v =_\beta v', \exists u_2' : u_2 \xrightarrow{c?v'} u_2'$ and $u_1' \approx_\beta u_2'$. Then we can deduce with R_7 that $u_2 + u \stackrel{c?v'}{\Longrightarrow} u_2'$. With $u_1' \approx_\beta u_2'$ the conditions of Definition 12 are satisfied.

- $u_1 \xrightarrow{\omega} u_1'$, $\omega \neq c?v$ and $u_{new} = u_1'$, derived with R_7. With $u_1 \simeq_\beta u_2$ we know that $\exists \omega', \omega =_\beta \omega', \exists u_2' : u_2 \xRightarrow{\omega'} u_2'$ and $u_1' \approx_\beta u_2'$. Then we can deduce with R_7 that $u_2 + u \xRightarrow{\omega'} u_2'$. With $u_1' \approx_\beta u_2'$ the conditions of Definition 12 are satisfied.
- $u \xrightarrow{\omega} u'$ and $u_{new} = u'$, derived with R_8. Similarly, we can derive with R_8 that $u_2 + u \xrightarrow{\omega} u'$ and, therefore, $u_2 + u \xRightarrow{\omega} u'$. Furthermore, $u' \approx_\beta u'$. □

Theorem 15. The axiomatic theory $\mathcal{AX}_{\simeq_\beta}$ is sound with respect to $\simeq_\beta$.

Proof. For each axiom of Figure 5 we define a relation and prove that it satisfies the conditions of Definition 12. For each step $t \xrightarrow{c!v,C} t'$ we assume that $C \cap fc(t) = \emptyset$.

– Axiom (37): $\gamma; \tau = \gamma$.
Let $R = \{(\gamma; \tau, \gamma)\}$ be a relation. We show that it satisfies the conditions of Definition 12. Therefore, we distinguish the following cases:
  - $\gamma; \tau \xrightarrow{\gamma} \mathbf{1}; \tau$, derived with R_9. With $\gamma \xrightarrow{\gamma} \mathbf{1}$ we know that $\gamma \xRightarrow{\gamma} \mathbf{1}$. Furthermore, we know with Proposition 11 that $\mathbf{1}; \tau \approx_\beta \mathbf{1}$.
  - $\gamma \xrightarrow{\gamma} \mathbf{1}$. Then $\gamma; \tau \xrightarrow{\gamma} \mathbf{1}; \tau$, derived with R_9. With Proposition 11 we know that $\mathbf{1}; \tau \approx_\beta \mathbf{1}$. Furthermore, we can derive with R_2, R_{10} that $\mathbf{1}; \tau \xrightarrow{\tau} \mathbf{1}; \mathbf{1}$ and $\mathbf{1}; \mathbf{1} \approx_\beta \mathbf{1}$. Therefore, we have $\gamma; \tau \xRightarrow{\gamma} \mathbf{1}; \mathbf{1}$ and $\mathbf{1}; \mathbf{1} \approx_\beta \mathbf{1}$.

– Axiom (38): $spawn(\tau; t) = \tau; spawn(t)$.
Let $R = \{(spawn(\tau; u), \tau; spawn(u)) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the conditions of Definition 12. Therefore, we distinguish the following cases:
  - We assume $spawn(\tau; u) \xrightarrow{\omega} u'$ and distinguish the following cases:
    * $\omega = \tau$ and $u' = spawn(\mathbf{1}; u)$, derived with R_2, R_9, R_{15}. Similarly, we can derive with R_2, R_9 that $\tau; spawn(u) \xrightarrow{\tau} \mathbf{1}; spawn(u)$. Furthermore, $spawn(\mathbf{1}; u) \approx_\beta \mathbf{1}; spawn(u)$.
    * $\omega = \checkmark$ and $u' = spawn(\tau; u)$, derived with R_{14}. With R_2,R_9 we can derive that $\tau; spawn(u) \xrightarrow{\tau} \mathbf{1}; spawn(u)$; moreover, we know that $\mathbf{1}; spawn(u) \approx_\beta spawn(u)$ and $spawn(\tau; u) \approx_\beta spawn(u)$, derived with Proposition 11. Then we can deduce with R_1, R_{14}, R_{10} that $\mathbf{1}; spawn(u) \xrightarrow{\checkmark} \mathbf{1}; spawn(u)$ and therefore $\tau; spawn(u) \xRightarrow{\checkmark} \mathbf{1}; spawn(u)$. Furthermore, due to $\mathbf{1}; spawn(u) \approx_\beta spawn(u)$ we can deduce that $spawn(\tau; u) \approx_\beta \mathbf{1}; spawn(u)$.
  - We assume $\tau; spawn(u) \xrightarrow{\omega} u'$. Then we know with R_2, R_9 that $\omega = \tau$ and $u' = \mathbf{1}; spawn(u)$. Similarly, we can derive with R_2, R_9, R_{15} that $spawn(\tau; u) \xrightarrow{\tau} spawn(\mathbf{1}; u)$. Furthermore, we know that
$spawn(\mathbf{1}; u) \approx_\beta \mathbf{1}; spawn(u)$.

– Axiom (40): $E?x; \tau; t = E?x; t$.
Let $R = \{(E?x; \tau; u, E?x; u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the conditions of Definition 12. We assume $red[\![E]\!] = c$.
  - $E?x; \tau; u \xrightarrow{c?v} (\tau; u)\langle v/x\rangle$, derived with R_5. With $\forall \sigma : \tau\sigma = \tau$ we know that $(\tau; u)\langle v/x\rangle = \tau; u\langle v/x\rangle$. Similarly, we can derive with R_5 that $E?x; u \xrightarrow{c?v} u\langle v/x\rangle$ and, therefore, $E?x; uTtransc?vu\langle v/x\rangle$. Furthermore, we know with Proposition 11 that $\tau; u\langle v/x\rangle \approx_\beta u\langle v/x\rangle$.
  - For $E?x; u \xrightarrow{c?v} u\langle v/x\rangle$ analogous.

– Axiom (39): $t + \tau; t = \tau; t$.
Let $R = \{(u + \tau; u, \tau; u) \mid u \in \text{TERM}\}$ be a relation. We show that it satisfies the conditions of Definition 12.
  - We assume $u + \tau; u \xrightarrow{\omega} u_{new}$ and distinguish the followoing cases:
    * $u \xrightarrow{\omega} u'$ and $u_{new} = u'$, derived with R_7. Then we can derive with R_2, R_9 that $\tau; u \xrightarrow{\tau} \mathbf{1}; u$. With $u \sim_\beta \mathbf{1}; u$ and Proposition 13 we know that $u \approx_\beta \mathbf{1}; u$. Furthermore, we can derive with R_{10} that $\mathbf{1}; u \xrightarrow{\omega} \mathbf{1}; u'$ and $u' \approx_\beta \mathbf{1}; u'$. Therefore, $\tau; u \xRightarrow{\gamma} \mathbf{1}; u'$ and $u' \approx_\beta \mathbf{1}; u'$.

  - $\tau; u \xrightarrow{\tau} \mathbf{1}; u$, $\omega = \tau$ and $u_{new} = \mathbf{1}; u$, derived with R_2, R_9, R_8. Similarly, we can deduce with R_2, R_9 that $\tau; u \xrightarrow{\tau} \mathbf{1}; u$ and, therefore, $\tau; u \xRightarrow{\tau} \mathbf{1}; u$. Furthermore, $\mathbf{1}; u \approx_\beta \mathbf{1}; u$.
- We assume $\tau; u \xrightarrow{\omega} u_{new}$. Then we have $\omega = \tau$ and $u_{new} = \mathbf{1}; u$, derived with R_2, R_9. Then we can derive with R_2, R_9, R_8 that $u + \tau; u \xrightarrow{\tau} \mathbf{1}; u$ and, therefore, $u + \tau; u \xRightarrow{\tau} \mathbf{1}; u$. Furthermore, $\mathbf{1}; u \approx_\beta \mathbf{1}; u$. □

Theorem 17. If $\vdash E : T$ and $E \hookrightarrow E'$, then $\vdash E' : T$.

For the proof of this theorem we need the following auxiliary result:

Lemma 24. *If $x : T' \vdash E : T$ and $\vdash v : T'$, then $\vdash E\langle v/x\rangle : T$.*

Proof. Analogous to the proof for Lemma 21. □

Proof. (of Theorem 17) We assume $\vdash E : T$, $E \hookrightarrow E'$ and prove the theorem for the rules of the reduction semantics in Figure 8.

R_{17} Then $E = \mathit{if}\ E_1\ \mathit{then}\ E_2\ \mathit{else}\ E_3$, $E_1 \hookrightarrow E_1'$ and $E' = \mathit{if}\ E_1'\ \mathit{then}\ E_2\ \mathit{else}\ E_3$. With T_{20} we know that $\vdash E_1 : \mathit{bool}$, $\vdash E_2 : T$ and $\vdash E_3 : T$. With the induction hypothesis we know that $\vdash E_1' : \mathit{bool}$. Then we can deduce with T_{20} that $\vdash \mathit{if}\ E_1'\ \mathit{then}\ E_2\ \mathit{else}\ E_3 : T$.

R_{18} Then $E = \mathit{if}\ \mathit{true}\ \mathit{then}\ E_1\ \mathit{else}\ E_2$ and $E' = E_1$. With T_{20} we know that $\vdash E_1 : T$.

R_{19} Then $E = \mathit{if}\ \mathit{false}\ \mathit{then}\ E_1\ \mathit{else}\ E_2$ and $E' = E_2$. With T_{20} we know that $\vdash E_2 : T$. With the induction hypothesis we can deduce that $\vdash E_1' : T\ \mathit{list}$ and, therefore, $\vdash \mathit{head}\ E_1' : T$.

R_{20} Then $E = (V_1, ..., V_{i-1}, E_i, ..., E_n)$ and $E' = (V_1, ..., V_{i-1}, E_i', ..., E_n)$. With T_{21} we know that $\exists T_i :\vdash E_i : T_i$. Then we can deduce with the induction hypothesis that $\vdash E_i' : T_i$ and, therefore, $\vdash E' : T$

R_{21} Then $E = \#i\ (V_1, ..., V_n)$ and $E' = V_i$. With T_{21}, T_{22} we know that $\exists T_1, ..., T_n : T = T_1 \times ... \times T_n$, $\vdash V_j : T_j$ for all $1 \leq j \leq n$ and $\vdash \#i\ (V_1, ..., V_n) : T_i$. Furthermore, we know that $\vdash V_i : T_i$.

R_{22} Then $E = \mathit{cons}\ E_1\ E_2$, $E_1 \hookrightarrow E_1'$ and $E' = \mathit{cons}\ E_1'\ E_2$. With T_{24} we know that $\vdash \mathit{cons}\ E_1\ E_2 : T\ \mathit{list}$ and $\vdash E_1 : T$. With the induction hypothesis we can deduce that $\vdash E_1' : T$ and, therefore, $\vdash E' : T\ \mathit{list}$.

R_{23} Then $E = \mathit{cons}\ V\ E_2$, $E_2 \hookrightarrow \mathit{cons}\ V\ E_2'$ and $E' = \mathit{cons}\ V\ E_2'$. With T_{24} we know that $\vdash E : T\ \mathit{list}$ and $\vdash E_2 : T\ \mathit{list}$. With the induction hypothesis we can deduce that $\vdash E_2' : T\ \mathit{list}$ and, therefore, $\vdash E' : T\ \mathit{list}$.

R_{24} Then $E = \mathit{head}\ E_1$, $E_1 \hookrightarrow E_1'$ and $E' = \mathit{head}\ E_1'$. With T_{25} we know that $\vdash \mathit{head}\ E_1 : T$ and $\vdash E_1 : T\ \mathit{list}$.

R_{25} Then $E = \mathit{head}\ (\mathit{cons}\ V_1\ V_2)$ and $E' = V_1$. With T_{25}, T_{24} we know that $\vdash E : T$, $\vdash \mathit{cons}\ V_1\ V_2 : T\ \mathit{list}$ and $\vdash x : T$. Therefore, we know $\vdash E' : T$.

R_{26} Then $E = \mathit{tail}\ E_1$, $E_1 \hookrightarrow E_1'$ and $E' = \mathit{tail}\ E_1'$. With T_{26} we know that $\vdash E : T\ \mathit{list}$ and $\vdash E_1 : T\ \mathit{list}$. With the induction hypothesis we can deduce that $\vdash E_1' : T\ \mathit{list}$ and, therefore, $\vdash E' : T\ \mathit{list}$.

R_{27} Then $E = \mathit{tail}\ (\mathit{cons}\ V_1\ V_2)$ and $E' = V_2$. With T_{26} we know that $\vdash E : T\ \mathit{list}$, $\vdash \mathit{cons}\ V_1\ V_2 : T\ \mathit{list}$ and $\vdash V_2 : T\ \mathit{list}$. Therefore, we know that $\vdash E' : T\ \mathit{list}$.

R_{28} Then $E = E_1 E_2$, $E_1 \hookrightarrow E_1'$ and $E' = E_1' E_2$. With T_{28} we know that $\vdash E_1 : T' \to T$ and $\vdash E_2 : T'$. With the induction hypothesis we can deduce that $\vdash E_1' : T' \to T$. Therefore, we know with T_{28} that $\vdash E_1'\ E_2 : T$.

R_{29} Then $E = V\ F$, $F \hookrightarrow F'$ and $E' = V\ F'$. With T_{28} we know that $\vdash V : T' \to T$ and $\vdash F : T'$. Then we can deduce with the induction hypothesis that $\vdash F' : T'$ and, therefore, $\vdash V\ F' : T$.

R_{30} Then $E = (\lambda x.F)V$ and $E' = F\langle v/x\rangle$. With T_{28} we can derive that $x : T' \vdash F : T' \to T$ and $\vdash V : T'$. Then we can deduce with Lemma 24 and T_1 that $\vdash F\langle V/x\rangle : T$.

R_{31} Then $E = (\mathit{let}\ x = E_1\ \mathit{in}\ E_2)$, $E_1 \hookrightarrow E_1'$ and $E' = (\mathit{let}\ x = E_1'\ \mathit{in}\ E_2)$. With T_{29} we know that $\vdash E_1 : T'$ and $x : T' \vdash E_2 : T$. With the induction hypothesis and T_{29} we can deduce that $\vdash E_1' : T'$ and, therefore, $\vdash \mathit{let}\ x = E_1'\ \mathit{in}\ E_2 : T$.

$\mathbf{R}_{32}$ Then $E = (let\ x = V\ in\ F)$ and $E' = F\langle v/x\rangle$. With T_{29} we know that $\vdash V : T'$ and $x : T' \vdash F : T$. Then we can derive with Lemma 24 and T_1 that $\vdash F\langle v/x\rangle : T$.

$\mathbf{R}_{33}$ Then $E = Y\ E_1\ E_2$, $E_1 \hookrightarrow E_1'$ and $E' = Y\ E_1'\ E_2$. With T_{30} we know that $\vdash E : T'$, $\vdash E_1 : (T \to T') \to T \to T'$ and $\vdash E_2 : T$. With the induction hypothesis we can deduce that $\vdash E_1' : (T \to T') \to T \to T'$ and, therefore, $\vdash E' : T'$.

$\mathbf{R}_{34}$ Then $E = Y\ V\ E_2$, $E_2 \hookrightarrow E_2'$ and $E' = Y\ V\ E_2'$. With T_{30} we know that $\vdash E : T'$, $\vdash V : (T \to T') \to T \to T'$ and $\vdash E_2 : T$. With the induction hypothesis we know that $\vdash E_2' : T$ and, therefore, $\vdash E' : T'$.

$\mathbf{R}_{35}$ Then $E = Y\ V_1\ V_2$ and $E' = (V_1\ (Y\ V_1))\ V_2$. With T_{30} we know that $\vdash E : T'$, $\vdash V_1 : (T \to T') \to T \to T'$ and $\vdash V_2 : T$. Then we can deduce that $\vdash Y\ V_1 : T \to T'$ and $\vdash V_1\ (Y\ V_1) : T \to T'$. Therefore, we know that $\vdash E' : T'$. □

Theorem 20.

The systems

$$\mathbf{ch}\ rc.\ spawn(RPC\_Simulate(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$$

and

$$\mathbf{ch}\ rc.\ spawn(RPC\_Server(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$$

have the same behaviour w.r.t. weak congruence.

Proof. For the proof, we assume that the memory cell has been proved to work correctly, meaning that if a tuple (adr, v, c) has been sent on *read* or *write*, the result of the memory access is delivered on the channel c.

First, we apply the equations of Figure 4 and Figure 5 to the system without the RPC-component. It is described by a term, in which first an instance of $RPC\_Simulate$ is created. Then a local channel *result* is declared, which is used as a return channel during the write access to the memory component. The equations used in the transformation steps are given as comments.

$$\begin{aligned}
&\mathbf{ch}\ rc.\ spawn(RPC_Simulate(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}\\
&\qquad\qquad \text{// Eq. (25)}\\
=\ &\mathbf{ch}\ rc.\\
&\quad spawn(\tau; rc?req; \{proc_ch \leftarrow \#1\ req, adr \leftarrow \#2\ req, value \leftarrow \#3\ req, return_ch \leftarrow \#4\ req\};\\
&\qquad spawn(proc_ch!(adr, value, return_ch)); RPC_Simulate(rc));\\
&\quad \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}\\
&\qquad\qquad \text{// Eq. (24)}\\
=\ &\mathbf{ch}\ rc, result.\\
&\qquad spawn(\tau; rc?req; \{proc_ch \leftarrow \#1\ req, adr \leftarrow \#2\ req, value \leftarrow \#3\ req, return_ch \leftarrow \#4\ req\};\\
&\qquad\qquad spawn(proc_ch!(adr, value, return_ch)); RPC_Simulate(rc));\\
&\qquad rc!(write, adr, v, result); result?x; \mathbf{1}\\
&\qquad\qquad \text{// Eq. (36)}\\
=\ &\mathbf{ch}\ rc, result.\\
&\qquad \tau; spawn(rc?req; \{proc_ch \leftarrow \#1\ req, adr \leftarrow \#2\ req, value \leftarrow \#3\ req, return_ch \leftarrow \#4\ req\};\\
&\qquad\qquad spawn(proc_ch!(adr, value, return_ch)); RPC_Simulate(rc));\\
&\qquad\quad rc!(write, adr, v, result); result?x; \mathbf{1}\\
&\quad + rc!(write, adr, v, result);\\
&\qquad spawn(\tau; rc?req; \{proc_ch \leftarrow \#1\ req, adr \leftarrow \#2\ req, value \leftarrow \#3\ req, return_ch \leftarrow \#4\ req\};\\
&\qquad\qquad spawn(proc_ch!(adr, value, return_ch)); RPC_Simulate(rc));\\
&\qquad result?x; \mathbf{1}\\
&\qquad\qquad \text{// Eq. (36), } rc \text{ restricted}\\
=\ &\mathbf{ch}\ rc, result.\ \tau; \tau; spawn(spawn(write!(adr, v, result)); RPC_Simulate(rc)); result?x; \mathbf{1}\\
&\qquad\qquad \text{// Eq. (28)}
\end{aligned}$$

$= \mathbf{ch}\ rc, result.\ \tau; \tau; spawn(write!(adr, v, result)); spawn(RPC\_Simulate(rc)); result?x; \mathbf{1}$
// Eq. (37)
$= \mathbf{ch}\ rc, result.\ \tau; spawn(write!(adr, v, result)); spawn(RPC\_Simulate(rc)); result?x; \mathbf{1}$
// Eq. (27)
$= \mathbf{ch}\ rc, result.\ \tau; spawn(RPC\_Simulate(rc)); spawn(write!(adr, v, result)); result?x; \mathbf{1}$
// Eq. (4), (36), *result* restricted
$= \mathbf{ch}\ rc, result.\ \tau; spawn(RPC\_Simulate(rc)); write!(adr, v, result); spawn(\mathbf{1}); result?x; \mathbf{1}$
// $spawn(\mathbf{1}) = \mathbf{1}$, Eq. (3)
$= \mathbf{ch}\ rc, result.\ \tau; spawn(RPC\_Simulate(rc)); write!(adr, v, result); result?x; \mathbf{1}$

Now we apply the axiomatic theory to the system including the RPC component.

$\mathbf{ch}\ rc.\ spawn(RPC\_Server(rc)); \mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$
// Eq. (25)
$= \mathbf{ch}\ rc.$
$spawn(\tau; rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$\mathbf{ch}\ result.\ rc!(write, adr, v, result); result?x; \mathbf{1}$
// Eq. (24)
$= \mathbf{ch}\ rc, result.$
$spawn(\tau; rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$rc!(write, adr, v, result); result?x; \mathbf{1}$
// Eq. (36)
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$rc!(write, adr, v, result); result?x; \mathbf{1}$
$+ rc!(write, adr, v, result); spawn(\tau; rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$result?x; \mathbf{1}$
// *rc* restricted
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$rc!(write, adr, v, result); result?x; \mathbf{1}$
// Eq. (36)
$= \mathbf{ch}\ rc, result.\ \tau;$
$(rc?req; spawn(spawn(RPC\_Client(req)); RPC\_Server(rc));$
$rc!(write, adr, v, result); result?x; \mathbf{1}$
$+ rc!(write, adr, v, result); spawn(rc?req; spawn(RPC\_Client(req)); RPC\_Server(rc));$
$result?x; \mathbf{1}$
$+ \tau; spawn(spawn(RPC\_Client((write, adr, v, result))); RPC\_Server(rc)); result?x; \mathbf{1})$
// *rc* restricted
$= \mathbf{ch}\ rc, result.$
$\tau; \tau; spawn(spawn(RPC\_Client((write, adr, v, result))); RPC\_Server(rc)); result?x; \mathbf{1}$
// Eq. (37)
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(spawn(RPC\_Client((write, adr, v, result))); RPC\_Server(rc)); result?x; \mathbf{1}$
// Eq. (28)
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(RPC\_Client((write, adr, v, result))); spawn(RPC\_Server(rc)); result?x; \mathbf{1}$
// Eq. (25)
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(\tau; \mathbf{ch}\ local\_ch.\ write!(adr, v, local\_ch); local\_ch?x; result!x);$
$spawn(RPC\_Server(rc)); result?x; \mathbf{1}$
// Eq. (38), (37)
$= \mathbf{ch}\ rc, result.$
$\tau; spawn(\mathbf{ch}\ local\_ch.\ write!(adr, v, local\_ch); local\_ch?x; result!x);$

$$
\begin{aligned}
&\quad\quad spawn(RPC\_Server(rc)); result?x; \mathbf{1} \\
&\quad\quad \text{// Eq. (27)} \\
&= \mathbf{ch}\ rc, result. \\
&\quad\quad \tau; spawn(RPC\_Server(rc)) \\
&\quad\quad\quad\quad spawn(\mathbf{ch}\ local\_ch.\ write!(adr, v, local\_ch); local\_ch?x; result!x); result?x; \mathbf{1} \\
&\quad\quad\quad\quad \text{// Eq. (24),(36), } rc \text{ restricted} \\
&= \mathbf{ch}\ rc, result, local\_ch. \\
&\quad\quad \tau; spawn(RPC\_Server(rc)); \\
&\quad\quad\quad\quad write!(adr, v, local\_ch); spawn(local\_ch?x; result!x); result?x; \mathbf{1} \\
&\quad\quad\quad\quad \text{// Eq. (36), } rc \text{ restricted} \\
&= \mathbf{ch}\ rc, result, local\_ch. \\
&\quad\quad \tau; spawn(RPC\_Server(rc)); \\
&\quad\quad\quad\quad write!(adr, v, local\_ch); local\_ch?x; spawn(result!x); result?x; \mathbf{1} \\
&\quad\quad\quad\quad \text{// Eq. (36), } rc \text{ restricted} \\
&= \mathbf{ch}\ rc, result, local\_ch.\ \tau; spawn(RPC\_Server(rc)); write!(adr, v, local\_ch); local\_ch?x; \tau; \mathbf{1} \\
&\quad\quad\quad\quad \text{// Eq. (38), (40), (16)} \\
&= \mathbf{ch}\ rc, local\_ch.\ \tau; spawn(RPC\_Server(rc)); write!(adr, v, local\_ch); local\_ch?x; \mathbf{1} \\
&\quad\quad\quad\quad \text{// } \alpha\text{-conversion} \\
&= \mathbf{ch}\ rc, result.\ \tau; spawn(RPC\_Server(rc)); write!(adr, v, result); result?x; \mathbf{1}
\end{aligned}
$$

As it can be seen, both systems can be transformed into similar terms which only differ in the call of *RPC_Simulate* and *RPC_Server*, resp. These processes cannot perform any further actions, because both expect input on the restricted channel rc. Therefore, we have shown that both systems show the same behaviour w.r.t. weak congruence. □

Technische Universität Braunschweig
Informatik-Berichte ab Nr. 96-02

96-02	M. Goldapp, U. Grottker, G. Snelting	Validierung softwaregesteuerter Meßsysteme durch Program Slicing und Constraint Solving
96-03	C. Lindig, G. Snelting	Modularization of Legacy Code Based on Mathematical Concept Analysis
96-04	J. Adámek, J. Koslowski, V. Pollara, W. Struckmann	Workshop Domains II (Proceedings)
96-05	F.-J. Grosch	A Syntactic Approach to Structure Generativity
96-06	E. H. A. Gerbracht, W. Struckmann	Zur Diskussion elementarer Funktionen aus algorithmischer Sicht
96-07	H.-D. Ehrich	Object Specification
97-01	A. Zeller	Versioning Software Systems through Concept Descriptions
97-02	K. Neumann, R. Müller	Implementierung von Assertions durch Oracle7-Trigger
97-03	G. Denker, P. Hartel	TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics
97-04	F.-J. Grosch	M - eine typisierte, funktionale Sprache für das Programmieren-im-Grossen
97-05	J. Küster Filipe	Putting Synchronous and Asynchronous Object Modules together: an Event-Based Model for Concurrent Composition
97-06	J. Küster Filipe	A categorical Hiding Mechanism for Concurrent Object Systems
97-07	G. Snelting, U. Grottker, M. Goldapp	VALSOFT Abschlussbericht
98-01	J. Krinke, G. Snelting	Validation of Measurement Software as an application of Slicing and Constraint Solving
98-02	S. Petri, M. Bolz, H. Langendörfer	Transparent Migration and Rollback for Unmodified Applications in Workstation Clusters
98-03	M. Cohrs, E. H. A. Gerbracht, W. Struckmann	DISKUS - Ein Programm zur symbolischen Diskussion reeller elementarer Funktionen
98-04	C. Lindig	Analyse von Softwarevarianten
98-05	Gregor Snelting, Frank Tip	Reengineering Class Hierarchies Using Concept Analysis
98-06	Juliana Küster Filipe	On a Distributed Temporal Logic for Modular Object Systems
98-07	J. Schönwälder, M. Bolz, S. Mertens, J. Quittek, A. Kind, J. Nicklisch	SMX - Script MIB Extensibility Protocol Version 1.0
98-08	C. Heimann, S. Lauterbach, T. Förster	Entwurf und Implementierung eines verteilten Ansatzes zur Lösung langrechnender Optimierungsprobleme aus dem Bereich der Ingenieurwissenschaften
99-01	A. Zeller	Yesterday, my program worked. Today, it does not. Why?
99-02	P. Niebert	A Temporal Logic for the Specification and Verification of Distributed Behaviour
99-03	S. Eckstein, K. Neumann	Konzeptioneller Entwurf mit der Unified Modeling Language
99-04	T. Gehrke, A. Rensink	A Mobile Calculus with Data