On Generic Context Lemmas for Lambda Calculi with Sharing

Manfred Schmidt-Schauß and David Sabel

Institut für Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt, Germany
schauss@cs.uni-frankfurt.de

Technical Report Frank-27
Revised Version[1]

June 19, 2008

Abstract. This paper proves several generic variants of context lemmas and thus contributes to improving the tools for observational semantics of deterministic and non-deterministic higher-order calculi that use a small-step reduction semantics. The generic (sharing) context lemmas are provided for may- as well as two variants of must-convergence, which hold in a broad class of extended process- and extended lambda calculi, if the calculi satisfy certain natural conditions. As a guide-line, the proofs of the context lemmas are valid in call-by-need calculi, in call-by-value calculi if substitution is restricted to variable-by-variable and in process calculi like variants of the π-calculus. For calculi employing beta-reduction using a call-by-name or call-by-value strategy or similar reduction rules, some iu-variants of ciu-theorems are obtained from our context lemmas. Our results reestablish several context lemmas already proved in the literature, and also provide some new context lemmas as well as some new variants of the ciu-theorem. To make the results widely applicable, we use a higher-order abstract syntax that allows untyped calculi as well as certain simple typing schemes. The approach may lead to a unifying view of higher-order calculi, reduction, and observational equality.

Keywords: lambda calculus, observational semantics, context lemma, functional programming languages

1 Introduction

A workable semantics is indispensable for every formal modeling language, in particular for all kinds of programming languages and process calculi. This paper

[1] This document is a revised and extended version of an earlier version originally published on the web in August 2007.

will make a contribution to the tools, in particular so-called context-lemmas, which support operational reasoning about semantical properties of higher-order functional programming languages, process calculi and extended lambda-calculi on the basis of an observational semantics. A semantics is very useful to obtain safe knowledge about the evaluation and optimizations of programs, correctness of program transformations, and correctness of translations into other calculi.
For various higher-order calculi a widely used observational semantics is contextual equivalence based on a (small-step) reduction semantics in the style of [Mor68], i.e. two expressions are equal if their termination behavior is always the same when they are plugged into an arbitrary program context. We assume that a calculus is given consisting of a language of terms, a small step reduction relation "$\to$" on terms, and a set of answer terms. A term t is called may-convergent if there exists a finite sequence of $\to$-reductions starting with t and reaching an answer. Usually answers are weak head normal forms for call-by-need and call-by-name calculi, weak normal forms for call-by-value calculi, and irreducible (or successful) processes in process calculi. For non-deterministic calculi, contextual equivalence must be based on the conjunction of two termination behaviors (see e.g. [Ong93]): may-convergence and must-convergence, where the latter takes all reduction possibilities into account. There are two different definitions of must-convergence in the literature:

1. iff every term t' reachable from t by a sequence of $\to$-reductions is may-convergent.
2. iff every maximal sequence of reductions starting with t ends in an answer, in particular, there are no infinite reductions. We will call this form of must-convergence also total must-convergence.

The first definition of must-convergence also includes terms that may evaluate infinitely but the chance of finding an answer is never lost. These terms are called weakly divergent in [CHS05]. Note that a similar combination of may- and must-convergence is also known from the use of convex powerdomains in domain-theoretic models (see [Plo76]).
In this paper we will consider several variants of contextual approximations and equivalences based on may-, must- and total must-convergence. Usually, a first step and a strong tool for further proof techniques is to prove a context lemma that reduces the test for convergence (may- and/or must-) to a subclass of contexts, the reduction contexts (also called evaluation contexts), instead of all contexts. This technique dates back to [Mil77] for showing full abstractness of denotational models of lambda-calculi.
Contextual (may-convergence) approximation $s \leq_\downarrow t$ means that for all contexts C: $C[s] \downarrow \implies C[t] \downarrow$. The context lemma for may-convergence $\downarrow$ tells us that contextual approximation $\leq_\downarrow$ can be tested by observing may-convergence in reduction contexts only, i.e., if for all reduction contexts R: $R[s] \downarrow \implies R[t] \downarrow$, then $s \leq_\downarrow t$. For total must-convergence, the claim is similar, whereas for must-convergence, a conjunction with may-convergence is required (see Theorem 5.12).
We formulate natural conditions on extended (sharing) lambda-calculi and process calculi and their reduction semantics, and then prove generic context lemmas

for the three types of convergencies for calculi satisfying these conditions. An informal account of our results is as follows: we assume that a calculus in a higher-order abstract syntax is given together with a small-step reduction relation, and a set of answers; also an algorithm to determine reduction positions is given. The (sharing) assumptions are as follows:

1. The set of reduction positions of a term (or context) is determined top-down, and does not depend on non-reduction positions.
2. The property of being an answer-term does not depend on non-reduction positions.
3. The small-step reduction relation has rather limited abilities to modify subterms at non-reduction positions: it is permitted to remove, transport or duplicate them; also to apply renaming of bound variables. It is not permitted to modify subterms at non-reduction positions. In strongly sharing calculi, the reduction rules do not modify non-reduction positions, and in weakly sharing calculi, reduction rules may modify non-reduction positions via a restricted form of variable-variable substitutions.
4. All properties are invariant under renaming of bound variables, and also under permutation of free variables, as long as no type conditions are violated and there is no capture of free variables.

The obtained results are six context lemmas for the combinations of strongly and weakly sharing and the three types of convergencies: for strongly sharing calculi, a context lemma allows to restrict the observation of the convergence to reduction contexts instead of all contexts, where reduction contexts are exactly the contexts where the hole is a reduction position. In the case of weakly sharing calculi, we have to observe the behavior of $R[\sigma(s)]$, where $R[]$ is a reduction context, and σ a perhaps non-injective substitution replacing variables by variables without a variable capture.

We also obtain generic variants of (c)iu-theorems for call-by-value and call-by-name calculi where we have to observe the behavior of $R[\sigma(s)]$, where $R[]$ is a reduction context, and σ a substitution replacing variables by expressions, where in the case of call-by-value-like calculi, σ may be restricted to be a value-substitution. Our (c)iu-theorems are obtained for the three types of convergencies in the case that the sharing Reduction Assumption does not hold, but the Non-Sharing Assumption holds (see Section 6). The Non-Sharing Assumption roughly requires that reductions are similar to beta-reduction. The assumptions ensure that the effects of substitutions (caused by e.g. beta-reduction) can be translated using let-environments into a sharing calculus. In addition it is required that there is no letrec-construct and that reduction positions are not in the scope of variable binders. The (c)iu-theorems are obtained using the translation technique from [SSNSS08].

Our proof technique for obtaining the sharing context lemmas works for process calculi like the π-calculus, and program calculi with sharing variants of beta-reduction. Since programming languages or their respective abstract machines almost always exploit sharing mechanisms, our results become applicable if the modeling calculi also take sharing into account. We consider two forms of sharing

lambda calculi: strongly sharing and weakly sharing calculi. In strongly sharing calculi, the (normal-order) reduction may only modify non-reduction positions through renamings of bound variables. E.g. the full beta-rule $(\lambda x.s)\ t \to s[t/x]$ violates our (sharing) assumptions, since there may be non-reduction occurrences of x in s that are replaced by the beta-rule. The restricted beta-rule $(\lambda x.s)\ y \to s[y/x]$ may be allowed in weakly sharing calculi, but only if the argument position in applications is syntactically restricted to be a variable. In this case, there may be a substitution of variables by variables. The rules $(\lambda x.s)\ t \to (\texttt{let}\ x = t\ \texttt{in}\ s)$ and $(\texttt{let}\ x = v\ \texttt{in}\ R[x]) \to (\texttt{let}\ x = v\ \texttt{in}\ R[v])$ are the sharing variants of beta-reduction and permitted in strongly sharing calculi. The corresponding rules in explicit substitution calculi (see [ACCL91]) are compatible, though in connection with non-deterministic operators the set of rules has to be adapted. Similar considerations hold for other rules like case-rules. Examples for calculi, where the context-lemma for may-convergence is immediately applicable are the deterministic calculi in [AFM$^+$95,AF97,MOW98,AS98,SS07]. Non-deterministic calculi where also the must-context lemmas are applicable, are in [KSS98,Man05,SSS08,NSSSS07], the latter is the calculus in [NSS06] with some adaptations. The context lemmas also hold in process calculi like variants of the π-calculus (see [Mil99,SW01]) and the join-calculus (see [FG96,Lan96]), which from our point of view are weakly sharing, since a full replacement of names by names is performed by reduction rules.

The context lemma is an important tool for further investigations into correctness of program transformations and optimizations, for example, the diagram methods in [SSSS08,SSS08,NSSSS07] demonstrate their strength only if the context lemma holds. There is no context lemma used in [KSS98], which severely complicates the equivalence proofs that use diagrams.

There is also related work on context lemmas for calculi not satisfying our (sharing) conditions. For call-by-value languages with beta-reduction, there is another form of a context lemma, the so-called ciu-theorem (for all closed instantiation of uses), which appeared in [MT91,FH92,MST96,PS98,Las98], and which also holds for a class of languages, and was even formally checked by an automated reasoner (see [FM01,FM03]). For PCF-like languages, also with full beta-reduction, there is also a context lemma proved for a class of languages extending PCF (see [JM97]).

Another related generic tool is bisimilarity for extended lambda-calculi (see [How89,How96]), and for typed languages (see [Gor99]). An extension for calculi with sharing w.r.t. may-convergence is done for a non-deterministic calculus in [Man05] and for a class of calculi in [MSS06].

The structure of this paper is as follows. After presenting the abstract syntax for higher-order calculi (section 2), in section 3 the assumptions on the calculi are presented and discussed. Section 4 presents the different convergence relations and contextual approximations, section 5 contains the proofs of the various generic context lemmas. In section 6 we show how the (c)iu-theorem for non-sharing higher-order calculi can be obtained from the context lemma for sharing, and the final section 7 contains a recipe (Subsection 7.1) for using the results

of this paper and a discussion on the range of calculi where the instances of the generic context lemmas hold.

2 Abstract Syntax and Language

In the following we provide generic mechanisms to describe the language of expressions, the renamings and the reduction relation of a calculus CALC. For the generic formulation of the language we use higher-order abstract syntax, (see e.g. [How89,How96]), which is extended by a system of simple types. The construction of terms of the language requires variables, operators (i.e. symbols with arity), and variable-binding primitives. We allow the extension by a recursive `letrec` which is used as an extra operator with its own binding rules. We also require a distinction of terms as values and non-values, mainly for the connection to the ciu-theorem in Section 6 in the case of call-by-value calculi. The main purpose of the types is to allow different syntactic categories in the respective languages, for example, channel names and processes, or lambda-expressions and processes, but also enables to model untyped calculi. It may also be used to model forms of simple typing.

Definition 2.1. *A* signature $\mathcal{L}$ *of a* higher order computation language *is a 6-tuple* $(O, VO, T_0, \alpha, \beta, \beta_\tau)$ *where*

- O *is a (possibly infinite) set of* operators*, which may contain* `letrec`*,*
- VO *is a set of value-operators with* $VO \subset O$*, and* `letrec` $\notin VO$*,*
- T_0 *is the set of* basic types*, which defines the set of types* $\mathcal{TYP}$ *inductively as* $T_0 \subseteq \mathcal{TYP}$ *and* $t_1 \to t_2 \in \mathcal{TYP}$ *if* $t_1, t_2 \in \mathcal{TYP}$[2]*.*
- $\alpha : (O \setminus \{\texttt{letrec}\}) \to \mathbb{N}_0$ *defines the* arity *for every operator except for* `letrec`*.*
- *For every operator* $f \in O \setminus \{\texttt{letrec}\}$*,* $\beta(f)$ *is an* $\alpha(f)$*-tuple with components in* $\mathbb{N} \cup \{\text{“}V\text{”}, \text{“}W\text{”}, \text{‘}T\text{”}\}$*, indicating the number of possible variables that may be bound at the corresponding argument position, or that there are no binders and that only variables (“V”) or that only values (“W”), or that any term (“T”) is permitted.*
  For operators $f \in VO$*,* $\beta(f)$ *must not contain “V”,“W”-components.*
- *Let* $f \in O \setminus \{\texttt{letrec}\}$ *with* $\alpha(f) = n$ *and* $\beta(f) = (b_1, \ldots, b_n)$*. Then* f *has a type* $\beta_\tau(f) \in \mathcal{TYP}$ *satisfying the following conditions:*
  - $\beta_\tau(f) = \tau_1 \to \tau_2 \to \ldots \to \tau_{n+1}$
  - *if* $b_i \in \mathbb{N}$*, then* τ_i *must be of the form* $\tau_{i,1} \to \tau_{i,2} \to \ldots \to \tau_{i,b_i+1}$

Note that we do not insist on τ_{n+1} and τ_{i,b_i+1} being base types in the definition of $\beta_\tau(f)$.
Given a signature $\mathcal{L}$ the values and terms of the higher order computation language are defined as follows, where we assume that there is a subset $\mathcal{TYP}_V \subseteq \mathcal{TYP}$, such that for every $\tau \in \mathcal{TYP}_V$, there is an infinite set of variables V_τ of type τ.

[2] We use the convention of right-association: $\tau_1 \to \tau_2 \to \tau_3$ means $\tau_1 \to (\tau_2 \to \tau_3)$

Definition 2.2. *Let $\mathcal{L} = (O, VO, T_0, \alpha, \beta, \beta_\tau)$ be a signature of a higher order computation language, then* values *and* terms $\mathcal{T}(\mathcal{L})$ *are inductively defined as follows, where the definition is mutually recursive:*
Terms *are defined as follows:*

- *Every $x \in V_\tau$ for $\tau \in \mathcal{TYP}_V$ is a term of type τ.*
- *If $f \in O \setminus \{\texttt{letrec}\}$ with $\alpha(f) = 0$, then f is a term with type $\beta_\tau(f)$.*
- *If $f \in O \setminus \{\texttt{letrec}\}$, then $f(a_1, \ldots, a_n)$ is a term provided that $n = \alpha(f) \geq 1$, and for every $i = 1, \ldots, n$ the following holds:*
  - *if $\beta(f)_i =$ "V", then a_i is a variable,*
  - *if $\beta(f)_i =$ "W", then a_i is a value,*
  - *if $\beta(f)_i =$ "T", then a_i is a term, and*
  - *if $\beta(f)_i = m \in \mathbb{N}$, then a_i is of the form $x_1, \ldots, x_m \,.\, t_i$, where $x_1, \ldots, x_m$ are different variables, and t_i is a term[3].*

  The typing must be as follows: if f has type $\tau_1 \to \ldots \to \tau_n \to \tau_{n+1}$, then a_i has type τ_i for all i, and the operands $a_i = x_1, \ldots, x_m \,.\, t_i$ are defined to have type $\tau_i = (\tau_{i,1} \to \ldots \tau_{i,m} \to \tau_{i,m+1})$, where x_j has type $\tau_{i,j}$ for all j and t_i has type $\tau_{i,m+1}$.
- *If $\texttt{letrec} \in O$, $n \geq 0$, $x_1, \ldots, x_n$ are different variables, and if $t_1, \ldots, t_n, s$ are terms, then $(\texttt{letrec}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ s)$ is a term of type τ which is the type of s, where for all i: the terms x_i, t_i must be equally typed.*

Values *are defined as follows:*

- *A variable is also a value,*
- *If $f \in VO$ with $\alpha(f) = 0$, then f is a value with type $\beta_\tau(f)$.*
- *If $f \in VO$, then $f(a_1, \ldots, a_n)$ is a value, provided that $n = \alpha(f) \geq 1$, and for every $i = 1, \ldots, n$ the following holds: if $\beta(f)_i =$ "T", then a_i is a value (or a variable), and in the other cases of $\beta(f)_i$, the construction restrictions are exactly as for terms.*

As usual, the scope of every variable $x_i, i = 1, \ldots, n$ in $x_1, \ldots, x_n \,.\, t$ is the term t, and the scope of every variable x_i in $(\texttt{letrec}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ s)$ is the set of terms $t_1, \ldots, t_n, s$. Variable occurrences that are in the scope of a binder that binds them, are called bound *occurrences of variables, others are* free occurrences *of variables.*

The set of *free variables* of a term t is denoted as $\mathcal{FV}(t)$. As usual, a term t is *closed* if all of its variables are bound, i.e. $\mathcal{FV}(t) = \emptyset$, otherwise it is called *open*.
Since we have to deal in depth with different kinds of renamings in later sections, we do not assume anything about implicit renamings, though it is known how to correctly rename terms (cf. [Bar84]).
Concerning the typing, we assume in the following that only correctly typed terms are syntactically acceptable, that every correct term has exactly one type, and that modifications, renamings, reductions do not change the type of a term. At places, where it is important, we emphasize the typing, whereas we often do not mention the typing, if it is clear from the context.

[3] The expressions $x_1, \ldots, x_m \,.\, t_i$ are called operands in [How96]

Example 2.3. With $O = \{\lambda, \texttt{letrec}, @, \texttt{Cons}, \texttt{Nil}, \texttt{case}\}$ a language with `let(rec)`, application, abstractions, lists and a `case` is defined. We obtain an untyped lambda-calculus, if *Term* is the only base type and the result type of every operator is just *Term*. The description is $\alpha(\lambda) = 1$, $\beta(\lambda) = (1)$, $\alpha(@) = 2$, $\beta(@) = (\text{“}T\text{”}, \text{“}T\text{”})$, `Cons` is specified like @, $\alpha(\texttt{Nil}) = 0$, and $\alpha(\texttt{case}) = 3, \beta(\texttt{case}) = (\text{“}T\text{”}, 2, \text{“}T\text{”})$. The lambda-term $\lambda x.x$ is represented as $\lambda(x.x)$. A term like
`let x = (Cons x Nil) in case x of (Cons z1 z2) -> y; Nil -> Nil`
would be expressed as $(\texttt{letrec}\ x = \texttt{Cons}(x, \texttt{Nil})\ \texttt{in}\ \texttt{case}(x, z_1 z_2.y, \texttt{Nil}))$.

Example 2.4. The non-recursive construct `let` can be expressed as an operator with $\alpha(\texttt{let}) = 2$, $\beta(\texttt{let}) = (2, \text{“}T\text{”})$, such that $\texttt{let}(x.s, t)$ corresponds to the common $\texttt{let}\ x = t\ \texttt{in}\ s$.

Examples of languages where “V” and/or “W” is necessary, i.e., with the variable restriction are: [MSC99] where arguments of applications are only variables, the language in [NSSSS07], where e.g. the first argument in cell-expressions $(x\ \texttt{c}\ t)$ must be a variable, and the second argument must be a value, and process calculi like the π-calculus, which have argument restrictions for variables. Also, we will use the “W”-restriction in Section 6 to derive the iu-theorem(s).
In the following, we will sometimes use the standard notation, and write e.g. $\lambda x.s$ instead of $\lambda(x.s)$ and $(\texttt{let}\ x = t\ \texttt{in}\ s)$ instead of $\texttt{let}(x.s, t)$.

Example 2.5. The π-calculus with the syntax

$$P, Q ::= (P \mid Q) \mid \nu x.P \mid 0 \mid P + Q \mid x(y_1, \ldots, y_n).P \mid \overline{x}\langle y_1, \ldots, y_n\rangle.P \ \mid\ !P$$

can be represented. We have to use at least two types for an appropriate encoding: “*process*” and “*channel*”, and moreover, we assume that there are variables of type *channel*, but no variables of type *process*. The common scoping policy in $x(y_1, \ldots, y_n).P$ is also easily representable as $\texttt{in}(x, (y_1, \ldots, y_n.P))$, where the first argument of `in` is restricted to variables. The process $\overline{x}\langle y_1, \ldots, y_n\rangle.P$ can be represented as $\texttt{out}_n(x, y_1, \ldots, y_n, P)$ with perhaps different operators, where all arguments but the last one are marked “V” in the arity-tuple.

Remark 2.6. Values have the following specific properties: if v is a value, then $\sigma(v)$ is also a value, provided σ replaces values or variables for variables. Hence values are invariant under renamings. The definition of values implies that there are places within terms (besides the immediate “W”-restricted places), where due to the syntactic restrictions only values are permitted. Our definition of values corresponds exactly to the values as used in call-by-value calculi. The value operators correspond to constructors, and the arguments may only be variables or values or abstractions, i.e. with lambda as top level operator. In the latter case the operand is of the form $x.t$ with an arbitrary term t.

We will use *positions* to address subterms and variables in binders using a slightly extended Dewey decimal notation. The addressing is such that prefixes of addresses of term positions are again term positions. We write $s_{|p}$ for the subterm

of s at position p, and $s(p)$ for the head-symbol of the subterm $s_{|p}$. For example, the term $t = \texttt{Cons}(\lambda(x.@(x, @(y, x)), z)$ has e.g. the following term positions: y is at position 1.1.2.1, x occurs at positions 1.1.1 and 1.1.2.2, and $t(1) = \lambda$ and $t(1.1.1) = x$. The binders are addressed using a "B", such that the binding position of x above is $1.1.B.1$. The positions in $(\texttt{letrec}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ s)$ are as follows: t_i is at position i, s at position $n + 1$, and the bound variable x_i at position $B.i$. For the description of "renamings" of free variables we also assume that there is a virtual binder for free variables at the top of the term.
A *context* C is like a term, where the hole $[\cdot]$ is allowed at a single term-position, i.e. not at a value position or "V"-position, and the hole must be typed according to the abstract syntax. If the type τ of the hole is important, then we denote this as $C[]_\tau$. The expression $C[s]$ denotes the result of plugging in a (correctly typed) term s into C, where captures of variables are permitted. We will also use *multi-contexts* M that may have more holes, which are contexts with multiple, distinguishable holes, where the holes are in left-to right-order, if not mentioned otherwise. If $t_1, \ldots, t_n$ are terms (or values), and M is a multicontext with n holes, then $M[t_1, \ldots, t_n]$ is the term after plugging in the n terms, in left-to-right order.

Definition 2.7 (Distinct Variable Convention). *A term t satisfies the* distinct variable convention (DVC)*, iff all bound variables in t are distinct, and moreover, all bound variables in t are distinct from all the free variables in t.*

2.1 Renamings and Substitutions

We introduce notions around renamings, substitutions and mappings, since we have to separate reduction-steps and renaming-steps later on. Usually, a renaming renames bound variables in a term. For a uniform treatment, we will also use the notion "renaming" for a bijective replacement of free variables. We will use the word "substitution" if we mean a perhaps non-bijective mapping on free variables. In this and the following sections we tacitly assume that replacements of variables by variables is only performed if the variables have equal type.
A *modification* (function) of variables of the term s is described by a finite set S of pairs $(p, x \mapsto y)$, where p is a binding position of x. Modifying a term means to apply all the replacements $x \mapsto y$ to all occurrences of the variable x in the scope of the binder at p, and also to the binder, where all the replacements have to be done "in parallel". Note that also free variables may be subject to modification. E.g. for $s = \lambda x.@(x, (\lambda x.x))$, the set $S_1 = \{(1.B.1, x \mapsto y), (1.2.B.1, x \mapsto z)\}$ represents the modification of s resulting in $\lambda y.@(y, (\lambda z.z))$. A modification of t is called *capture-free*, iff the relation between occurrence of a variable and its binding position remains unchanged, for all variable occurrences in the term t. Note that the modifications are always meant w.r.t. a given term. Modifications do not make sense without mentioning the term to which they are applied. If it is unambiguous, we represent a modification by a set $S = \{(p_1, x_1 \mapsto y_1), \ldots, (p_n, x_n \mapsto y_n)\}$ by omitting the positions as $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$.

Now we define renamings and variable-substitutions as specialized modifications. A *bv-renaming* of s is defined as a modification that is capture-free and renames only bound variables (the virtual binder is not allowed in this case). The relation $s =_\alpha t$ denotes that t can be reached from s by a (perhaps empty) sequence of bv-renamings. An *fvbv-renaming* σ is defined as a capture-free modification that renames free and bound variables of a term s, and that is injective on $\mathcal{FV}(s)$. With $\mathrm{fvp}(\sigma)$ we denote the induced mapping of a fvbv-renaming σ of s on $\mathcal{FV}(s)$. If $\{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$ is the representation of the mapping on variables, then $\{x_1, \ldots, x_n\}$ is the *domain*, and $\{y_1, \ldots, y_n\}$ the *codomain*. A *vvbv-substitution* γ of a term s is a capture-free modification like an fvbv-renaming, but it may be not injective on $\mathcal{FV}(s)$. We also use $\mathrm{fvp}(\gamma)$, which we call in this case a *vv-substitution*, denoted as ν. Given a term s, a set W of variables with $\mathcal{FV}(s) \subseteq W$, a vv-substitution ν on W, and a vvbv-substitution γ. Then we say γ is *compatible with* ν on W, iff $\mathrm{fvp}(\gamma)(x) = \nu(x)$ for all $x \in W$. This notion is also used for fvbv-renamings.

Example 2.8. The term $\lambda y.x$ can be modified into $\lambda x.y$ by an fvbv-renaming (or a vvbv-substitution). However, this cannot be represented as $\sigma\mu_1$ for a bv-renaming μ_1 and a substitution $\sigma = \{x \mapsto y\}$, nor as $\mu_1\sigma$. It can only be represented as $\sigma_1\mu_1\sigma_2$ with two vv-substitutions $\sigma_2 = \{x \mapsto z\}$, and $\sigma_1 = \{z \mapsto y\}$ and a bv-renaming $\mu_1 = \{y \mapsto x\}$.

Lemma 2.9.

- *If* $t_1 \xrightarrow{\sigma_1} t_2 \xrightarrow{\sigma_2} t_3$ *by vvbv-substitutions* σ_1, σ_2*, then the composition* $\sigma_3 = \sigma_2 \circ \sigma_1$ *with* $t_1 \xrightarrow{\sigma_3} t_3$ *is a vvbv-substitution with* $\mathrm{fvp}(\sigma_3)(x) = \mathrm{fvp}(\sigma_2)\mathrm{fvp}(\sigma_1)(x)$ *for all* $x \in \mathcal{FV}(t_1)$*. This also holds for appropriate restrictions to fvbv-renamings and bv-renamings.*
- $s =_\alpha t$ *iff* $s = t$ *or* $s \xrightarrow{\sigma} t$ *by a single bv-renaming* σ*.*
- *If* $s \xrightarrow{\sigma} t$ *by an fvbv-renaming, then the reverse renaming is also capture-free, i.e. a fvbv-renaming. Again this holds also for bv-renamings.*
- *If* s *is a term not satisfying the DVC, then there is a term* s' *satisfying the DVC, and a bv-renaming* σ *with* $\sigma(s) = s'$*. This can be accomplished by renaming bound variables with fresh variables.*
- *If* $s \xrightarrow{\sigma} t$ *is a vvbv-substitution, and* $\rho = \mathrm{fvp}(\sigma)$*, then there are bv-renamings* σ_1, σ_2*, and a vv-substitution* ρ*, such that* $s \xrightarrow{\sigma_1} \xrightarrow{\rho} \xrightarrow{\sigma_2} t$*, i.e.* $\sigma = \sigma_2 \circ \rho \circ \sigma_1$*.*

Proof. Easy computations.

It is interesting to note that terms and vvbv-substitutions form a category with terms as objects and vvbv-substitutions as arrows; the same holds for fvbv-renamings and bv-renamings. We will also use fvbv-renamings and bv-renamings for contexts and multicontexts.

Definition 2.10. *Let* C *be a (one-hole) context. Then* $BP_{\mathrm{hole}}(C)$ *is defined as the set of binder-positions in* C *that have the hole of* C *in their scope,* $BP_{\overline{\mathrm{hole}}}(C)$ *is the complement, i.e. the set of binder-positions that do not have the hole in*

their scope, and $V_{\text{hole}}(C)$ *is the set of variables that are bound by the binders in* $BP_{\text{hole}}(C)$*. For a multi-context* M *with any number of holes, the notation* $BP_{\text{hole}}(M,i)$ *and* $V_{\text{hole}}(M,i)$ *mean the corresponding notions for the* i^{th} *hole.*

It is obvious that all variables bound by binders in $BP_{\text{hole}}(C)$ are different, i.e. there are no binders with equal variables in this set.

Lemma 2.11. *Let* C *be a context,* s *be a term, and* σ *be an fvbv-renaming of* $C[s]$*. Then* σ *can be split into an fvbv-renaming* σ_C *of* C*, and an fvbv-renaming* σ_s *of* s*, where* $\sigma(C[s]) = \sigma_C(C)[\sigma_s(s)]$ *and the mapping* $\text{fvp}(\sigma_s)$ *is injective on* $V_{\text{hole}}(C)$ *as well as on* $\mathcal{FV}(s)$*. Also the mapping induced by* σ *on* $V_{\text{hole}}(C)$ *is injective.*

Example 2.12. Let $C = \lambda y_1.(x\ \lambda z.z\ \lambda y_2.[])$ and $s = (x\ y_2\ \lambda u.u)$. Then $V_{\text{hole}}(C) = \{y_1, y_2\}$. Let σ be the fvbv-renaming of $C[s]$ with $\{x \mapsto x', y_1 \mapsto y_1', y_2 \mapsto y_2', z \mapsto z', u \mapsto u'\}$. Then $\sigma_C = \{x \mapsto x', y_1 \mapsto y_1', y_2 \mapsto y_2', z \mapsto z'\}$, and $\sigma_s = \{x \mapsto x', y_1 \mapsto y_1', y_2 \mapsto y_2', u \mapsto u'\}$. Note that y_2 is a bound variable in C, but free in s.

3 Generic Lambda Calculi with Sharing

We assume that a calculus CALC is given and describe the required notions and properties that are required such that the context lemmas hold.

Definition 3.1. *We assume that for the calculus* CALC *the following is given:*

- *A language of expressions in the higher-order abstract syntax, according to Definition 2.1.*
- *An algorithm* UNWIND*, detecting all the potential reduction positions (which must be term positions, but not "V"-positions and not "W"-positions). We assume that the algorithm has a term or a multi-context as input and non-deterministically produces a sequence of term positions, all of which are reduction positions.*
- *A small-step reduction relation* $\to_0$ *on terms, where* $s \to_0 t_0$ *is defined only for terms* s *satisfying the DVC, but the term* t_0 *is not further restricted. The* small-step reduction $s \to t$ *is then defined as* $s \to_0 t_0 \to_1 t$*, where* $\to_1$ *is a bv-renaming (see Assumption 3.5).*
- *A set of answers* ANS*, which are accepted as successful results of reductions.*
- *A set* VV *of vv-substitutions. There are only two permitted possibilities*
  - VV *is the set of all vv-substitutions. In this case we call* CALC *a* weakly sharing calculus.
  - $VV = \{Id\}$*. In this case we call* CALC *a* strongly sharing calculus.

Note that the restriction $VV = \{Id\}$ for strongly sharing calculi means that these calculi do not make use of vv-substitutions in the definition of equivalence. The calculus in [MS99] presented in Example 3.7, as well as the π-calculus as in Example 3.8 are weakly sharing, whereas the other calculi in the examples

below are strongly sharing. A rule of thumb is that a calculus that allows the replacement of a variable by another one at all occurrences can only be weakly sharing. Note the calculus in [NSSSS07] is a strongly sharing version of a previously published, weakly sharing, calculus in [NSS06].
In the example calculi (see Examples 3.6, 3.7, 3.8, and 3.10) the reduction $\to$ is either the (call-by-need) normal-order reduction, and the answers are the WHNFs, or the reduction is the process-reduction, and answers are processes without any further communication in 3.8, or successful processes in 3.10.
The algorithm UNWIND has a term or a multicontext t as input and (perhaps nondeterministically) produces a sequence of term-positions, starting with $p_1 = \varepsilon$. Given t, the possible sequences $p_1, p_2, \ldots$ produced by UNWIND are called the *valid* UNWIND-*runs* of t. We do not enforce the sequences to be maximal. The following conditions must hold:

*Assumption 3.2 (*UNWIND *Assumptions).*

1. If $p_1, \ldots, p_n$ is a valid UNWIND-run of t, then for all $1 \leq i \leq n$, $p_1, \ldots, p_i$ is also a valid UNWIND-run of t, the set $\{p_1, \ldots, p_n\}$ is a prefix-closed set of positions, and $p_n \notin \{p_1, \ldots, p_{n-1}\}$.
2. If t, t' are terms or multicontexts, $p_1, \ldots, p_n$ is a valid UNWIND-run for t, and for all $i < n$, we have $t(p_i) = t'(p_i)$, then $p_1, \ldots, p_n$ is also a valid UNWIND-run for t'.
3. If t, t' are terms or multicontexts with $t = \sigma(t')$ for an fvbv-renaming σ, and $p_1, \ldots, p_n$ is a valid run for t, then $p_1, \ldots, p_n$ is also a valid UNWIND-run for t'.
4. For weakly sharing calculi, the following additional assumption is required: if t, t' are terms or multicontexts, $\nu \in VV$, γ a vvbv-substitution compatible with ν with $t' = \gamma(t)$, and $p_1, \ldots, p_n$ is a valid UNWIND-run for t, then $p_1, \ldots, p_n$ is also a valid UNWIND-run for t'.

Given a term or a multicontext t. Then the position p of t is called a *reduction position* in t, iff p is contained in some valid UNWIND-run of t. The set of all reduction positions is defined as $\mathrm{RP}(t) = \{p \mid p \text{ is contained in some valid UNWIND-run of } t\}$. Note that every reduction position is a term-position by definition. Since we also apply the formalism to multicontexts, we can speak of reduction positions of multicontexts as well as of terms. A single-hole context $C[]$ is defined as a *a reduction context*, if the hole $[]$ of C is a reduction position in $C[]$: we denote reduction contexts as $R[]$.

Lemma 3.3. *Let M be a multicontext with n holes, and $s_j, j = 1, \ldots, n$ be terms, such that for some i: $M[s_1, \ldots, s_{i-1}, [], s_{i+1}, \ldots s_n]$ is a reduction context. Then there is some $j \in \{1, \ldots, n\}$, such that for all terms $t_k, k = 1, \ldots, n$, $M[t_1, \ldots, t_{j-1}, [], t_{j+1}, \ldots t_n]$ is a reduction context.*

Proof. Let p be the position of the hole in $t := M[s_1, \ldots, s_{i-1}, [], s_{i+1}, \ldots s_n]$. By the UNWIND Assumption, there is a valid UNWIND-run $p_1, \ldots, p_m$ of t, such that $p_m = p$. Let Q be the set of the positions of the n holes of M. Then there is a least k such that $p_1, \ldots, p_k$ is a valid UNWIND-run of t, $p_k \in Q$ and p_k is the position

of some hole. Minimality of k implies that $p_1, \ldots, p_{k-1}$ are positions within M, but not the position of any hole of M. Now we can apply the conditions on UNWIND, in particular condition (2): UNWIND produces the valid UNWIND-run $p_1, \ldots, p_k$, irrespective of the terms in the holes of M. Hence the claim of the lemma holds. □

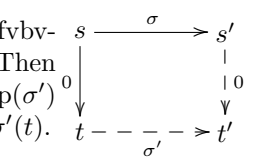
Assumption 3.4 (Answer Assumption). There is a set ANS of *answer terms*. We assume that the following conditions are satisfied:

1. If $t \xrightarrow{\sigma} t'$ for terms t, t' by a fvbv-renaming σ, and $t \in \text{ANS}$, then $t' \in \text{ANS}$.
2. If $t = M[t_1, \ldots, t_n]$ is an answer for some multicontext M, and no hole of M is a reduction position, then $M[t'_1, \ldots, t'_n]$ is also an answer.

The paradigm condition is that the closed answers are exactly the closed irreducible expressions. However, there are important calculi that do not satisfy this condition. Hence we allow more flexibility in matching our assumptions in higher-order calculi: answers are allowed to be reducible, which is the case for the variant of the π-calculus in Example 3.8. We allow also that not every irreducible expression (not even irreducible closed expression) is an answer, which allows to apply our methods to untyped higher-order calculi with stuck expressions.
The essence of the following assumption is that reduction commutes with renaming, and that reduction in strongly sharing calculi does not modify non-reduction positions up to renamings, and in weakly sharing calculi a replacement of variables by variables is permitted under further restrictions.

Assumption 3.5 (Reduction Assumption). It is assumed that CALC only defines a (small-step) relation $\to_0$ that is applicable to terms satisfying the DVC, and that the full small-step relation $\to$ is derived from $\to_0$ and a subsequent renaming of variables. The relation $\to$ is defined such that $s \to t$ holds whenever s, t satisfy the DVC, and $s \to_0 t_0 \xrightarrow{\sigma} t$ for some t_0 and some bv-renaming σ of t_0. We assume that the following conditions are satisfied for $\to$ and $\to_0$:

1. If $s \to t$, then s, t have the same type.
2. Let $t = M[t_1, \ldots, t_n]$ be a term that satisfies the DVC, where M is a multicontext with n holes that are at non-reduction positions, and let t' be a term with $t \to_0 t'$. Then there is a multicontext M' with n' holes, a mapping $\pi : \{1, \ldots, n'\} \to \{1, \ldots, n\}$, vv-substitutions $\nu_i \in VV, i \in \{1, \ldots, n'\}$, where the domain and codomain-variables of all ν_i already occur in M, such that for all terms $s_1, \ldots, s_n$: If $M[s_1, \ldots, s_n]$ satisfies the DVC, then $M[s_1, \ldots, s_n] \to_0 M'[\nu_1(s_{\pi(1)}), \ldots, \nu_{n'}(s_{\pi(n')})]$. In particular, $t' = M'[\nu_1(t_{\pi(1)}), \ldots, \nu_{n'}(t_{\pi(n')})]$.
   Note that $\nu_i, i = 1, \ldots, n$ are capture-free since the DVC holds before reduction and that the assumption enforces that ν_i are independent from the terms $s_1, \ldots, s_n$.
3. If s is a term satisfying the DVC, $s \to_0 t$, $s \xrightarrow{\sigma} s'$ an fvbv-renaming of s, such that $s' := \sigma(s)$ satisfies the DVC. Then there is a term t' and an fvbv-renaming σ' of t, where $\text{fvp}(\sigma')$ is a restriction of $\text{fvp}(\sigma)$, such that $s' \to_0 t'$ and $t' = \sigma'(t)$.



Note that in (2) the terms s_i satisfy the DVC, but there may be multiple occurrences of some s_i; also the multicontext M' may violate the DVC.

We give examples of calculi and illustrate the assumptions:

Example 3.6. Let the call-by-need λ-calculus[4] be given with syntax $E ::= V \mid (E\ E) \mid \lambda V.E \mid (\texttt{let}\ V = E\ \texttt{in}\ E)$ (see [AFM+95]). Then the search for a normal-order redex is deterministic and can be specified by a label-shift that starts with t^L and has the rules $(t_1\ t_2)^L \to (t_1^L\ t_2)$; $(\texttt{let}\ x = t_1\ \texttt{in}\ t_2)^L \to (\texttt{let}\ x = t_1\ \texttt{in}\ t_2^L)$ and $(\texttt{let}\ x = t\ \texttt{in}\ C[x^L]) \to (\texttt{let}\ x = t^L\ \texttt{in}\ C[x])$. It is clear that the UNWIND Assumptions are satisfied, since the search for the redex does not depend on former non-L-positions.
The answers are defined to be of the form $A ::= \lambda V.E \mid (\texttt{let}\ V = E\ \texttt{in}\ A)$. Then the Answer Assumption is satisfied.
The small-step reduction is defined at reduction positions and makes local changes at reduction positions, but non-reduction positions are never substituted. The full specification is beyond the scope of this paper, but it is easy to see (cf. [AFM+95]) that the Reduction Assumptions for a strongly sharing calculus are satisfied.

Example 3.7. Let a fragment of the untyped non-deterministic call-by-need λ-calculus with `amb` be given with syntax $E ::= V \mid (E\ E) \mid \lambda V.E \mid (\texttt{amb}\ E\ E) \mid (\texttt{letrec}\ x_1 = E_1, \ldots, x_n = E_n\ \texttt{in}\ E)$ (see [SSS08]). This calculus is strongly sharing in our sense. The search for a normal-order redex specified by a label-shift is non-deterministic. The labels are "T" for top-term, "S" for subterm, and "L" standing for "S" or "T". Let UNWIND start with t^T and let the rules be $(t_1\ t_2)^L \to (t_1^S\ t_2)$; $(\texttt{letrec}\ Env\ \texttt{in}\ r)^T \to (\texttt{letrec}\ Env\ \texttt{in}\ r^S)$, $(\texttt{letrec}\ x = t, Env\ \texttt{in}\ C[x^S]) \to (\texttt{let}\ x = t^S, Env\ \texttt{in}\ C[x])$, $(\texttt{letrec}\ x = t, y = C[x^S], Env\ \texttt{in}\ r) \to (\texttt{letrec}\ x = t^S, y = C[x], Env\ \texttt{in}\ r)$, and the non-deterministic rules for `amb` be: $(\texttt{amb}\ t_1\ t_2)^L \to (\texttt{amb}\ t_1^S\ t_2)$ and $(\texttt{amb}\ t_1\ t_2)^L \to (\texttt{amb}\ t_1\ t_2^S)$. If a position is visited twice, then the algorithm stops. Again it is clear that the UNWIND Assumptions are satisfied, since the search for the redex does not depend on non-labeled positions.
The answers are abstractions $\lambda V.E$ as well as abstractions with an enclosing `letrec`-expression $(\texttt{letrec}\ Env\ \texttt{in}\ \lambda V.E)$. The Answer Assumptions are satisfied.
The small-step reduction is defined at reduction positions and makes local changes, but again non-reduction positions are not modified by substitution. It is not hard to check that the Reduction Assumptions are satisfied. We give an illustration of one rule: $(\texttt{letrec}\ x = \lambda y.s, Env\ \texttt{in}\ C[x]) \to (\texttt{letrec}\ x = \lambda y.s, Env\ \texttt{in}\ C[\lambda y.s])$, provided there is an UNWIND-run with intermediate labeling $\ldots C[x^L]$. The Reduction Assumption (2) is satisfied, since e.g. $M[s_1, \ldots, s_n] = (\texttt{letrec}\ x = \lambda y.M_1[s_1, \ldots, s_n], Env\ \texttt{in}\ C[x])$ is reduced to $(\texttt{letrec}\ x = \lambda y.M_1[s_1, \ldots, s_n], Env\ \texttt{in}\ C[\lambda y.M_1[s_1, \ldots, s_n]])$, which

[4] There may be minor variations in the normal-order redex in the cited calculi

can be written as $M'[s_1, \ldots, s_n, s_1, \ldots, s_n]$, where $M'[\ldots] = (\texttt{letrec}\ x = \lambda y.M_1[\ldots], Env\ \texttt{in}\ C[\lambda y.M_2[\ldots]])$.
The non-deterministic call-by-need λ-calculus with `choice` in [MS99] is a bit different insofar as only variables are permitted as arguments in applications, and that beta-reduction is always a replacement of variables for variables, the rule being $(\lambda x.r)\ y \to r[y/x]$. This calculus is weakly sharing. For the case-rule (see [MS99]) a joint replacement of several variables will take place, which may be a non-injective mapping. Nevertheless, all our assumptions are satisfied.

Example 3.8. The π-calculus as already mentioned in Example 2.5 can also be checked for an appropriate representation and for the validity of our assumptions. In the literature there are different variants of the π-calculus, which are usually equipped with a theory based on bisimilarity. We will add a variant that is operationally admissible, though a full analysis is left for future work. Instead of equivalence axioms for concurrent processes and new name-binders, we view these as reduction rules, which will turn out to be completely adequate for our may- and must-convergence definitions, but are not compatible with the notion of total must convergence, since this encoding will introduce infinite reduction sequences.
The presentation of UNWIND as a label-shift algorithm has the following rules, where we also add the non-deterministic possibilities: $(P \mid Q)^L \to (P^L \mid Q)$, or $(P \mid Q)^L \to (P \mid Q^L)$; $(\nu x.P)^L \to \nu x.P^L$; and $(!P)^L \to (!\ P^L)$. This algorithm for finding reduction positions satisfies our UNWIND Assumptions. The reduction rules adapted to our view of calculi are the rules that correspond to "structural equivalences", which are replaced by the corresponding rules, like e.g. $(P \mid Q) \to (Q \mid P)$; $(P_1 \mid (P_2 \mid P_3)) \to ((P_1 \mid P_2) \mid P_3)$; $(\nu x.P) \mid Q \to (\nu x.(P \mid Q))$, if $x \notin \mathcal{FV}(Q)$, $\nu x.\nu y.P \to \nu y.\nu x.P$, and the rule $!P \to P \mid !P$. The important rules are $(P\ +\ Q) \to P$, $(P\ +\ Q) \to Q$ and the communication rule (COM): $x(y_1, \ldots, y_n).P \mid \overline{x}\langle z_1, \ldots, z_n\rangle.Q \to P[z_1/y_1, \ldots, z_n/y_n] \mid Q$. The rules can only be applied if the redex is a reduction position and not below an !-operator. The latter restriction shows that the notion of reduction position and the notion of redex may be different.
An appropriate definition of answers (or successful processes) that satisfies our assumptions is as follows: a process P is an answer, iff the rules for $P + Q$ and the communication rules are not applicable, even after a finite number of "equivalence reductions". This set of definitions satisfies all our assumptions for a weakly sharing calculus.
To use the structural equivalences and the definition of reduction modulo this equivalence is not covered by our framework, where the associative-commutative rules can be encoded, but the ν-shifting rules have to be made explicit.
Note that the (COM)-reduction rule from the π-calculus is the only rule among all other considered rules that has a non-linear left hand side, however, the rule application is severely restricted, insofar as its applicability depends only on equality of two variable names.

Remark 3.9. If a calculus that has value positions, then the possible interactions are restricted by the assumptions: for example, in a reduction $s \to t$, it

is not possible to copy/move a value v that is at a term position and also at a non-reduction position in s to a position in t that is value-position, since the Reduction Assumptions require that v is visible in the multicontext M' at a term position. This does not happen in common copy-rules, since only values are copied that are in reduction position.

Example 3.10. The calculus $\lambda(\mathsf{fut})$ in [NSSSS07] is an extension of the lambda-calculus with features of the π-calculus, however, there are cells and futures instead of channels. The representation in our syntax requires, similar as in the π-calculus, the two types process and term, where only variables of type term are permitted. The structural congruences can be represented as reduction rules as for the π-calculus. The lambda-calculus evaluation follows a call-by-value strategy, though there are futures (i.e. variables) used for sharing results instead of a replacing beta-rule. The answers are so called successful processes, which have no pending (non-equivalence) reduction possibilities. The calculus $\lambda(\mathsf{fut})$ satisfies our assumptions and is a strongly sharing calculus, using a specific abstract-syntax-encoding, hence the two context lemmas for may- and must-convergence hold.
If the abstract-syntax encoding uses the value-construct, and encodes cell-expressions $(x\ \mathtt{c}\ t)$, such that the second argument is a "W"-position of the operator $(\cdot\ \mathtt{c}\ \cdot)$, and only variables and abstractions are permitted as second argument, then all assumptions are satisfied, and again we derive the two context lemmas for may- and must-convergence.

Remark 3.11. We give examples of calculi that do not fall into the scope of our method to prove context lemmas:

- If reduction within abstractions is not permitted, then the beta-reduction rule violates our Reduction Assumption, since non-reduction positions may be modified: an example being $(\lambda x.C[x])(\lambda y.y) \to C[(\lambda y.y)]$. Using the multicontext $M = (\lambda x.[])$ and $s_1 = x$, the only possibility for an M' is $M' = ([])$, however, there is a replacement of x, and hence this is impossible.
- It is not permitted by our assumptions to have a beta-rule replacing only variables by variables like $(\lambda x.C[x])\ y \to C[y]$, if y is at a potential position of a hole: With $M = ((\lambda x.[])\ [])$, it is not possible to find an appropriate M', since the vv-substitution must only depend on M, not on the contents of the holes. If the argument position of applications is restricted to variables then the Reduction Assumptions are not violated.
- If we try to extend the assumptions and proof method such that $M[s_1, \ldots, s_n]$ may have a more general result $M'[\sigma_1(s_1), \ldots, \sigma_n(s_n)]$ after a single reduction, where σ_i is a substitution of terms for variables, then we run into technical trouble in the proofs of the context lemma(s) since the substitutions may depend on the terms s_i and not only on the multicontext M.
- Similar arguments hold for pattern match or case-rules that reduce $(\mathtt{case}\ (c\ s_1 \ldots s_n)\ \mathtt{of}(c\ x_1 \ldots x_n) \to s, \ldots)$ to e.g. $s[t_1/x_1, \ldots, t_n/x_n]$, which cannot be covered.

– A rule of the form $f\ (g\ a) \to b$, where f, g are unary and a, b are constants and the subterm $(g\ a)$ is a non-reduction position can not be represented, since this would imply by our assumptions that $f\ s \to b$ for all subterms s.

4 Contextual Preorder and Equivalence for May- and Must-Convergence

In this section we define different kinds of convergence properties of terms, and the corresponding notions of contextual preorder and equivalence. There are three main notions of convergence of a term t: may-convergence, which means that t may reduce to an answer, must-convergence, which means that every term reachable by reduction from t is may-convergent, and total must-convergence, which means that t has no reduction to a must divergent term (failure term) and no infinite reduction.

Definition 4.1. *A term t is called*

– may-convergent *iff there is some answer t' with $t \xrightarrow{*} t'$, denoted as $t\downarrow$.*
– must-divergent *iff t is not may-convergent, denoted as $t\Uparrow$.*
– may-divergent *iff there is some term $t' \Uparrow$ with $t \xrightarrow{*} t'$, denoted as $t\uparrow$.*
– must-convergent *iff $t \xrightarrow{*} t'$ implies $t'\downarrow$, denoted as $t\Downarrow$.*
– totally must-convergent *iff $t\Downarrow$ and there is no infinite reduction starting with t, denoted as $t\Downarrow\!\Downarrow$.*
– totally may-divergent *iff $t\uparrow$, or there is an infinite reduction starting with t, denoted as $t\uparrow\!\uparrow$.*

Note that t is not may-divergent iff it is must-convergent.
In calculi with a deterministic $\to_0$-reduction the may- and must-predicates are identical. As a generalization, we call a calculus *deterministic* iff the may- and must-convergence predicates are identical for all terms. Terms t with $t\Downarrow$, but not $t\Downarrow\!\Downarrow$, are called *weakly divergent* in [CHS05]. Must-convergence is interesting because it is linked to fairness (see e.g. [CHS05,SSS08,NSSSS07]); further justification for non-total may-divergence is in [SS03].

Definition 4.2. *Let s, t be two terms of the same type τ, and $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$. Then*

– *$s \leq_{\mathcal{M},\tau} t$ iff for all $C[]_\tau : C[s]\mathcal{M} \implies C[t]\mathcal{M}$, and*
– *$s \leq_{\mathcal{M}\nu,\tau} t$, iff for all $C[]_\tau$, for all vv-substitutions $\nu \in VV$ and for all vvbv-substitutions γ_s, γ_t compatible with ν on $\mathcal{FV}(s) \cup \mathcal{FV}(t)$: $C[\gamma_s(s)]\mathcal{M} \implies C[\gamma_t(t)]\mathcal{M}$.*

In the following we omit mention of τ in the suffix of the relations, if this is not ambiguous.
Easy consequences are that for all terms s: $s\Downarrow\!\Downarrow \implies s\Downarrow \implies s\downarrow$, that for $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$, the relations $\leq_{\mathcal{M}}$ are compatible with contexts, and reflexive and transitive, and that $\leq_{\mathcal{M}\nu} \subseteq \leq_{\mathcal{M}}$.

Note that for a general proof of transitivity of $\leq_{\mathcal{M}}$, without further assumptions on the calculus, it is far more appropriate to permit that $C[s], C[t]$ may contain free variables. The corresponding proof w.r.t. a definition of $\leq_{\mathcal{M}}$ that restricts $C[s], C[t]$ to be closed is in general not applicable, since the middle term $C[s_2]$ is not necessarily closed, if $C[s_1], C[s_3]$ are closed. An example for this phenomenon are the expressions $s_1 = (\lambda x.0)\ y$, $s_2 = (\lambda x.0)\ z$, $s_3 = (\lambda x.0)\ u$, where x, y, u are variables. In a lambda-calculus like in [SSSS08,SSS08], the expressions s_1, s_2, s_3 are equivalent. Using the context $C = (\texttt{letrec}\ y = 0, u = 0\ \texttt{in}\ [\cdot])$, we have that $C[s_1], C[s_3]$ are closed, but $C[s_2]$ is open, and hence transitivity using the "for all closing contexts"-definition of $\leq_{\mathcal{M}}$ can only be proved using specific properties of the calculus under consideration.
Contextual equivalence is defined as $\sim_{\downarrow} := \leq_{\downarrow} \cap \geq_{\downarrow}$ for deterministic calculi and for nondeterministic calculi as $\sim_{\downarrow\Downarrow} := \sim_{\downarrow} \cap \leq_{\Downarrow} \cap \geq_{\Downarrow}$ or $\sim_{\downarrow\Downarrow} := \sim_{\downarrow} \cap \leq_{\Downarrow\Downarrow} \cap \geq_{\Downarrow\Downarrow}$ depending on the used must-convergence predicate. The relations $\sim_{\downarrow\nu}$ are defined analogously using the respective $\leq$-relations.

Example 4.3. The relation $\leq_{\downarrow\nu}$ may be different from $\leq_{\downarrow}$ (in exotic calculi). Consider the calculus with one binary constructor c and a constant d, and the reduction rule: $c\ x\ x \to d$, let d be the only answer, and let all positions be reduction positions. Then $c\ x\ y \leq_{\downarrow} c\ x\ z$, but $c\ x\ y \not\leq_{\downarrow\nu} c\ x\ z$.

For simplifying several proofs in the following sections, we introduce a *0-1-labelled variant* of $\to$-reduction sequences, which is nothing else but a reduction of the form $s_{1,0} \to_0 s_{1,1} \to_1 s_{2,0} \to_0 s_{2,1} \to_1 \ldots$. I.e., a reduction, where $\to_0$-reductions and bv-renamings $\to_1$ are alternating, and the terms $s_{i,0}$ satisfy the DVC.

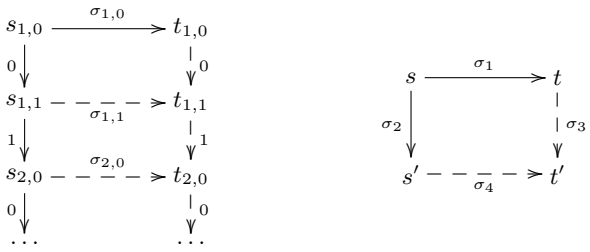


Fig. 1. Reduction diagrams for Lemma 4.4

Lemma 4.4. *Let s, t be terms satisfying the DVC with $s \xrightarrow{\sigma} t$ by an fvbv-renaming σ, and Red be a 0-1-labelled reduction of s as follows: $s = s_{1,0} \to_0 s_{1,1} \to_1 s_{2,0} \to_0 s_{2,1} \ldots \to_0 s_{n,1} \to_1 s_{n+1,0}$. Then there is also a reduction of t of the form $t = t_{1,0} \to_0 t_{1,1} \to_1 t_{2,0} \to_0 t_{2,1} \ldots \to_0 t_{n,1} \to_1 t_{n+1,0}$ with terms $t_{i,k}$, such that for all i, k: $s_{i,k} \xrightarrow{\sigma_{i,k}} t_{i.k}$ by fvbv-renamings $\sigma_{i,k}$, $\mathrm{fvp}(\sigma_{i,k})$ is a restriction of $\mathrm{fvp}(\sigma)$, the terms $t_{i,0}$ satisfy the DVC, and $s_{n+1,0}$ is an answer iff*

$t_{n+1,0}$ is an answer (see left diagram in figure 1).
The lemma holds also for bv-renamings instead of fvbv-renamings. The lemma also holds, if the 0-1-labelled reduction of s starts with a $\to_1$-reduction, in which case the reduction of t also starts with a $\to_1$-reduction.

Proof. We show the claim by induction on the number of reductions. The base case holds using the Answer Assumption 3.4. Let s, t be terms satisfying the DVC with $s \xrightarrow{\sigma} t$ by a fvbv-renaming σ, and $s \to_0 s'$. Then by the Reduction Assumption 3.5, there is some t' with $t \to_0 t'$, and $s' \xrightarrow{\sigma'} t'$, where $\mathrm{fvp}(\sigma')$ is a restriction of $\mathrm{fvp}(\sigma)$. Let s, t be terms with $s \xrightarrow{\sigma_1} t$ by an fvbv-renaming, and $s \xrightarrow{\sigma_2} s'$ by a bv-renaming, and s' satisfies the DVC. If s already satisfies the DVC, then σ_2 is the identity. The reverse σ_2^{-1} of σ_2 is also a bv-renaming. Moreover, by Lemma 2.9, there is a bv-renaming σ_3, such that $t \xrightarrow{\sigma_3} t'$ and t' satisfies the DVC. With $\sigma_4 = \sigma_2^{-1} \circ \sigma_1 \circ \sigma_3$, we have $s' \xrightarrow{\sigma_4} t'$ by Lemma 2.9, since composition of fvbv-renamings is also a fvbv-renaming. Moreover, the fv-parts of σ_1 and σ_4 are the same. (see right diagram in figure 1. □

Proposition 4.5. *Let s, s' be terms with $s' = \sigma(s)$, where σ is a fvbv-renaming. Then $s\mathcal{M} \Leftrightarrow s'\mathcal{M}$ for all $\mathcal{M} \in \{\downarrow, \Downarrow, \uparrow, \Uparrow, \Downarrow, \Uparrow\}$.*

Proof. Let s, t be terms with $t = \sigma(s)$, where σ is an fvbv-renaming.
If $s \downarrow$, then Lemma 4.4 and the condition on answer-terms 3.4 shows that a reduction from s to an answer can be translated to a reduction of t to an answer. Hence $t \downarrow$. Since the reverse of σ is a fvbv-renaming by Lemma 2.9, the converse also holds. This immediately also shows that $s \Uparrow \iff t \Uparrow$.
Now let $s \uparrow$. Then Lemma 4.4 and the first part of the proof shows that a reduction from s to a must-divergent term can be translated to a reduction of t to a must-divergent term. Hence $t \uparrow$. Since the reverse of σ is a fvbv-renaming by Lemma 2.9, the converse implication also holds. An immediate consequence is that $s \Downarrow \iff t \Downarrow$.
Let $s \Uparrow$. Then a reduction from s to a must-divergent term or an infinite $\to$-reduction starting from s can be transferred using Lemma 4.4 to a reduction from t to a must-divergent term or to an infinite reduction. Hence $t \Uparrow$. The reverse implication also holds. Again, an immediate consequence is $s \Downarrow\!\Downarrow \iff t \Downarrow\!\Downarrow$. □

5 Context Lemmas

We define the preorders restricted to reduction contexts and show the context lemmas for all the combinations of the different notions of convergencies and for the two kinds of calculi.

Definition 5.1. *For all terms s, t of equal type τ and $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$:*

- *$s \leq_{\mathcal{M}, R\nu, \tau} t$ iff for all reduction contexts $R[]_\tau$, all vv-substitutions $\nu \in VV$, all vvbv-substitutions γ_s, γ_t of s, t compatible with ν on $\mathcal{FV}(s) \cup \mathcal{FV}(t)$, we have $R[\gamma_s(s)]\mathcal{M} \implies R[\gamma_t(t)]\mathcal{M}$.*

- $s \leq_{\mathcal{M},R\overline{\nu},\tau} t$ *iff for all reduction contexts* $R[]_\tau$*, all vv-substitutions* $\nu \in VV$*, such that* $\nu(s), \nu(t)$ *is without variable capture, we have* $R[\nu(s)]\mathcal{M} \implies R[\nu(t)]\mathcal{M}$*.*
- *The relation* $s \leq_{\mathcal{M},R,\tau} t$ *holds iff for all reduction contexts* $R[]_\tau$*, we have* $R[s]\mathcal{M} \implies R[t]\mathcal{M}$*.*

Note that in the following, we will drop the type-suffix τ.
We can slightly restrict the necessary reduction contexts for $\leq_{\mathcal{M},R}$, by using renamings and Proposition 4.5:

Lemma 5.2. *Let* CALC *be strongly sharing and* $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$*. Then* $\leq_{\mathcal{M},R\nu} = \leq_{\mathcal{M},R}$*.*

Proof. This follows from Proposition 4.5, since for a reduction context R, σ_s, σ_t (with adapted positions) are also bv-renamings of $R[s], R[t]$, respectively. □

In the technical part for weakly sharing calculi we will work with the relation $\leq_{\mathcal{M},R\nu,\tau}$. We show that $\leq_{\mathcal{M},R\overline{\nu},\tau}$, which is more intuitive and which is also easier to check, is indeed equivalent to $\leq_{\mathcal{M},R\nu,\tau}$.

Lemma 5.3. *Let* $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$*. Then* $\leq_{\mathcal{M},R\nu,\tau} = \leq_{\mathcal{M},R\overline{\nu},\tau}$*.*

Proof. The containment $\leq_{\mathcal{M},R\nu,\tau} \subseteq \leq_{\mathcal{M},R\overline{\nu},\tau}$ is trivial. The nontrivial part is to show the other direction: $\leq_{\mathcal{M},R\overline{\nu},\tau} \subseteq \leq_{\mathcal{M},R\nu,\tau}$. For $s \leq_{\mathcal{M},R\overline{\nu},\tau} t$, we have to show that $s \leq_{\mathcal{M},R\nu,\tau} t$: let R be a reduction context, $\nu \in VV$, and γ_s, γ_t be vvbv-substitutions compatible with ν on $\mathcal{FV}(s) \cup \mathcal{FV}(t)$, and $R[\gamma_s(s)]\mathcal{M}$. Let $\sigma_{s,1}$ be a modification of $R[\gamma_s(s)]$ that renames all bound variables in $\gamma_s(s)$ to fresh and different variables. We will use Proposition 4.5 several times in the following. The first consequence is $\sigma_{s,1}R[\gamma_s(s)]\mathcal{M}$. Since the variables are fresh, there is a modification $\sigma_{s,2}$ of s, such that $\sigma_{s,1}R[\gamma_s(s)] = R[\nu(\sigma_{s,2}(s))]$. Note that the latter is an application of ν without capturing of variables. Now we can apply the assumption $s \leq_{\mathcal{M},R\overline{\nu},\tau} t$, which in turn implies that for all bv-modifications $\sigma_{s,3}, \sigma_{t,3}$, the relation $\sigma_{s,3}(s) \leq_{\mathcal{M},R\overline{\nu},\tau} \sigma_{t,3}(t)$ holds. Similar as for s, there are modifications $\sigma_{t,1}, \sigma_{t,2}$, such that $R[\gamma_t(t)]\mathcal{M} \iff \sigma_{t,1}R[\gamma_t(t)]\mathcal{M}$. There is a modification $\sigma_{t,2}$ of bound variables of t with fresh variables, such that $\sigma_{t,1}R[\gamma_t(t)] = R[\nu(\sigma_{t,2}(t))]$. Since from the assumption and from $R[\nu(\sigma_{s,2}(s))]\mathcal{M}$, we derive $R[\nu(\sigma_{t,2}(t))]\mathcal{M}$. The equivalences now show that $R[\gamma_t(t)]\mathcal{M}$. Since this holds for all reduction contexts R, the claim is proved.

We separate the proofs for weakly and strongly sharing calculi, since there is a different treatment of renamings, and the weakly sharing part requires more arguments w.r.t. vvbv-substitutions.

Lemma 5.4 (May-Convergence and Weakly Sharing). *Let* CALC *be weakly sharing. Then* $\leq_{\downarrow,R\nu} = \leq_{\downarrow\nu}$*.*

Proof. We show the following generalized claim:
For all n, all multicontexts M with n holes, all $i = 1, \ldots, n$ and all vv-substitutions $\nu_i \in VV$, and compatible vvbv-substitutions $\gamma_{s,i}, \gamma_{t,i}$ on $\mathcal{FV}(s_i) \cup$

$\mathcal{FV}(t_i)$: if for terms s_i, t_i: $s_i \leq_{\downarrow,R\nu} t_i$, then $M[\gamma_{s,1}(s_1),\ldots,\gamma_{s,n}(s_n)] \downarrow \Longrightarrow M[\gamma_{t,1}(t_1),\ldots,\gamma_{t,n}(t_n)] \downarrow$. For convenience let $s'_i := \gamma_{s,i}(s_i), t'_i := \gamma_{t,i}(t_i)$. Note that in the inductive proof below we will only use the weakened precondition $s'_i \leq_{\downarrow,R\nu} t'_i$. Proposition 4.5 permits us to assume, by applying bv-renamings, that the bound variables in s'_i and t'_i are distinct.
The claim is shown by induction on the length l of 0-1-labelled-reductions of $M[s'_1,\ldots,s'_n]$ to an answer, and second on the number of holes of M.
As a base case, the claim is obviously true, if the number n of holes is equal to 0, since then $M[s'_1,\ldots,s'_n] = M[t'_1,\ldots,t'_n]$.
There are two cases:
1. In $M[s'_1,\ldots,s'_n]$ some s'_i is in a reduction position. This means that at least one of the contexts $M_i = M[s'_1,\ldots,s'_{i-1},[],s'_{i+1},\ldots,s'_n]$ is a reduction context. Then Lemma 3.3 shows that there is some j, such that $M[s'_1,\ldots,s'_{j-1},[],s'_{j+1},\ldots,s'_n]$ as well as $M[t'_1,\ldots,t'_{j-1},[],t'_{j+1},\ldots,t'_n]$ is a reduction context. Using the induction hypothesis for the context $M' := M[[],\ldots,[],s'_j,[],\ldots,[]]$, which has $n-1$ holes, it follows that $M[s'_1,\ldots,s'_n]\downarrow \Longrightarrow M[t'_1,\ldots,t'_{j-1},s'_j,t'_{j+1},\ldots,t'_n]\downarrow$.
Since $M[t'_1,\ldots,t'_{j-1},[],t'_{j+1},\ldots,t'_n]$ is a reduction context, the assumption and $M[t'_1,\ldots,t'_{j-1},s'_j,t'_{j+1},\ldots,t'_n]\downarrow$ imply that $M[t'_1,\ldots,t'_{j-1},t'_j,t'_{j+1},\ldots,t'_n]\downarrow$.

2. For all i: None of the contexts $M_i = M[s'_1,\ldots,s'_{i-1},[],s'_{i+1},\ldots,s'_n]$ is a reduction context. Lemma 3.3 implies that none of the holes of M is at a reduction position. If $l=0$, then $M[s'_1,\ldots,s'_n]$ is an answer-term, and by Assumption 3.4, the expression $M[t'_1,\ldots,t'_n]$ is also an answer term. Now assume that $l>0$:

2a. First we consider the case that $M[s'_1,\ldots,s'_n]$ satisfies the DVC and that the reduction on $M[s'_1,\ldots,s'_n]$ is a $\to_0$-reduction. Let $M[s'_1,\ldots,s'_n] \to_0 s'$ be the start of the 0-1-labelled reduction of length l to an answer. By the Reduction Assumption 3.5 (2), there is a multicontext M' with n' holes, $\nu_i \in VV$ for $i \in \{1,\ldots,n'\}$, and a mapping $\pi : \{1,\ldots,n'\} \to \{1,\ldots,n\}$, such that $s' = M'[\nu_1(s'_{\pi(1)}),\ldots,\nu_{n'}(s'_{\pi(n')})]$.
The same holds by Reduction Assumption 3.5 (2) for $M[t'_1,\ldots,t'_n]$: there is a reduction $M[t'_1,\ldots,t'_n] \to_0 M'[\nu_1(t'_{\pi(1)}),\ldots,\nu_{n'}(t'_{\pi(n')})]$. Now we can apply the induction hypothesis, since the number of reductions to an answer of s' is $l-1$, and the required preconditions hold: for all R,ν_R and if $\gamma_{Rs,i},\gamma_{Rt,i}$ are compatible with ν_R on $\mathcal{FV}(s'_i)\cup\mathcal{FV}(t'_i)$, then for all $i=1,\ldots,n'$: $R[\gamma_{Rs,i}\nu_i(s'_{\pi(i)})]\downarrow \Longrightarrow R[\gamma_{Rt,i}\nu_i(t'_{\pi(i)})]\downarrow$ holds, since $\gamma_{Rs,i}\nu_i$ and $\gamma_{Rt,i}\nu_i$ are also vvbv-substitutions compatible with a common vv-substitution (see Lemma 2.9).

2b. The other case is that $M[s'_1,\ldots,s'_n]$ is the result of a $\to_0$-reduction and the next reduction step in the 0-1-labelled reduction is a bv-renaming. Then the reduction consists of applying some bv-renaming $M[s'_1,\ldots,s'_n] \xrightarrow{\sigma} M'[s''_1,\ldots,s''_n]$, such that $M'[s''_1,\ldots,s''_n]$ satisfies the DVC.
Using Lemma 2.11, let σ_M be the part of the renaming σ for the binder positions that are in M. Let $W_i, i=1,\ldots,n$ be the set of variables that may be potentially bound in hole i, i.e. $W_i = V_{\text{hole}}(M,i)$, and let $\rho_i, i=1,\ldots,n$ be the mappings on W_i induced by σ. Note that ρ_i is injective on W_i. The effect of the bv-renaming

σ can be modelled as follows:
It induces a bv-renaming $M \xrightarrow{\sigma_M} M'$, and fvbv-renamings μ_i with $s'_i \xrightarrow{\mu_i} s''_i$, where μ_i is compatible with ρ_i for all i. We construct an appropriate bv-renaming σ' for $M[t'_1, \ldots, t'_n]$ by using σ_M again for M, and for every i fvbv-renamings μ'_i for t'_i, where for all i: μ'_i is compatible with ρ_i. For the bv-part of μ'_i fresh variables must be used, which ensures that $\sigma'(M[t'_1, \ldots, t'_n])$ satisfies the DVC. By construction, we have $\sigma'(M[t'_1, \ldots, t'_n]) = M'[\mu'_1(t'_1), \ldots, \mu'_n(t'_n)]$.
It remains to show that the preconditions $\mu_i(s'_i) \leq_{\downarrow, R\nu} \mu'_i(t'_i))$ hold for all pairs $(\mu_i(s'_i), \mu'_i(t'_i))$: Let i be fixed in the following, and let R be a reduction context, let ν' be a vv-substitution, and $\gamma'_{s,i}$ be vvbv-substitutions of $\mu_i(s'_i)$ compatible with ν' such that $R[\gamma'_{s,i}\mu_i(s'_i)] \downarrow$. Now let $\gamma'_{t,i}$ be a vvbv-substitution of $\mu'_i(t'_i)$ compatible with ν'. The substitutions $\gamma'_{s,i}\mu_i$ and $\gamma'_{t,i}\mu'_i$ are compatible with the same vv-substitution $\nu\rho_{i|\rho_i(W_i)}$ by Lemma 2.9. Thus we obtain from $s_i \leq_{\downarrow, R\nu} t_i$ that $R[\gamma'_{t,i}\mu'_i(t'_i)] \downarrow$.
Since the preconditions are satisfied for all i, the multicontext is the same M' for $s_i, t_i, i = 1, \ldots, n$, and the reduction length has been reduced, we can apply the induction hypothesis. □

For a finite set of variables W, a context C is called *fresh for* W, iff for all variables $x \in W$, x is not bound by a binder in $BP_{\overline{\text{hole}}}(C)$.

Lemma 5.5. *Let s, t be terms, $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\}$ and W be a finite set of variables that contains all variables occurring in s, t. Then $s \leq_{\mathcal{M},R} t$ holds, iff for all reduction contexts R that are fresh for W, we have $R[s]\mathcal{M} \implies R[t]\mathcal{M}$.*

Proof. Let R be an arbitrary reduction context with $R[s] \downarrow$. We have to show that $R[t] \downarrow$. Let σ be a bv-renaming that renames all the binders in $BP_{\overline{\text{hole}}}(R)$ by fresh variables that are not in W. Then $\sigma(R)$ is a reduction context due to the unwind-conditions and satisfies the preconditions. Hence $\sigma(R)[s] \downarrow$ by Proposition 4.5. The preconditions imply that $\sigma(R)[t] \downarrow$. Since the reverse of σ is also a bv-renaming, we have also $R[t] \downarrow$, again by Proposition 4.5. The same arguments, but using the other claims of Proposition 4.5, show the other parts of the lemma. □

Lemma 5.6 (May-Convergence and Strongly Sharing). *Let* CALC *be strongly sharing. Then $\leq_{\downarrow,R} \;=\; \leq_{\downarrow}$.*

Proof. We show the following generalized claim:
For all n and all multicontexts M with n holes: if for terms $s_i, t_i, i = 1, \ldots, n$, and for all $i = 1, \ldots, n$: $s_i \leq_{\downarrow,R} t_i$, then $M[s_1, \ldots, s_n] \downarrow \implies M[t_1, \ldots, t_n] \downarrow$.
Note that the induction and the cases are instances of the cases in the proof of Lemma 5.4, where vv-substitutions can be omitted (see also Lemma 5.2); only the final part (2b) is different. We will present this part of the proof in detail:
2b. Consider the case that $M[s_1, \ldots, s_n]$ is the result of a $\to_0$-reduction, that it does not satisfy the DVC, and the next reduction step in the 0-1-labelled reduction is a renaming. Then the reduction consists of applying some bv-renaming $M[s_1, \ldots, s_n] \xrightarrow{\sigma} M'[s'_1, \ldots, s'_n]$, such that $M'[s'_1, \ldots, s'_n]$ satisfies the DVC. Note

that every term s_i satisfies the DVC, but there may be double occurrences of the same term.
Using Lemma 2.11, let σ_M be the part of the renaming σ for the binder positions that are in M. Let $W_i = V_{\text{hole}}(M, i), i = 1, \ldots, n$, and let $\rho_i, i = 1, \ldots, n$ be the mappings on W_i induced by σ. Note that ρ_i is injective on W_i. The bv-renaming σ induces fvbv-renamings μ_i with $s_i \xrightarrow{\mu_i} s'_i$, where μ_i is compatible with ρ_i. We construct an appropriate bv-renaming σ' for $M[t_1, \ldots, t_n]$ by using σ_M again for M, and for every i fvbv-renamings μ'_i for t_i, where for all i: μ'_i is compatible with ρ_i. For the bv-part of μ'_i fresh variables must be used, which ensures that $\sigma'(M[t_1, \ldots, t_n])$ satisfies the DVC. By construction, we have $\sigma'(M[t_1, \ldots, t_n]) = M'[\mu'_1(t_1), \ldots, \mu'_n(t_n)]$.
We have to show that the precondition $\mu_i(s_i) \leq_{\downarrow,R} \mu'_i(t_i))$ holds for all pairs $(\mu_i(s_i), \mu'_i(t_i)), i = 1, \ldots, n$: Let i be fixed in the following, and let R be a reduction context with $R[\mu_i(s_i)] \downarrow$, where we assume using Lemma 5.5 that the binders $BP_{\overline{\text{hole}}}(R)$ use fresh variables. We have to show that $R[\mu'_i(t_i)] \downarrow$.
We construct an fvbv-renaming σ_2 of $R[\mu_i(s_i)]$ as follows, such that σ_2 acts as the inverse of μ_i on s_i; moreover, the mapping of σ_2 on $V_{\text{hole}}(R)$ is a restriction of ρ_i^{-1}. Note that σ_2 may also act on free variables in $R[\mu_i(s_i)]$, since R may have too few binders. The construction of σ_2 is possible due to the freshness assumption on R and since s_i satisfies the DVC. Then $\sigma_2(R[\mu_i(s_i)]) = \sigma_2(R)[s_i]$, and $\sigma_2(R)$ is a reduction context by Assumption 3.2 on UNWIND. Proposition 4.5 shows that $\sigma_2(R)[s_i] \downarrow$. The assumptions $s_i \leq_{\downarrow,R} t_i$ now imply that $\sigma_2(R)[t_i] \downarrow$.
Now starting with $R[\mu'_i(t_i)]$, we also construct an fvbv-renaming σ_3 of $R[\mu'_i(t_i)]$ that acts as a reverse of μ'_i on t_i, such that the induced mapping on $V_{\text{hole}}(R)$ is a restriction of ρ_i^{-1}. We also assume that σ_3 renames binders in $BP_{\overline{\text{hole}}}(R)$ exactly as σ_2. Since t_i satisfies the DVC, and due to the freshness assumptions for R, there is no conflict between variables from t_i, bound variables in R and variables in $\mu'_i(t_i)$, hence σ_3 can be constructed and is an fvbv-renaming. Then $\sigma_3(R[\mu'_i(t_i)] = \sigma_3(R)[t_i] = \sigma_2(R)[t_i]$. Now $\sigma_2(R)[t_i] \downarrow$ and Proposition 4.5 imply $R[\mu'_i(t_i)] \downarrow$. □

Lemma 5.7 (Must-Convergence and Strongly Sharing). *Let* CALC *be strongly sharing. Then* $\leq_{\downarrow,R} \cap \leq_{\Downarrow,R} \subseteq \leq_{\Downarrow}$.

Proof. We show the following generalized claim, using may-divergence.
For all n and all multicontexts M with n holes: if for all for terms $s_i, t_i, i = 1, \ldots, n$, and for all $i = 1, \ldots, n$: $s_i \leq_{\downarrow,R} t_i \wedge s_i \leq_{\Downarrow,R} t_i$, then $M[t_1, \ldots, t_n] \uparrow \Longrightarrow M[s_1, \ldots, s_n] \uparrow$.
The claim is shown by induction on the number l of $\to_0$ and $\to_1$-reductions of 0-1-labeled reductions of $M[t_1, \ldots, t_n]$ to a must-divergent term, and second on the number of holes of M. The proof is almost a copy of the proof of the context lemma 5.6 for may-divergence; we give a sketch and emphasize the differences:
If some term t_i is in a reduction position in $M[t_1, \ldots, t_n]$, then the arguments are the same as in the proof of Lemma 5.6.
If no hole of $M[t_1, \ldots, t_n]$ is a reduction position, $l > 0$, and the reduction is a $\to_0$-reduction, then the same arguments as in the in proof of Lemma 5.6 show

that we can use induction on l.
The base case $l = 0$ is that $M[t_1, \ldots, t_n]$ is must-divergent: suppose that $M[s_1, \ldots, s_n]$ is not must-divergent. Then it is may-convergent, which by the assumption $\forall i : s_i \leq_{\downarrow,R} t_i$ and the context lemma 5.6 implies that $M[t_1, \ldots, t_n] \downarrow$, which is a contradiction. Hence $M[s_1, \ldots, s_n] \uparrow$, and the base case is proved.
If no hole of $M[t_1, \ldots, t_n]$ is a reduction position, $l > 0$, and the reduction is a renaming $\rightarrow_1$, then the same arguments as in the proof of the may-context lemma 5.6 apply. □

An immediate consequence is:

Corollary 5.8. *Let* CALC *be strongly sharing. Then*

$$\begin{array}{ll} \leq_{\downarrow} \cap \leq_{\Downarrow,R} & \subseteq \leq_{\Downarrow} \\ \leq_{\downarrow,R} \cap \leq_{\Downarrow,R} & = \leq_{\downarrow} \cap \leq_{\Downarrow} \\ \leq_{\downarrow} \cap \leq_{\Downarrow,R} & = \leq_{\downarrow} \cap \leq_{\Downarrow} \end{array}$$

Lemma 5.9 (Must-Convergence and Weakly Sharing). *Let* CALC *be weakly sharing. Then* $\leq_{\downarrow,R\nu} \cap \leq_{\Downarrow,R\nu} \subseteq \leq_{\Downarrow\nu}$, *and hence also* $\leq_{\downarrow,R\nu} \cap \leq_{\Downarrow,R\nu} = \leq_{\downarrow\nu} \cap \leq_{\Downarrow\nu}$,

Proof. The proof can be done similar to the argumentation of the proof of Lemma 5.7 with analogous extensions as done in the proof of Lemma 5.4. □

Lemma 5.10 (Total Must-Convergence and Strongly Sharing). *Let* CALC *be a strongly sharing calculus. Then* $\leq_{\Downarrow\Downarrow,R} t = \leq_{\Downarrow\Downarrow}$.

Proof. We show the following generalized claim:
For all n and all multicontexts M with n holes: if for all terms s_i, t_i and for all $i = 1, \ldots, n$: $s_i \leq_{\Downarrow\Downarrow,R} t_i$, then $M[s_1, \ldots, s_n]\Downarrow\Downarrow \implies M[t_1, \ldots, t_n]\Downarrow\Downarrow$.
Thus, let us assume that M, s_i, t_i are given, that $M[s_1, \ldots, s_n]\Downarrow\Downarrow$, and that the claim holds for all terms that can be reached from $M[s_1, \ldots, s_n]$ by a 0-1-labelled $\rightarrow$-reduction sequence, where at least one $\rightarrow_0$ reduction is included. Proposition 4.5 permits us to assume, by applying bv-renamings, that the bound variables in s_i and t_i are distinct. The claim is shown by well-founded induction on the order $\xrightarrow{+}$ defined by the reduction $\rightarrow$ for all the descendents of $M[s_1, \ldots, s_n]$, and second on the number of holes of M. As a base case, the claim is obviously true, if $n = 0$.
There are several cases, we give a sketch for every case:

1. Some s_i or t_i is in a reduction position in $M[s_1, \ldots, s_n]$ or $M[t_1, \ldots, t_n]$, respectively. Then some hole of $M[., \ldots, .]$ is in a reduction context, and the arguments in case (1) of the proof of Lemma 5.6 (resp. Lemma 5.4) apply using induction on the number of holes.
2. None of the contexts $M_i = M[s_1, \ldots, s_{i-1}, [], s_{i+1}, \ldots, s_n]$ is a reduction context. Then no hole of M is a reduction position. We have to show that all reduction sequences of $M[t_1, \ldots, t_n]$ terminate. If $M[t_1, \ldots, t_n]$ is an answer-term, then we are finished. If $M[t_1, \ldots, t_n]$ is irreducible, then by the same

arguments as in in the proof of Lemma 5.6, $M[s_1, \ldots, s_n]$ is also irreducible, which implies that $M[s_1, \ldots, s_n]$ is an answer, and hence $M[t_1, \ldots, t_n]$ is an answer, too. If $M[t_1, \ldots, t_n]$ has a reduction, then using the Reduction Assumptions 3.5 for $\to_0$ and the same arguments as in Lemma 5.6, we can apply the induction hypothesis. □

The already demonstrated techniques suffice to prove:

Lemma 5.11 (Total Must-Convergence for Weakly Sharing). *Let* CALC *be weakly sharing. Then* $\leq_{\Downarrow,R\nu} \;=\; \leq_{\Downarrow\nu}$.

Also using Lemma 5.3, we obtain:

Theorem 5.12 (Generic Context Lemma). *Let* CALC *be a sharing calculus, i.e. such that our Assumptions 3.2, 3.4 and 3.5 are satisfied. Then the following relations hold:*

strongly sharing	*weakly sharing*
$\leq_{\downarrow,R} \;=\; \leq_{\downarrow}$	$\leq_{\downarrow,R\overline{\nu}} \;=\; \leq_{\downarrow\nu} \;\subseteq\; \leq_{\downarrow}$
$\leq_{\Downarrow,R} \;=\; \leq_{\Downarrow}$	$\leq_{\Downarrow,R\overline{\nu}} \;=\; \leq_{\Downarrow,\nu} \;\subseteq\; \leq_{\Downarrow}$
$\leq_{\downarrow,R} \cap \leq_{\Downarrow,R} \;=\; \leq_{\downarrow} \cap \leq_{\Downarrow}$	$\leq_{\downarrow,R\overline{\nu}} \cap \leq_{\Downarrow,R\overline{\nu}} \;=\; \leq_{\downarrow\nu} \cap \leq_{\Downarrow\nu} \;\subseteq\; \leq_{\Downarrow}$

6 Deriving a Generic IU-Theorem for Non-Sharing Calculi

In this section we extend the context lemmas also to call-by-name and call-by-value calculi that violate the Reduction Assumption, and in addition do not permit a `letrec`. E.g. calculi employing beta-reduction, fixpoint-reduction and/or a substituting case-reduction. We will show in the following that under certain restrictions, variants of ciu-theorems can be derived from the context lemmas in sharing calculi via a translation. We call them iu-theorems, since we have to omit the closedness restriction.

The idea of deriving such an iu-theorem for calculi with substituting beta- or case-reduction is as follows: For a generic calculus CALC with non-sharing call-by-value or call-by-name reduction we translate CALC into a sharing variant CALC_L of this calculus, by adding a `let`-construct and by modifying the substituting reduction of CALC into a reduction strategy which delays the substitutions until they are needed. After showing that CALC_L fulfills the requirements such that the context lemma holds, we transfer the context lemma derived for CALC_L into the setting of CALC by showing that the translation from CALC into CALC_L is adequate, i.e. is compositional and preserves and reflects may-, must- and total-must-convergence.

In the formulation of the assumptions we assume that the multicontext D is like a linear lambda-expression, i.e. the sequence of the holes may be different from the sequence of the occurrences in the represented expression, e.g. $D[\cdot_1, \cdot_2] = (\cdot_2\ \cdot_1)$ is allowed. We also allow holes for an operand $x_1 \ldots . x_n.t$ for an easier presentation.

For values v we define the set $FLP(v)$ of *flat positions*, which are all positions in v that are not bound within v. More formally, the set is inductively defined as follows: $\varepsilon \in FLP(v)$; and if $v = f(v_1, \ldots, v_n)$ and $\beta(f)_i =$ "T", then for all $p \in FLP(v_i)$, also $i.p \in FLP(v)$.

Assumption 6.1 (Non-Sharing Assumption). The assumptions are formulated for call-by-value and call-by-name variants of calculi: (Note that $\to$ is the reduction $\to_0$ with a subsequent renaming of variables if necessary).

1. If $s \to_0 t$, then s, t have the same type.
2. There is no reduction position in the scope of a variable binder.
3. For the call-by-value variant: whenever v is in a reduction position, then all flat positions of v are also in a reduction position.
4. If $s \to_0 t$, then the reduction either satisfies the Reduction Assumption 3.5 for strongly sharing calculi, or one of the following cases holds:
   (a) For call-by-value calculi: the reduction $s \to_0 t$ is of the form $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]] \to_0 R[D'[t_0[t_1/x_1, \ldots, t_n/x_n]]]$, where R is a reduction context, D, D' are multicontexts, all term-positions including the holes for the expressions t_i within D are reduction positions in $R[D[\cdot, \ldots, \cdot]]$, all terms t_i are values, and $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]]$ satisfies the DVC. Moreover, for all expressions $r, r_1, \ldots, r_n$, if $D[x_1 \ldots x_n.r, r_1, \ldots, r_n]$ satisfies the DVC, and all r_i are values, then $R[D[x_1 \ldots x_n.r, r_1, \ldots, r_n]] \to_0 R[D'[r[r_1/x_1, \ldots, r_n/x_n]]]$ is also a reduction.
   (b) For call-by-name calculi: the reduction $s \to_0 t$ is of the form $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]] \to_0 R[D'[t_0[t_1/x_1, \ldots, t_n/x_n]]]$, where R is a reduction context, D, D' are multicontexts, all term-positions (not necessarily the holes) within D are reduction positions in $R[D[\cdot, \ldots, \cdot]]$, and $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]]$ satisfies the DVC. Moreover, for all expressions $r, r_1, \ldots, r_n$, if $D[x_1 \ldots x_n.r, r_1, \ldots, r_n]$ satisfies the DVC, then $R[D[x_1 \ldots x_n.r, r_1, \ldots, r_n]] \to_0 R[D'[r[r_1/x_1, \ldots, r_n/x_n]]]$ is also a reduction.

   We assume that the renaming condition in Assumption 3.5.3 holds, which is true, if for R, D, D' above, also the fvbv-renamed variants satisfy the for-all-conditions above.

Note that the variables $x_i, i = 1, \ldots, n$ are free variables in t_0, and that a variable capture in the replacement $t_0[t_1/x_1, \ldots, t_n/x_n]$ cannot occur.

Definition 6.2. *A calculus is a* non-sharing call-by-value calculus *(a* non-sharing call-by-name calculus*, respectively) iff it is a calculus according to Section 3 that satisfies the following*

1. *The calculi must not permit "V"-positions in the syntax. A non-sharing call-by-name calculus in addition must not permit "W"-positions.*
2. UNWIND-*Assumption 3.2 with the additional restriction that there is no reduction position in the scope of a binder,*

3. *the Answer Assumption 3.4,*
4. *instead of the Reduction Assumption 3.5 it satisfies the Non-Sharing Assumption 6.1 for call-by-value (call-by-name, respectively).*

In the rest of this section we assume that CALC is a non-sharing calculus according to Definition 6.2, where at certain places we have to distinguish between the call-by-name and call-by-value variant.

Example 6.3. The beta-reduction rule $R[(\lambda x.s)\ t] \to R[s[t/x]]$ does not satisfy the Reduction Assumption 3.5, provided some occurrences of x in $R[s]$ are not in a reduction position. The call-by-name beta-reduction satisfies the Non-Sharing Assumption, where $D = (@(\lambda[\cdot], [\cdot]))$, which is easily verified. The call-by-value beta-reduction also satisfies the Non-Sharing Reduction Assumption, where the values are in general the abstractions, or if there are constructors: all expressions constructed from constructors, variables and abstractions. If the calculus contains case and constructors, the substituting case-rule is like the beta-reduction, where, however, depending on the types, the constructors and their arities, a family of D, D'-pairs is required. E.g.: $R[\texttt{case}\ (\texttt{Cons}\ s_1\ s_2)\ \texttt{of Nil} \to r_1;\ (\texttt{Cons}\ x_1\ x_2)] \to r_2) \longrightarrow R[r_2[s_1/x_1, s_2/x_2]]$.

Remark 6.4. The following usual fixpoint rules can also be used in the respective calculi, if the beta-reduction rule is also permitted.

1. Let the calculus be a call-by-value calculus, such that all abstraction are values. Then $R[Fix\ (\lambda x.s)] \to R[s[\lambda y.((Fix\ (\lambda x.s))\ y)/x]]$, where R is a reduction context, is a fixpoint reduction rule.
2. Let the calculus be a call-by-name calculus. Then $R[Fix\ (\lambda x.s)] \to t = R[s[(Fix\ (\lambda x.s))/x]]$, where R is a reduction context, is a fixpoint rule.
3. It is also possible to permit the following fixpoint-reduction rule $R[(\mu x.s)] \to R[s[(\mu x.s)/x]]$, where R is a reduction context. For call-by-value calculi, $\mu x.r$ must be a value for all r.

The rules do not satisfy the Non-Sharing Reduction Assumption, however, they can be performed using two steps, where the first step satisfies the Sharing Reduction Assumption and the second step satisfies the Non-Sharing Reduction Assumption. The first reduction can be done in two steps in call-by-value calculi: $R[Fix\ (\lambda x.s)] \to R[(\lambda x.s)\ (\lambda y.((Fix\ (\lambda x.s))\ y))] \to R[s[\lambda y.((Fix\ (\lambda x.s))\ y)/x]]$. Of course, we have to assume that the calculus has an appropriate definition of reduction position, e.g. the usual one.
Similarly, for the second rule in call-by-name calculi, the fixpoint reduction rule can be done in two steps: $R[Fix\ (\lambda x.s)] \to R[(\lambda x.s)\ (Fix\ (\lambda x.s))] \to R[s[(Fix\ (\lambda x.s))/x]]$.
The μ-reduction rule can in the same way as be splitted into $R[(\mu x.s)] \to R[(\lambda x.s)\ (\mu x.s)] \to R[s[(\mu x.s)/x]]$.
For each rule, we assume that the translation into it CALC^L below is applied only to the beta-rule. □

We construct a strongly sharing calculus CALC^L corresponding to CALC: its syntax is the syntax of CALC extended by a (non-recursive) `let`-construct (`let` $x = t$ `in` s), where $\beta(\texttt{let}) = (1, \text{`}T\text{"})$ for call-by-name, and $\beta(\texttt{let}) = (1, \text{`}W\text{"})$ for call-by-value. We abbreviate right-nested lets as e.g (`let` $x_1 = t_1, \ldots, x_n = t_n$ `in` r), and mean (`let` $x_1 = t_1$ `in` $\ldots$ (`let` $x_n = t_n$ `in` r)$\ldots$).
In (`let` $x = t$ `in` s), the argument t is a value-argument for call-by-value, whereas for call-by-name, we assume that t is a term-argument.
The UNWIND in CALC^L is the same as for CALC, where the UNWIND always jumps from a `let`-expression (`let` $x_1 = t_1, \ldots, x_n = t_n$ `in` r) to r.
The answers of CALC^L are exactly the expressions that produce CALC-answers if every let-expression (`let` $x_1 = t_1, \ldots, x_n = t_n$ `in` r) is replaced by its rightmost expressions, (i.e. r).
The reductions in CALC^L are as follows:

1. Whenever a `let`-expression is not the top-expression, not the right term of a `let`, and it is in a reduction position, then a reduction step is to move the `let`-environment one level higher. I.e. for a context C_1, where the hole has a position of length 1: $R[C_1[(\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ t)]] \to R[(\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ C_1[t])]$. Note that $R[(\texttt{let}\ y_1 = s_1, \ldots, y_m = s_m\ \texttt{in}\ (\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ t))] = R[(\texttt{let}\ y_1 = s_1, \ldots, y_m = s_m, x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ t)]$.
2. Whenever a variable x is in a reduction context, and x is in the scope of a `let`-expression with binding $x = s$, then a reduction step is $R[x] \to R[s]$.
3. CALC-reductions that obey the Reduction Assumption 3.5 are also permitted in CALC^L.
4. CALC-reductions that do not obey the (sharing) Reduction Assumption 3.5 are translated to CALC^L as follows: they are of the form $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]] \to R[D'[t_0[t_1/x_1, \ldots, t_n/x_n]]]$ in CALC, and translated as $R[D[x_1 \ldots x_n.t_0, t_1, \ldots, t_n]] \to R[D'[(\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ t_0)]]$.

We define two translations T : CALC $\to$ CALC^L and T^- : CALC^L $\to$ CALC where $T(t) := t$ and $T^-(\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ r) := T^-(r)[T^-(t_1)/x_1, \ldots, T^-(t_n)/x_n]$, and $T^-(c\ t_1, \ldots, t_n) := c\ (T^-(t_1), \ldots, T^-(t_n))$ for other operators c.
The translation T translates terms as well as contexts, whereas T^- does not translate contexts. It is obvious that $T^-(T(t)) = t$ for all CALC-expressions t.

Proposition 6.5. *The calculus* CALC^L *is a strongly sharing calculus.*

Proof. We check all the assumptions.

1. The UNWIND-Assumption is satisfied, since the search for a reduction positions in the intermediate `let`-expressions always proceed with the right expression, and since the assumption for values in call-by-value calculi is defined top-down.
2. The Answer-Assumption is also satisfied, which follows from the validity of the answer-assumption for CALC, and from the definition of answers in the `let`-case.

3. The Reduction Assumption for CALC-reductions is satisfied, which requires more arguments. The renaming condition in the Reduction Assumption 3.5.3 is satisfied, which follows from the assumptions on renamings in the Non-Sharing Reduction Assumption. We look at the other cases one-by-one:
   For the let-shifting the Reduction Assumption obviously holds. For the copy-reduction the Reduction Assumption holds: the expressions in non-reduction positions are only moved to other positions or are copied to other positions. The translated call-by-name reduction is restricted such that every expression in non-reduction position is either in the context R, and hence inherited to the result, or it is a subexpression of $t, t_1, \ldots, t_n$, and hence only moved. Moreover, the "for-all"-condition also holds.
   The translated call-by-value reduction is slightly different. Again expressions in non-reduction position are either in the context R, and hence inherited to the result, or they are subexpressions of t_0, or of the values $t_1, \ldots, t_n$, but in this case only at non-flat positions of t_i, and hence these subexpressions are only moved. The assumption on reduction positions within values shows that if a value is moved from a non-reduction position into a reduction position, then all its flat positions are also in reduction position. Hence holes at value positions in the multicontexts are not required (in the proof of the context lemma).

□

Proposition 6.6. *For every* CALC*-expression t, we have*
$t\downarrow \iff T(t)\downarrow$, $t\Downarrow \iff T(t)\Downarrow$, $t\Downarrow\!\Downarrow \iff T(t)\Downarrow\!\Downarrow$.
For every CALC^L*-expression s, we have*
$s\downarrow \iff T^-(s)\downarrow$, $s\Downarrow \iff T^-(s)\Downarrow$, $s\Downarrow\!\Downarrow \iff T^-(s)\Downarrow\!\Downarrow$.

Proof. A base case is that t is an answer iff $T(t)$ is an answer.

- If $t\downarrow$, then there is a CALC-reduction sequence $t = t_0 \to t_1 \to \ldots \to t_n$, where t_n is an answer. Inspecting the reduction in CALC^L, it is easy to see that $T(t_0) = t'_0 \xrightarrow{+} t'_1 \xrightarrow{+} \ldots \xrightarrow{+} t'_n$, such that $T^-(t'_i) = t_i$ for all i, and t'_n is an answer. The reverse is also easy to show: that for every CALC^L-reduction sequence $t'_0 \to t'_1 \to \ldots \to t'_n$, we always have a sequence in CALC with $T^-(t'_0) = t_0; t_1 \ldots t_n$, such that $T^-(t'_i) = t_i$ for all i, and either $t_{i-1} = t_i$, or $t_{i-1} \to t_i$.
- We show that $t\uparrow \iff T(t)\uparrow$. The base case is that $t\Uparrow$. Then the first part shows that this is equivalent to $T(t)\Uparrow$. Now the same arguments as above show how CALC-reduction sequences to must-divergent expressions can be transferred into CALC^L-reduction sequences to must-divergent expressions.
- To show that $t\Downarrow\!\Downarrow \iff T(t)\Downarrow\!\Downarrow$ we first show $t\Downarrow\!\Downarrow \implies T(t)\Downarrow\!\Downarrow$ using well-founded induction. This is possible since $t\Downarrow\!\Downarrow$ means that every reduction terminates. The base case is already shown, and the induction step is already contained in the arguments above. The reverse is analogous.

The next part is to show the same for T^-, which is completely analogous to the proof above, the only difference is that the CALC^L-expressions may have more let-subexpressions. □

Proposition 6.7. *The translation T is adequate in the following sense: For all* CALC*-expressions s, t and for $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$: $T(s) \leq_{c,\mathcal{M}} T(t) \implies s \leq_{c,\mathcal{M}} t$.*

Proof. This follows from Proposition 3.3 in [SSNSS08], since the translation T is compositional, i.e. $T(C[s]) = T(C)[T(s)]$ for all CALC-contexts C and CALC-expressions s and since T is convergence equivalent (see Proposition 6.6). □

Now we can conclude that the (sharing) context lemmas hold in the language CALC^L, however, our goal was to obtain (non-sharing) context lemmas in CALC. These are obtained by backtranslation. Before stating and proving the theorem, we define the following relations:

Definition 6.8. *Let* CALC *be a non-sharing calculus, let $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$ and let s, t be two* CALC*-expressions of type τ. For call-by-name calculi we define:*
$s \leq_{\mathcal{M},R\sigma,\tau} t$, iff for all reduction contexts $R[]_\tau$ and for all substitutions (respectively for call-by-value: all value substitutions) σ that can be applied to s, t without variable capture, we have $R[\sigma(s)]\mathcal{M} \implies R[\sigma(t)]\mathcal{M}$.

Now we can prove a variant of a ciu-theorem for call-by-value and call-by-name calculi. Our ciu-theorem is a bit weaker than the known ones in the literature, which only require to test closed instantiations. On the other hand, it is strong, since for non-deterministic call-by-value calculi, only value-substitutions are required, and it holds for three convergencies.

Theorem 6.9 (IU-Theorem). *Let* CALC *be a non-sharing calculus. Then:*

$$\begin{array}{lll} (1)\ \leq_{\downarrow,R\sigma,\tau} & \subseteq & \leq_{\downarrow,\tau} \\ (2)\ \leq_{\downarrow,R\sigma,\tau} \cap \leq_{\Downarrow,R\sigma,\tau} & \subseteq & \leq_{\downarrow,\tau} \cap \leq_{\Downarrow,\tau} \\ (3)\ \leq_{\Downarrow\!\Downarrow,R\sigma,\tau} & \subseteq & \leq_{\Downarrow\!\Downarrow,\tau} \end{array}$$

Proof. We show that $s \leq_{\mathcal{M},R\sigma,\tau} t$ implies $T(s) \leq_{\mathcal{M},R,\tau} T(t)$ for all $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\!\Downarrow\}$ and all CALC-expressions s, t.
Thus assume $s \leq_{\mathcal{M},R\sigma,\tau} t$. Let R^L be a CALC^L-reduction context, and let $R^L[T(s)]\mathcal{M}$. Let $R' := T^-(R^L)$, $\sigma := \{x_1 \mapsto t'_1, \ldots, x_n \mapsto t'_n\}$, where $x_i = t_i$ are the bindings that affect the hole in R^L, and t'_i is the backtranslated expression t_i, where all bindings are substituted. Note that in the call-by-value case, the terms t'_i are values due to our assumptions for for the `let`-operator. We have $T^-(R^L[T(s)]) = R'[\sigma(s)]$, and hence by Proposition 6.6, we also have $R'[\sigma(s)]\mathcal{M}$. Since $s \leq_{\mathcal{M},R\sigma,\tau} t$, this implies $R'[\sigma(t)]\mathcal{M}$. The same computation for t shows that $T^-(R^L[T(t)]) = R'[\sigma(t)]$ and thus $R^L[T(t)]\mathcal{M}$ using the equivalence of convergence. Now we obtain that for all R^L: $R^L[T(s)]\mathcal{M} \implies R^L[T(t)]\mathcal{M}$, i.e. $T(s) \leq_{\mathcal{M},R,\tau} T(t)$.
Now the context lemma for sharing (Theorem 5.12) in CALC^L shows for $\mathcal{M} \in \{\downarrow, \Downarrow\!\Downarrow\}$ and all expressions s, t of CALC: $s \leq_{\mathcal{M},R\sigma,\tau} t$ implies $T(s) \leq_{\mathcal{M},\tau} T(t)$. Since T is adequate (Proposition 6.7) this also shows for $\mathcal{M} \in \{\downarrow, \Downarrow\!\Downarrow\}$ and all expressions s, t of CALC: $s \leq_{\mathcal{M},R\sigma,\tau} t$ implies $s \leq_{\mathcal{M},\tau} t$.
It remains to show the second claim of the theorem, which follows analogously: the context lemma for sharing (Theorem 5.12) shows for all expressions s, t of

CALC: $s \leq_{\downarrow,R\sigma,\tau} t \wedge s \leq_{\Downarrow,R\sigma,\tau} t$ implies $T(s) \leq_{\downarrow,\tau} T(t) \wedge T(s) \leq_{\Downarrow,\tau} T(t)$. Using adequacy of T this can be equivalently formulated as: $s \leq_{\downarrow,R\sigma,\tau} t \wedge s \leq_{\Downarrow,R\sigma,\tau} t$ implies $s \leq_{\downarrow,\tau} t \wedge s \leq_{\Downarrow,\tau} t$ for all expressions s, t of CALC. □

Corollary 6.10. *If* CALC *is a non-sharing calculus with the beta-reduction rule (in the call-by-value or call-by-name variant, respectively), and if* $(\lambda x.s)\ t \sim s[t/x]$ *for all* s, t*, where* t *must be a value for call-by-value calculi, then in Theorem 6.9 we can replace the "*$\subseteq$*"-relations by "*$=$*".*

Proof. We show only the missing part in the proof of Theorem 6.9.
We use the translation T^- that translates a CALC^L-context C into a term functions of the form $C'[\sigma(.)]$. Using these term functions as observers in the sense of [SSNSS08], it is easy to see that T^- is compositional and convergence equivalent (see Proposition 6.6), and hence fully abstract using Proposition 3.6 in [SSNSS08]. This can be stated as follows. Let $s \leq_{\mathcal{M},C\sigma,\tau} t$ iff for all CALC-contexts C, for all $\mathcal{M} \in \{\downarrow, \Downarrow, \Downarrow\}$. and for all substitutions σ: $C[\sigma(s)]\mathcal{M} \implies C[\sigma(t)]\mathcal{M}$. Full abstraction means that $s \leq_{\mathcal{M},\tau} t \implies T^-(s) \leq_{\mathcal{M},C\sigma,\tau} T^-(t)$ for all CALC^L-terms s, t. But in CALC we have $\leq_{\mathcal{M},C\sigma,\tau} = \leq_{\mathcal{M},\tau}$, since substitutions (value substitutions for call-by-value calculi, respectively) can be simulated by contexts and beta-reduction, and since beta-reduction is correct.

7 Examples for Sharing Calculi and Context Lemmas

7.1 A Recipe for Checking the Preconditions

In order to support the intuition and an easy check of a given calculus, we give some informal hints on the checks of reductions in a given calculus with the goal to verify the Assumptions.

- The calculus must have a lexical scoping discipline and thus provides renamings of bound variables in the usual style.
- Check whether the syntax of the expressions matches the abstract syntax: The binding constructs like λx, patterns or channel prefixes of the π-calculus have an obvious encoding. However, one question is crucial: Are there positions (belonging to the term part) where only certain expressions are permitted, which cannot be enforced by simple types, e.g. only variables, or only abstractions (values), or similar restrictions. Then the "V", "W"-encoding may be used, where the latter requires to identify values as a subclass of the expressions (see below).
- The next step is to look for the positions where a potential reduction may take place, i.e. to distinguish reduction positions from non-reduction positions. Here we mean only the normal-order reduction or the standard reduction of the calculus. Finding reduction positions going top down in the expression must only depend on the already recognized reduction positions.
- Successful reduction of an expression ends with answers, which must be specified. Whether or not an expression is an answer must not depend on non-reduction positions.

- Reduction steps $s \to t$ are usually defined by a set of rules. It has to be checked that expressions at non-reduction positions do not influence the reduction and that they are only transported or copied (perhaps with bound variables renamed), but not modified. Then the strong form of the Reduction Assumption holds. The special form of weakly sharing allows also modifications by renaming of free (local) variables. For checking this, consult the examples in this paper and then the formal definitions.
- If the check for the sharing Reduction Assumption fails, then one may check the Non-Sharing Reduction Assumption in order to obtain the iu-Theorem. If reduction positions are never in the scope of binders, then you may go on; otherwise you have to look deeper into the paper for possible extensions. The rules that fail the sharing Reduction Assumption check must be like the beta-reduction $(\lambda x.s)\ t \to s[t/x]$: To check them, write the rule in its sharing form $(\lambda x.s)\ t \to \texttt{let}\ x = s\ \texttt{in}\ t$, and then check whether this satisfies the sharing reduction assumption, i.e. what happens with the expressions at non-reduction positions.
- The special cases where value positions are in the calculus is a bit subtle. Such a value position must be a syntactically distinguished position, where only values may be placed, but not a general expression. e.g. cells which only can contain values. In non-sharing calculi only for call-by-value calculi the value-positions are permitted.
- For checking a call-by-value extended lambda-calculus, usually there are no value positions, and answers are the values with the exception of variables. If values are defined using constructors like `cons` and `nil`, and a `cons`-expression is in a reduction position, then also the components must be in reduction position. For values with top-binders like an abstraction, the body is unrestricted.

7.2 Strengthening the Context Lemma for Weakly Sharing Calculi

The context lemma for weakly sharing calculi has the slight disadvantage that in addition to all reduction contexts, also all vv-substitutions have to be checked (which are finitely many for fixed expressions s, t), and also that the resulting contextual preorder is differently defined.
This difference can be avoided in most calculi by simulating $s[y_1/x_1, \ldots, y_n/x_n]$ by $(\texttt{letrec}\ x_1 = y_1, \ldots, x_n = y_n\ \texttt{in}\ s)$ or a similar context, which is usually of the form $R[s]$, where R is a reduction context. Note, however, that the relation $\forall \ldots : s[y_1/x_1, \ldots, y_n/x_n] \sim (\texttt{letrec}\ x_1 = y_1, \ldots, x_n = y_n\ \texttt{in}\ s)$, which is sufficient to drop all the ν's, may require an extra proof in the respective calculus.
In the π-calculus, which is weakly sharing, it is also possible to avoid this extra test using the context: $\nu z.(z(x_1, \ldots, x_n).[\cdot] \mid \overline{z}\langle y_1, \ldots, y_n\rangle.0)$, where the variables $y_1, \ldots, y_n$ are not necessarily different. The communication operation for channel z will simulate the vv-substitution $[y_1/x_1, \ldots, y_n/x_n]$.
These observations show that in most weakly sharing calculi the vv-substitutions can be removed from the context lemma and that in addition the contextual

preorder definition can also ignore the vv-substitutions. This is the case for the weakly sharing calculus in [Mor98,MSC99], and also for the π-calculus.

7.3 Examples for Sharing Higher-Order Calculi

Our method to derive (sharing) context lemmas is applicable in lambda-calculi and other higher-order calculi, even with `letrec`, with strict and non-strict reduction provided there are no substituting reduction rules, which is usually only possible, if a form of sharing is permitted by e.g. `let`, `letrec` or explicit substitutions; or an emulation of the sharing with abstraction as e.g. in the letless calculus in [MOW98]. As a general guideline, note that the beta-rule (or similar rules like the substituting case-rule) in general violates our sharing assumptions. The restricted beta-rule $(\lambda x.s)\ y \to s[y/x]$ may be allowed in weakly sharing calculi, provided y is a variable-only position. The rules $(\lambda x.s)\ t \to (\texttt{let}\ x = t\ \texttt{in}\ s)$, $(\texttt{let}\ x = v\ \texttt{in}\ R[x]) \to (\texttt{let}\ x = v\ \texttt{in}\ R[v])$ are permitted in strongly sharing calculi, if the replaced position of x is in a reduction position. Our result can be used for may- as well as must-convergence in its two forms, with or without taking infinite reductions into account.
We mention several sharing calculi, where the result is applicable:
The call-by-need-calculi in [AFM$^+$95,AF97,MOW98] are deterministic, use a `let` to represent sharing, and use a sharing variant of beta-reduction. All the assumptions are satisfied, where the answers according to our definition are of the form `let x1 = t1 in let x2 = t2 in ...  in` λ`x.s`. The context lemma for may-termination holds for these strongly sharing calculi. In [MOW98], there is also a let-less call-by-need calculus that implements sharing by different reduction rules. This calculus is an example of a lambda-calculus that also reduces in abstraction, but in a restricted way. Our context lemma for may-convergence holds also this let-less (strongly sharing) calculus.
The `letrec`-calculi in [AS98,SS07] are deterministic and provide `letrec` for expressing sharing. The context lemma for may-convergence holds for these strongly sharing calculi. The non-deterministic call-by-need calculi in [KSS98,Man05] provide a `let` and a non-deterministic choice. The assumptions are satisfied, where UNWIND is deterministic. Context lemmas for may-termination as well as must-termination for these strongly sharing calculi hold. Note that [KSS98] uses total must-divergence, and makes no use of a context lemma, whereas the calculus in [Man05] did not treat must-divergence.
The call-by-need calculus in [MSC99] with letrec, choice, case and constructors uses may- and total must-convergence, and satisfies our sharing assumptions. The calculus is weakly sharing since the beta-rule-variant and the case-rule use vv-substitutions, and since the arguments in applications as well as the arguments in constructor expressions $(c\ x_1 \ldots x_n)$ may only be occupied by variables. The context lemmas for may- , must and total must-convergence hold in this calculus, though only the may- and total must-context lemmas are used.
The call-by-need calculi in [SSS08,Mor98] provide amb, letrec, case and constructors. They satisfy our sharing criteria, where the first is strongly, and the second is weakly sharing. UNWIND and normal-order reduction are non-deterministic.

Our results confirm the respective context lemmas, and also show a new one for the call-by-need variant in [Mor98], since there is no proof of a context lemma for total must-convergence in [Mor98]. Our method to derive context lemmas is also applicable for the fair (i.e. using resources by annotations) variant of the amb-calculi in [Mor98,SSS08], where the encoding of $(\mathtt{amb}_{m,n}\ s\ t)$ can be done by using infinitely many operators $\mathtt{amb}_{m,n}$.

Process calculi like the π-calculus are in the scope of our method. The result is that process contexts (no vv-substitutions required) are sufficient to check observational equivalence of processes w.r.t. may- and must-convergence. The call-by-value concurrent process calculus $\lambda(\mathsf{fut})$ in [NSSSS07] has a sharing variant of beta-reduction, and is derived from a calculus with beta-reduction (see [NSS06]), which has mutable cells, and a non-deterministic reduction. The sharing variant in [NSSSS07] satisfies our assumptions for a strongly sharing calculus, and requires two basic types in our type system. Note that UNWIND is nondeterministic. After some preprocessing is done, the context lemmas for may- and must-convergence for expressions can be derived from our results. Note that the context lemmas for processes in $\lambda(\mathsf{fut})$ are trivial, since all process contexts are reduction contexts in $\lambda(\mathsf{fut})$.

8 Conclusion

We have exhibited a broad class of higher-order calculi including typed and untyped, deterministic and non-deterministic extended lambda-calculi with a form of sharing; and higher-order process calculi, where the generic context lemmas for may- as well as must-convergence can be derived, and hence also used by only verifying our assumptions. Three natural assumptions must hold in the given sharing calculus in order to obtain the validity of the generic context lemmas. If the calculus in question does not satisfy the sharing assumptions, then it is very likely that it satisfies the non-sharing assumptions and thus the generic (c)iu-theorems can be applied. This not only paves the way for analyzing already described calculi, but also future calculi modeling programming languages and communicating processes. The use of abstract syntax, and the three natural sharing assumptions and the two non-sharing assumptions on reduction position, answers, and reductions, show that seemingly very different program calculi have a lot in common and that certain aspects like context lemmas can be derived in a general way.

All of the following are left for future research: We did not investigate the connection to special rule formats. Also, it is likely that our method can be extended to also prove stronger ciu-theorems. We also did not investigate the combination of non-sharing calculi with letrec-constructs.

Acknowledgement We thank the anonymous referees for their valuable comments and suggestions.

References

ACCL91. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J Lévy. Explicit substitutions. *J. Funct. Programming*, 1(4):375–416, 1991.

AF97. Z. M. Ariola and M Felleisen. The call-by-need lambda calculus. *J. Funct. Programming*, 7(3):265–301, 1997.

AFM+95. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *POPL'95*, pages 233–246, San Francisco, California, 1995. ACM Press.

AS98. Zena M. Ariola and Amr Sabry. Correctness of monadic state: An imperative call-by-need calculus. In *POPL 98*, pages 62–74, 1998.

Bar84. H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, New York, 1984.

CHS05. Arnaud Carayol, Daniel Hirschkoff, and Davide Sangiorgi. On the representation of McCarthy's amb in the pi-calculus. *Theoret. Comput. Sci.*, 330(3):439–473, 2005.

FG96. Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *POPL'96*, pages 372–385. ACM Press, 1996.

FH92. Matthias Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoret. Comput. Sci.*, 103:235–271, 1992.

FM01. Jonathan Ford and Ian A. Mason. Operational techniques in PVS - a preliminary evaluation. *Electron. Notes Theor. Comput. Sci.*, 42, 2001.

FM03. Jonathan Ford and Ian A. Mason. Formal foundations of operational semantics. *Higher Order Symbol. Comput.*, 16(3):161–202, 2003.

Gor99. Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Theoret. Comput. Sci.*, 228(1-2):5–47, October 1999.

How89. D. Howe. Equality in lazy computation systems. In *4th IEEE Symp. on Logic in Computer Science*, pages 198–203, 1989.

How96. D. Howe. Proving congruence of bisimulation in functional programming languages. *Inform. and Comput.*, 124(2):103–112, 1996.

JM97. T. Jim and A.R. Meyer. Full abstraction and the context lemma. *SIAM J. Comput.*, 25(3):663–696, 1997.

KSS98. Arne Kutzner and Manfred Schmidt-Schauß. A nondeterministic call-by-need lambda calculus. In *International Conference on Functional Programming 1998*, pages 324–335. ACM Press, 1998.

Lan96. C. Laneve. On testing equivalence: May and must testing in the join-calculus. Technical Report Technical Report UBLCS 96-04, University of Bologna, 1996.

Las98. Søren Bøgh Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Faculty of Science, University of Aarhus, 1998.

Man05. Matthias Mann. Congruence of bisimulation in a non-deterministic call-by-need lambda calculus. *Electron. Notes Theor. Comput. Sci.*, 128(1):81–101, 2005.

Mil77. R. Milner. Fully abstract models of typed λ-calculi. *Theoret. Comput. Sci.*, 4:1–22, 1977.

Mil99. Robin Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge university press, 1999.

Mor68. J.H. Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, MIT, 1968.

Mor98. A. K. D. Moran. *Call-by-name, call-by-need, and McCarthys Amb*. PhD thesis, Dept. of Comp. Science, Chalmers university, Sweden, 1998.

MOW98. John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *J. Funct. Programming*, 8:275–317, 1998.

MS99. A. K. D. Moran and D. Sands. Improvement in a lazy context: An operational theory for call-by-need. In *POPL 1999*, pages 43–56. ACM Press, 1999.

MSC99. Andrew K. D. Moran, David Sands, and Magnus Carlsson. Erratic fudgets: A semantic theory for an embedded coordination language. In *Coordination '99*, volume 1594 of *Lecture Notes in Comput. Sci.*, pages 85–102. Springer-Verlag, 1999.

MSS06. Matthias Mann and Manfred Schmidt-Schauß. How to prove similarity a precongruence in non-deterministic call-by-need lambda calculi. Frank report 22, Inst. f. Informatik, J.W.Goethe-University, Frankfurt, January 2006.

MST96. Ian Mason, Scott F. Smith, and Carolyn L. Talcott. From operational semantics to domain theory. *Inform. and Comput.*, 128:26–47, 1996.

MT91. Ian Mason and Carolyn L. Talcott. Equivalence in functional languages with effects. *J. Funct. Programming*, 1(3):287–327, 1991.

NSS06. Joachim Niehren, Jan Schwinghammer, and Gert Smolka. A concurrent lambda calculus with futures. *Theoret. Comput. Sci.*, 364(3):338–356, November 2006.

NSSSS07. Joachim Niehren, David Sabel, Manfred Schmidt-Schauß, and Jan Schwinghammer. Observational semantics for a concurrent lambda calculus with reference cells and futures. *Electron. Notes Theor. Comput. Sci.*, 173:313–337, 2007.

Ong93. C.-H. L. Ong. Non-determinism in a functional setting. In *Proc. 8th IEEE Symposium on Logic in Computer Science (LICS '93)*, pages 275–286. IEEE Computer Society Press, 1993.

Plo76. G.D. Plotkin. A powerdomain construction. *SIAM J. Comput.*, 5(3):452–487, 1976.

PS98. A.M. Pitts and Ian Stark. *Operational Reasoning for functions with local state*, pages 227–273. Publications of the Newton Institute. Cambridge university press, 1998.

SS03. Manfred Schmidt-Schauß. FUNDIO: A lambda-calculus with a `letrec`, `case`, constructors, and an IO-interface: Approaching a theory of `unsafePerformIO`. Frank report 16, Inst. f. Informatik, J.W.Goethe-University, Frankfurt, 2003.

SS07. Manfred Schmidt-Schauß. Correctness of copy in calculi with letrec. In *Term Rewriting and Applications (RTA-18)*, volume 4533 of *Lecture Notes in Comput. Sci.*, pages 329–343. Springer, 2007.

SSNSS08. Manfred Schmidt-Schauß, Joachim Niehren, Jan Schwinghammer, and David Sabel. Adequacy of compositional translations for observational semantics. Frank report 33, Inst. f. Informatik, J.W.Goethe-University, Frankfurt, 2008.

SSS08. David Sabel and Manfred Schmidt-Schauß. A call-by-need lambda-calculus with locally bottom-avoiding choice: Context lemma and correctness of transformations. *Math. Structures Comput. Sci.*, 18(03):501–553, 2008.

SSSS08. Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 18(04):503–551, 2008.

SW01. D. Sangiorgi and D. Walker. *The π-calculus: a theory of mobile processes.* Cambridge university press, 2001.