To be published in the Proceedings of AMAST'97, Sydney, Dec 13–17 1997

The Update Calculus (extended abstract)

Joachim Parrow* Björn Victor†

September 1, 1997

Abstract

In the *update calculus* concurrent processes can perform update actions with side effects, and a scoping operator can be used to control the extent of the update. In this way it incorporates fundamental concepts both from imperative languages or concurrent constraints formalisms, and from functional formalisms such as the λ- and π-calculi. Structurally it is similar to but simpler than the π-calculus; it has only one binding operator and a symmetry between input and output. We define the structured operational semantics and the proper bisimulation equivalence and congruence, and give a complete axiomatization. The π-calculus turns out to be an asymmetric subcalculus.

1 Introduction

Theory of concurrent computation is a diverse field where many different approaches have been proposed and no consensus has emerged on the best paradigms. In this paper we take a step towards unifying two seemingly contradictory schools of thought: *global* vs *local* effects of concurrent actions. We define a calculus where an explicit scope operator regulates the extent of an action effect. The calculus is structurally simple and incorporates fundamental concepts both from imperative languages or concurrent constraints formalisms, and from functional formalisms such as the λ- and π-calculi.

*Dept. of Teleinformatics, Royal Institute of Technology, Electrum 204, S-164 40 Kista, Sweden. Work partially supported by ESPRIT BRA project No. 8130: LOMAPS, The Human Capital and Mobility Project EXPRESS, and the working group CONFER-2.

†Dept. of Computer Systems, Uppsala University, Box 325, S-751 05 Uppsala, Sweden.

A concurrent system consists of parallel processes that perform *actions*. "Globalists" and "localists" differ on how the effects of actions propagate through the system. For globalists, which typically deal with imperative languages or concurrent constraint formalisms, an action may update a global variable or unify a logical variable. In this case the effect of the action is immediately present in an unbounded part of the system, even in those processes which do not take part in the action. Formal semantics and proof methods using this paradigm can be found in, e.g., [CM88, BKS88, MP91] and normally use a separate "global state" encompassing all processes.

For a localist, on the other hand, an action may be to call a subroutine with actual parameters replacing formal parameters, or to send a message from one agent to another. The action is local to the interacting parties, and these must explicitly communicate with the rest of the system to spread the effect. Localists typically work with functional languages and formalisms such as the λ-calculus or the π-calculus [Bar84], [MPW92, Mil91, San92, PS95], [San96, San95a, PS96, Dam93]. Since these do not use a separate global state, proof methods are often conceptually simpler than for global formalisms, at the expense of some loss of expressiveness: in some ways the global paradigm is closer to many high-level programming languages.

It is well known that e.g. message-passing and shared variables can emulate each other. In general, a global state can be interpreted as just another process storing values and communicating with other processes; conversely, local interaction between processes is achieved by using a designated part of the global state as a temporary storage area. The global and local views do not entail absolute differences of expressiveness; rather, they differ in what phenomena can be modelled in a convenient and clear way. This has not prevented an occasionally fierce debate on which paradigm should be regarded as the most fundamental one.

The contribution of the present paper, the *update calculus* (or *up-calculus*) is a step towards a unifying framework for actions with local and global effect. There will be *update* actions written $[y/z]$, meaning "z should everywhere be replaced by y" with a potentially unbounded extent, and a *scope* operator (z) explicitly controlling the extent of updates of z. By varying the scope we can localize the effects. There is no separate global state, instead the update actions affect all processes directly. In this way we mitigate the global vs local controversy and demonstrate how both are special cases of a more basic concept.

It is interesting to compare the up-calculus with the π-calculus, which is riddled by a few curious asymmetries. In π there are two binding operators, input binding $a(x)\,.\,P$ meaning "receive something for x before continuing with P" and restriction $(\boldsymbol{\nu}x)P$ meaning "x in P will always be distinct from other names". Other non derivable binding operators, such as delay input $a(x)P$, which does not imply a temporal precedence between the instantiation of x and the rest of P, have also been suggested. And since input actions must be accompanied by a binding there is a fundamental asymmetry between input and output. In the up-calculus there is only one binding operator, treatment of input and output is symmetric, and the π-calculus can be recovered as a subcalculus (formed by requiring all input actions to be enclosed by a scope as tightly as possible). This throws light on how some of the different versions of the π-calculus interrelate.

To give a taste of the up-calculus, consider two agents $\overline{a}y \,.\, P$ and $az \,.\, Q$. The first emits a name y along a port a, the second receives something for z along a. In the latter the scope of z is not specified, so an interaction between the two agents will result in a global update action:

$$\overline{a}y \,.\, P \mid az \,.\, Q \quad \xrightarrow{[y/z]} \quad P \mid Q$$

The fact that this action is "global" can be seen if we include a further parallel component, R, not part in the interaction:

$$\overline{a}y \,.\, P \mid az \,.\, Q \mid R \quad \xrightarrow{[y/z]} \quad P \mid Q \mid R$$

The update action will affect R as much as P and Q. The scope operator (z) determines the extent of the update action:

$$(z)(\overline{a}y \,.\, P \mid az \,.\, Q \mid R) \quad \xrightarrow{\mathbf{1}} \quad (P \mid Q \mid R)\{y/z\}$$

Here $\mathbf{1}$ is an inert action, without any updating or communicating effect (cf τ in CCS) and the substitution y for z is performed on the right hand side. A further parallel component S outside the scope of z will not be affected by the substitution or update:

$$(z)(\overline{a}y \,.\, P \mid az \,.\, Q \mid R) \mid S \quad \xrightarrow{\mathbf{1}} \quad (P \mid Q \mid R)\{y/z\} \mid S$$

We claim that this entails a clean way to model assignment to shared structures and resolution in concurrent constraint languages. For example, in a concurrent constraint formalism there will be existentially quantified logical variables with well defined scopes:

$$\exists z(P \wedge Q \wedge R) \wedge S$$

Resolution of the conjunct P may result in a new fact $z = y$, in other words $P \to P' \wedge z = y$. Then the whole expression will reduce:

$$\exists z(P \wedge Q \wedge R) \wedge S \quad \rightarrow \quad (P' \wedge Q \wedge R)\{y/z\} \wedge S$$

The update calculus is particularly well suited to model such reductions, as will be demonstrated in forthcoming papers. Of course, concurrent constraints can also be encoded in π by explicitly representing the logical variables as registers (see [VP96]). Proponents of concurrent constraints would presumably rather encode the π-calculus in one of their favourite formalisms. The purpose of the present paper is to demonstrate that both are particular cases of something more basic.

In the next section we present the syntax and semantics of the up-calculus. We also show how the π-calculus is isomorphic to an asymmetric subcalculus. In Section 3 we define the proper notion of bisimulation congruence, called hyperequivalence. It is strictly stronger than the bisimulation equivalences previously investigated for the π-calculus, and turns out to be the *only* reasonable candidate for a bisimulation-based congruence. In Section 4 we give a set of algebraic equalities which constitute a complete axiomatization for it. Section 5 contains conclusions and directions for further research.

2 Basic Definitions

2.1 Syntax

We begin by presupposing an infinite set $\mathcal{N}$ of *names* ranged over by $a, b, \ldots, y, z$. As in the π-calculus names will be used to represent all of communication ports, values, and placeholders for ports and values. A more refined version of the calculus would use a type system to distinguish between those, but that is not necessary for the development in the present paper.

Definition 1

The free actions*, ranged over by $\alpha, \beta, \ldots$ are defined by*

$$\begin{array}{rll} \alpha ::= & ax & \text{(Input)} \\ & \overline{a}x & \text{(Output)} \\ & [x/y] & \text{(Update)} \end{array}$$

An input action ax means "input an object along the port a and replace x with that object." Contrary to the situation in the π-calculus this action does not bind x. In other words, the scope of x is not bounded by the action. The output action $\overline{a}x$ is familiar from the π-calculus and means "output the object x along the port a. The update action $[x/y]$ signifies an obligation to replace all y by x. It is not present in the π-calculus.

An action $[x/x]$, updating x to itself, is called an "inert" update and is often written $\mathbf{1}$, since the choice of x in $[x/x]$ will never matter. In all actions above (except in $\mathbf{1}$) x is called the *(free) object* of the action, written $\mathrm{obj}(\alpha)$. The inert update $\mathbf{1}$ has no object. In the input and output actions the name a is called the *subject* of the action, written $\mathrm{subj}(\alpha)$.

Definition 2 *The* agents*, ranged over by $P, Q, \ldots$, are defined by*

$$\begin{array}{rlll} P ::= & \mathbf{0} & & \text{(Inaction)} \\ & \alpha \,.\, Q & & \text{(Prefix)} \\ & Q + R & & \text{(Summation)} \\ & Q \mid R & & \text{(Composition)} \\ & (x)Q & & \text{(Scope)} \end{array}$$

The Prefix form means "first do α and then behave as Q." In a sense to be made precise below, update prefixes like $[x/y] \,.\, Q$ can be regarded as derived forms and are not strictly necessary in the definition, but admitting them simplifies the presentation. Summation is alternative choice and Composition is parallel execution, just as in the π-calculus. In examples to come we will often omit a trailing $\mathbf{0}$, therefore α will stand for the agent $\alpha \,.\, \mathbf{0}$ when no confusion is possible.

The Scope $(x)Q$ restricts the scope of the name x to Q. In a way this is a common denominator of π-calculus restriction $(\boldsymbol{\nu} x)Q$ and input binding $a(x) \,.\, Q$. Restriction implies that no name will ever replace the bound names, while input binding requires that the bound name in immediately replaced by something received along a. Scope neither forbids nor insists on a replacement; it merely states the extent of the name. As we will see, restriction and input binding can be recovered as special cases of Scope.

The name x is said to be *bound* in $(x)P$. The *free names* in P, denoted $\mathrm{fn}(P)$, are the names in P with a non-bound occurrence. $\mathrm{n}(P)$ denotes all names occurring in P. As usual we will not distinguish between alpha-variants of agents, i.e., agents differing only in the choice of bound names.

Definition 3 *The* substitution *of x for y in P, written $P\{x/y\}$, is the agent obtained by replacing all y by x in P, renaming bound names as necessary to avoid captures. We use σ to range over substitutions.*

2.2 Semantics

We first define a structural congruence to equate agents that we never want to distinguish for any reason related to semantics, and using that we give the transition rules.

Definition 4 *The* structural congruence, $\equiv$, *between agents is the least congruence satisfying the abelian monoid laws for Summation and Composition (associativity, commutativity and* $\mathbf{0}$ *as identity), and also the scoping laws*

$$(x)\mathbf{0} \equiv \mathbf{0}, \quad (x)(y)P \equiv (y)(x)P, \quad (x)(P+Q) \equiv (x)P + (x)Q$$

and also the scope extension *law* $P \mid (z)Q \equiv (z)(P \mid Q)$ *where* $z \notin \mathrm{fn}(P)$*.*

Note that if $z \notin \mathrm{fn}(P)$ then $(z)P \equiv (z)(P \mid \mathbf{0}) \equiv P \mid (z)\mathbf{0} \equiv P \mid \mathbf{0} \equiv P$. Also note that after an alpha-conversion, any scope (x) not under a prefix can be pulled to the top level of an agent while preserving structural congruence.

The actions in the transition may be free or bound:

Definition 5 *A* bound *action is of the form* $(x)\alpha$ *where* x *is the object of* α*. The* actions, *ranged over by* γ*, consist of the free actions and the bound actions. A* transition *is of the kind* $P \xrightarrow{\gamma} Q$*.*

The meaning of a bound action is simply that the object is emitted out of its scope and must therefore not be confused with any similar name in the environment. There is no bound action corresponding to $\mathbf{1}$ since $\mathbf{1}$ has no object. We write $\mathrm{n}(\gamma)$, $\mathrm{fn}(\gamma)$ and $\mathrm{bn}(\gamma)$ to mean all names, free names, and bound names respectively occurring in γ. For the inert update we define $\mathrm{n}(\mathbf{1}) = \emptyset$.

Definition 6 *The* transitions $P \xrightarrow{\gamma} Q$ *is the least family of transitions satisfying the laws in Table 1. In this definition structurally equivalent agents are considered the same, i.e., if* $P \equiv P'$ *and* $Q \equiv Q'$ *and* $P \xrightarrow{\gamma} Q$ *then also* $P' \xrightarrow{\gamma} Q'$*.*

No premise of any rule in Table 1 has a bound action, and the only rule for generating bound actions in transitions is OPEN. Using structural congruence, pulling the relevant scope to top level, we can still infer e.g. $P \mid (x)ax\,.\,Q \xrightarrow{(x)ax} P \mid Q$ using PREF and OPEN (provided $x \notin \mathrm{fn}(P)$, otherwise an alpha-conversion is necessary). Note also that α cannot be $\mathbf{1}$ (or $[z/z]$) in OPEN, since $\mathbf{1}$ has no object.

It is illuminating to consider the COM and SCOPE rules in particular. COM results in an update action, signifying a requirement to replace z by y everywhere. The SCOPE rule produces the substitution and also changes the action to an inert action, so that no z outside the scope will be affected.

PREF $\dfrac{-}{\alpha\,.\,P \xrightarrow{\alpha} P}$	SUM $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
PAR $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	COM $\dfrac{P \xrightarrow{az} P', \quad Q \xrightarrow{\overline{a}y} Q'}{P \mid Q \xrightarrow{[y/z]} P' \mid Q'}$
SCOPE $\dfrac{P \xrightarrow{[y/z]} P',\ y \neq z}{(z)P \xrightarrow{\mathbf{1}} P'\{y/z\}}$	PASS $\dfrac{P \xrightarrow{\alpha} P',\ z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha} (z)P'}$
OPEN $\dfrac{P \xrightarrow{\alpha} P', \begin{array}{l}\mathrm{obj}(\alpha) = z,\\ \mathrm{subj}(\alpha) \neq z\end{array}}{(z)P \xrightarrow{(z)\alpha} P'}$	

Table 1: The up-calculus: Laws of action.

The gain in expressiveness over the π-calculus lies in this possibility to decouple the input action from the binding of the input object. As an example we can define an agent $(z)(az\,.\,\mathbf{0} \mid P)$ where the input is *delayed*, in the sense that it may happen at any time:

$$\begin{array}{llll}
(z)(az\,.\,\mathbf{0} \mid P) & \xrightarrow{\gamma} & (z)(az\,.\,\mathbf{0} \mid P') & \text{if } P \xrightarrow{\gamma} P' \text{ and } z \notin \mathrm{n}(\gamma) \\
(z)(az\,.\,\mathbf{0} \mid P) & \xrightarrow{(z)az} & P & \text{for any } P
\end{array}$$

This agent has all transitions from P not involving z, and in any derivative so obtained it has the bound input transition $(z)az$.

As a final example assume $z \notin \mathrm{fn}(P)$ and consider the agent

$$(z)az\,.\,bz\,.\,P$$

There are *two* inputs with object z under the same scope. The agent can interact with an environment that emits something along a,

$$(z)az\,.\,bz\,.\,P \mid \overline{a}u\,.\,Q \quad \xrightarrow{\mathbf{1}} \quad bu\,.\,P \mid Q$$

and will proceed input something for u along b. In other words, the object to be *replaced* in an interaction along b is first transmitted along a. This has no immediate counterpart in the π-calculus.

Note in passing that update prefixes can be regarded as derived forms since $[y/z]\,.\,P$ has exactly the same transitions as $(a)(\overline{a}y\,.\,\mathbf{0} \mid az\,.\,P)$ when $a \notin \mathrm{fn}(P)$. (In the same way the τ prefix can be regarded as derived in CCS and in the π-calculus.)

2.3 Subcalculi

Consider a version of the ordinary monadic π-calculus [MPW92] which differs from the update calculus in the following way: There are neither input nor update prefixes, instead there is a *bound input* prefix written $a(x)\,.\,P$ with the transition $a(x)\,.\,P \xrightarrow{(x)ax} P$. There is no scoping operator but a related *restriction* operator $(\boldsymbol{\nu}x)$, inheriting the laws PASS and OPEN. There is no SCOPE law since there are no update actions; instead the COM law is

$$\frac{P \xrightarrow{(z)az} P', \quad Q \xrightarrow{\overline{a}y} Q'}{P \mid Q \xrightarrow{\mathbf{1}} P'\{y/z\} \mid Q}$$

(The notation for actions in π is usually slightly different. Bound actions $(z)az$ and $(z)\overline{a}z$ are usually written $a(z)$ and $\overline{a}(z)$, and the inert action $\mathbf{1}$ is written τ.)

The so defined π-calculus is isomorphic to a subcalculus of the up-calculus, formed by omitting update prefixes and by requiring that all input prefixes $ax\,.\,P$ occur immediately under a scope (x) of the object in the input prefix. Call this subcalculus the up$_\pi$-calculus. The isomorphy $\leftrightarrow$ between up$_\pi$ and π is simply the homomorphic extension of

$$\begin{array}{lll} (x)ax\,.\,P & \leftrightarrow & a(x)\,.\,P \\ (z)P & \leftrightarrow & (\boldsymbol{\nu} z)P \quad \text{if } P \text{ is not a free input prefix} \end{array}$$

The rationale behind this is as follows. Consider a scope $(x)Q$ in up$_\pi$. If it is immediately over an input prefix, $Q = ax\,.\,P$, then before anything happens in P something must be received along a to replace x in P, precisely as the effect of a π-calculus input prefix $a(x)\,.\,P$. On the other hand, if it is not immediately over an input prefix, $Q \neq ax\,.\,P$, then there cannot be any input prefix inside Q for that particular x. Because of the syntactic restrictions in up$_\pi$, a free input ax not at the top level of Q must be enclosed by another scope (x), making the object different from the x in $(x)Q$. So this x cannot occur in an input and there are no updates, therefore x will remain distinct from other names throughout execution of the agent. This is precisely the effect of $(\boldsymbol{\nu} x)Q$.

More formally, the result is (where $\circ$ is composition of relations):

Theorem 1 *For P a π-calculus agent and Q an up$_\pi$-calculus agent,*

1. *If $P \leftrightarrow Q$ and $P \xrightarrow{\gamma} P'$ then $Q \xrightarrow{\gamma} Q'$ and $P' \equiv \circ \leftrightarrow \circ \equiv Q'$.*
2. *If $P \leftrightarrow Q$ and $Q \xrightarrow{\gamma} Q'$ then $P \xrightarrow{\gamma} P'$ and $P' \equiv \circ \leftrightarrow \circ \equiv Q'$.*

The πI-calculus by Sangiorgi [San95b] is a subset of the π-calculus formed by requiring all output prefixes $\overline{a}x$ to occur immediately below a restriction $(\boldsymbol{\nu} x)$ of the object. Therefore that calculus is isomorphic to the subcalculus of up$_\pi$ requiring that *all* prefixes — not merely input prefixes — occur under a scope of the object. As Sangiorgi has remarked the πI-calculus has a pleasing symmetry between input and output actions. The up-calculus has the same kind of symmetry, achieved by extending rather than restricting the π-calculus.

3 Equivalence and Congruence

Our notion of equivalence uses the well known idea of bisimulation. In essence, for two agents to be bisimilar, each transition from one agent must be mimicked from the other, reaching agents which are again bisimilar. In calculi where transitions can contain placeholders (such as variables) for values, care has been taken to properly define what "again bisimilar" means. For example, in the π-calculus there are varieties such as *early*, *late* [MPW92], and *open* [San96] bisimulation. These differ in the degree of universal quantification over the names, and each of these lead to a different congruence.

In the up-calculus this diversity is mercifully absent; there appears to be only one sensible bisimulation congruence. The reason is that the update actions allow us to formulate more powerful contexts than what is possible in the π-calculus. Update actions can enforce a global substitution of *any* name, and therefore the congruence must be closed under arbitrary substitutions in a way which will be made precise below.

Let us first consider the proper notion of bisimulation.

Definition 7 *A* simulation *is a binary relation $\mathcal{S}$ between agents such that $P\ \mathcal{S}\ Q$ implies:*

If $P \xrightarrow{\gamma} P'$ *with* $\mathrm{bn}(\gamma) \notin \mathrm{fn}(Q)$ *then*

$$Q \xrightarrow{\gamma} Q' \text{ and } \begin{cases} P'\ \mathcal{S}\ Q' & \text{if } \gamma \text{ is an input or an output} \\ P'\{y/z\}\ \mathcal{S}\ Q'\{y/z\} & \text{if } \gamma \text{ is } [y/z] \text{ or } (y)[y/z] \end{cases}$$

A bisimulation *is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are simulations. P is bisimilar to Q, written $P \mathrel{\dot\sim} Q$, if $P\ \mathcal{S}\ Q$ for some bisimulation $\mathcal{S}$.*

The interesting point in this definition is the treatment of update actions. An update $[y/z]$ represents an obligation to replace z by y everywhere. Therefore, if γ above is such an update, it only makes sense to relate P' and Q' when the substitution $\{y/z\}$ has been performed. Consider for example the following, where the objects of input and output prefixes are omitted for the sake of brevity (so $y\,.\,P$ means $yw\,.\,P$ for some fresh w, etc.)

$$\begin{array}{rcl} y \mid \overline{z} & \dot\sim & y\,.\,\overline{z} + \overline{z}\,.\,y \\ y \mid \overline{y} & \not\dot\sim & y\,.\,\overline{y} + \overline{y}\,.\,y \\ y \mid \overline{y} & \dot\sim & y\,.\,\overline{y} + \overline{y}\,.\,y + \mathbf{1} \\ [y/z]\,.\,(y \mid \overline{z}) & \dot\sim & [y/z]\,.\,(y\,.\,\overline{z} + \overline{z}\,.\,y + \mathbf{1}) \end{array}$$

The third example implies the fourth even though $y \mid \overline{z}$ is not bisimilar to $y\,.\,\overline{z} + \overline{z}\,.\,y + \mathbf{1}$. In fact, we also have

$$[y/z]\,.\,y\,.\,\mathbf{0} \mathrel{\dot\sim} [y/z]\,.\,z\,.\,\mathbf{0}$$

Bisimilarity is not a congruence. An easy example is that $y \mid \overline{z}$ is bisimilar to $y\,.\,\overline{z}+\overline{z}\,.\,y$, but prefixing both with $[y/z]$ will result in non bisimilar agents. We therefore look for a congruence included in bisimilarity. This is achieved by closing the definition of bisimulation under arbitrary substitutions.

Definition 8 *A* hyperbisimulation *is a substitution closed bisimulation, i.e., a bisimulation $\mathcal{S}$ with the property that $P \mathcal{S} Q$ implies $P\{x/y\} \mathcal{S} Q\{x/y\}$ for any x, y. Two agents P and Q are* hyperequivalent, *written $P \sim Q$, if they are related by a hyperbisimulation.*

Theorem 2 *Hyperequivalence is a congruence.*

PROOF: Through a case analysis over the operators in the up-calculus. All cases are simple. □

Note that the special case for update actions in the definition of bisimulation is significant even though hyperbisimulation closes under all substitutions! The point is that $P' \mathcal{S} Q'$ would be a too *strong* requirement if $\gamma = [y/z]$, since it would require bisimulation also under the substitutions which do not replace z by y.

It is interesting to relate hyperequivalence to open equivalence, which is a congruence in the π-calculus and hence also in up_π. For a definition of open we refer to [San96], but even without the details of the definition a reader might appreciate the following arguments. Broadly speaking, for the purpose of hyperbisimulation, *all* names are considered universally quantified *after every transition.* That is, if we for a moment disregard the update actions (which are not part of the π-calculus), hyperbisimulation requires, if $P \mathcal{S} Q$ and $P \xrightarrow{\gamma} P'$, that $Q \xrightarrow{\gamma} Q'$ and $P'\sigma \mathcal{S} Q'\sigma$ *for all substitutions* σ. For late equivalence the corresponding requirement would be *for all substitutions σ involving a bound input name in γ.* For open equivalence the requirement would be *for all substitutions σ except those involving a bound output name in γ.*

In the π-calculus without the restriction operator the definitions of open and hyper would coincide. But with it hyper is strictly stronger, as evidenced by the up_π-calculus agents

$$(x)(y)\overline{a}x\,.\,\overline{a}y\,.\,(x \mid \overline{y}) \qquad \text{and} \qquad (x)(y)\overline{a}x\,.\,\overline{a}y\,.\,(x\,.\,\overline{y} + \overline{y}\,.\,x)$$

These are open equivalent but not hyperequivalent. To see this, after two bound output transitions the agents become

$$x \mid \overline{y} \qquad \text{and} \qquad x\,.\,\overline{y} + \overline{y}\,.\,x$$

and these are not bisimilar under the substitution $\{x/y\}$. Substitutions of names emitted as bound output objects are significant for hyper but not for open.

When we consider arbitrary up-contexts, this example can be used to demonstrate that open equivalence, though a congruence in π, is not a congruence in the up-calculus (which has more contexts and therefore puts a heavier burden on a congruence). Put the two agents in parallel with $(z)az\,.\,az$ (note that this agent is not in up_π because of the second non-bound input prefix az). There is then one more $\mathbf{1}$ transition from

$$(z)az\,.\,az \mid (x)(y)\overline{a}x\,.\,\overline{a}y\,.\,(x \mid \overline{y})$$

than from

$$(z)az\,.\,az \mid (x)(y)\overline{a}x\,.\,\overline{a}y\,.\,(x\,.\,\overline{y} + \overline{y}\,.\,x)$$

In essence, the parallel context $(z)az\,.\,az$ will enforce a unification of any two names emitted along a.

Even though open is not a congruence there might conceivably exist some other congruences larger than hyper, but our next theorem demonstrates that this is impossible. This result confirms that hyperequivalence is the only "interesting" congruence.

Theorem 3 *Hyperequivalence is the largest congruence in bisimilarity.*

We sketch the proof below. We construct a context which given a set of names N can perform any substitutions of names in N, and capture all bound names emitted on a name in N.

Let N be any finite set of names, and $\mathcal{F}_N = \{s_{xy}, r_x, q_x, e, d, u : x, y \in N\}$ consist of fresh names. We write $N + w$ for $N \cup \{w\}$ and $N - w$ for $N \setminus \{w\}$. Define $S_N^0 = \mathbf{0}$, and

$$\begin{aligned} S_N^k \quad = \quad & \sum_{\substack{x,y\,\in\,N \\ x \neq y}} s_{xy}\,.\,[x/y]\,.\,d\,.\,S_{N-y}^{k-1} \\ + \quad & \sum_{x \in N} r_x\,.\,(u)xu\,.\,\overline{e}\,u\,.\,S_{N+u}^{k-1} \\ + \quad & \sum_{x \in N} q_x\,.\,\overline{x}\,u\,.\,d\,.\,S_{N+u}^{k-1} \end{aligned}$$

The main Lemma states that if two agents are bisimilar in an S_N^k context, then they must also be hyperequivalent:

Lemma 4 $\mathcal{S} \equiv \{(P, Q) : \forall N : \mathrm{fn}(P, Q) \subseteq N, \forall k : S_N^k \mid P \dot{\sim} S_N^k \mid Q\}$ *is a hyperbisimulation.*

Now Theorem 3 is a consequence of Lemma 4, using S_N^k contexts: If $\mathcal{R}$ is a bisimulation congruence, $(P, Q) \in \mathcal{R}$ implies that for all contexts $\mathcal{C}$, $(\mathcal{C}[P], \mathcal{C}[Q]) \in \mathcal{R}$, so consider the family of contexts $\mathcal{C}[\bullet] = S_N^k \mid \bullet$, for all k, $N = \mathrm{fn}(P, Q)$, and then by Lemma 4, $\mathcal{R}$ is a hyperbisimulation. Thus there can not exist a larger (or "weaker") bisimulation congruence than hyperequivalence for the up-calculus.

4 Axiomatization

We now introduce two additional operators in the up-calculus: if P is an agent then so are $[x = y]P$, a *match*, and $[x \neq y]P$, a *mismatch*, with the laws of action

$$\text{MATCH}\ \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'} \qquad \text{MISM}\ \frac{P \xrightarrow{\alpha} P', x \neq y}{[x \neq y]P \xrightarrow{\alpha} P'}$$

These operators are found in earlier work on the π-calculus; match has been used for a sound axiomatization of late congruence [MPW92] as well as a complete axiomatization of open equivalence [San96], and mismatch has been used for complete axiomatizations of the late and early congruence [PS95]. We also use mismatch in our axiomatization, but we add it mainly in anticipation of its use for concurrent constraint modelling. A complete axiomatization of hyperequivalence without mismatch can be done along the lines of [San96]. Nevertheless, the axiomatization using mismatch displays the difference between scope and restriction in a clear way.

Here and in the following we use M, N to stand for a match or a mismatch operator, and define $\mathrm{n}([x = y]) = \mathrm{n}([x \neq y]) = \{x, y\}$. We add the scope law $(x)MP \equiv M(x)P$, *if* $x \notin \mathrm{n}(M)$ to the structural congruence. We write "match sequence" for a sequence of match *and* mismatch operators, ranged over by $\tilde{M}$, $\tilde{N}$, and we say that $\tilde{M}$ *implies* $\tilde{N}$, written $\tilde{M} \Rightarrow \tilde{N}$, if the conjunction of all matches and mismatches in $\tilde{M}$ logically implies all elements in $\tilde{N}$, and that $\tilde{M} \Leftrightarrow \tilde{N}$ if $\tilde{M}$ and $\tilde{N}$ imply each other. We write $(\tilde{x})$ for a sequence of scope operators $(x_1) \cdots (x_n)$, for $n \geq 0$, and $\sum_{i \in I} P_i$ for finite general summation, $P_1 + \cdots + P_n$.

For the axiomatization of hyperequivalence we subsume the fact that the equivalence is a congruence. We also subsume some of the laws for structural congruence (see Table 2) and the convention that $[x/x] \,.\, P = \mathbf{1} \,.\, P$. The axioms of the up-calculus are given in Table 3, and in Table 4 we present some derived rules (whose names start with **D**).

Definition 9 *A substitution σ* agrees with *a match sequence $\tilde{M}$, and $\tilde{M}$ agrees with σ, if for all x, y which appear in $\tilde{M}$ it holds that $\sigma(x) = \sigma(y)$ iff $\tilde{M} \Rightarrow [x = y]$.*

Of the derived rules, **DM2** follows from $[x = y]\alpha \,.\, P \overset{\mathbf{M2}}{=} [x = y](\alpha \,.\, P)\{x/y\} \overset{\mathbf{DM1}}{=} [x = y](\alpha \,.\, [x = y]P)\{x/y\} \overset{\mathbf{M2}}{=} [x = y]\alpha \,.\, [x = y]P$; **DM3** is derivable because we can write $\tilde{M}$ with the matches closest to P, and then apply **M2** repeatedly. The only worry is if the accumulated substitution becomes ambiguous, e.g. if $\{x/y\}\{z/y\}$ occurs. Since σ agrees with $\tilde{M}$, this means that $\tilde{M} \Rightarrow [x = y]$

Summation		
S1	$P + \mathbf{0} = P$	
S2	$P + Q = Q + P$	
S3	$P + (Q + R) = (P + Q) + R$	
Scope		
R0	$(x)\mathbf{0} = \mathbf{0}$	
R1	$(x)(y)P = (y)(x)P$	
R2	$(x)(P + Q) = (x)P + (x)Q$	
Match and Scope		
RM1	$(x)[y = z]P = [y = z](x)P$	if $x \neq y, x \neq z$

Table 2: Axioms from structural congruence.

Summation		
S4	$P + P = P$	
Match		
M1	$\tilde{M}P = \tilde{N}P$	if $\tilde{M} \Leftrightarrow \tilde{N}$
M2	$[x = y]P = [x = y](P\{x/y\})$	
M3	$MP + MQ = M(P + Q)$	
M4	$[x \neq x]P = \mathbf{0}$	
M5	$P = [x = y]P + [x \neq y]P$	
Scope		
R3	$(x)\alpha \,.\, P = \alpha \,.\, (x)P$	if $x \notin \mathrm{n}(\alpha)$
R4	$(x)\alpha \,.\, P = \mathbf{0}$	if $x = \mathrm{subj}(\alpha)$, α not upd.
Match and Scope		
RM2	$(x)[x = y]P = \mathbf{0}$	if $x \neq y$
Update		
U1	$[x/y] \,.\, P = [x/y] \,.\, [x = y]P$	
U2	$(z)[x/z] \,.\, P = \mathbf{1} \,.\, P$	if $z \neq x, z \notin \mathrm{fn}(P)$
Expansion		
E	for $P \equiv \Sigma_i M_i(\tilde{x}_i)\alpha_i.P_i$, $Q \equiv \Sigma_j N_j(\tilde{y}_j)\beta_j.Q_j$, $$P \mid Q \;=\; \sum_i M_i(\tilde{x}_i)\alpha_i \,.\, (P_i \mid Q) + \sum_j N_j(\tilde{y}_j)\beta_j \,.\, (P \mid Q_j) + \sum_{\alpha_i \mathrm{opp} \beta_j} M_i N_j(\tilde{x}_i, \tilde{y}_j)[a_i = b_j][z_i/w_j] \,.\, (P_i \mid Q_j)$$ where $\alpha_i \mathrm{opp} \beta_j$ means $\alpha_i \equiv \overline{a_i} z_i$ and $\beta_j \equiv b_j w_j$ or vice versa.	

Table 3: Axioms.

Match		
DM1	$[x = x]P = P$	
DM2	$[x = y]\alpha\,.\,P = [x = y]\alpha\,.\,[x = y]P$	
DM3	$\tilde{M}P = \tilde{M}(P\sigma)$	for σ agreeing with $\tilde{M}$
DM4	$M\mathbf{0} = \mathbf{0}$	
DM5	$MP + P = P$	
Match and Scope		
DRM1	$(x)[y \neq z]P = [y \neq z](x)P$	if $x \neq y, x \neq z$
DRM2	$(x)[x \neq y]P = (x)P$	if $x \neq y$
Update		
DU1	$[x/y]\,.\,P = [x/y]\,.\,(P\{x/y\})$	
DU2	$(z)[x/z]\,.\,P = \mathbf{1}\,.\,(P\{x/z\})$	if $x \neq z$

Table 4: Derived rules.

and $\tilde{M} \Rightarrow [z = y]$, but then $\tilde{M} \Rightarrow [x = z]$ as well, and a corresponding substitution must occur in σ.

The derived rules **DRM** for match and scope correspond to the mismatch cases of the axioms **RM**, which apply to matches. **DRM1** is derived along the lines of [PS95], using **R2,M3,M4,M5,RM1** and the axioms for sum and congruence. **DRM2** follows from $(x)P \overset{\mathbf{M5,R1}}{=} (x)[x = y]P + (x)[x \neq y]P \overset{\mathbf{RM2,S1}}{=} (x)[x \neq y]P$. Finally, **DU1** follows from **U1** and **M2**, while **DU2** follows from **N2** and **DU1**.

It is interesting to note that all axioms except the two for update are well known from previous axiomatizations of the π-calculus. For example, all the axioms from Sangiorgi's axiomatization of open bisimulation equivalence for the π-calculus [San96] hold for hyperequivalence in the up-calculus, except the axiom treating restriction and the weaker form of congruence wrt input prefix. Please note also that our expansion law **E** is syntactically simpler than the corresponding laws for π-calculus equivalences.

Distinction relations between names have been used in axiomatizating π-calculus equivalences in [MPW92, San96], but are not needed in our axiomatization of up-calculus. In both calculi **RM2**, **DRM2** and **DM2** are valid, saying that a new name (by Scope or Restriction) is not equal to another name in the immediately following action (**RM2,DRM2**), and that if two names are equal they stay equal (**DM2**). However, in the π-calculus the law $[x \neq y]\alpha\,.\,P = [x \neq y]\alpha\,.\,[x \neq y]P$ holds for early and late congruence [PS95], while it is not valid for hyperequivalence.

In the rest of this section we outline the completeness proof for our axiom system, and briefly present the axiom system for up-calculus without mismatch. We begin with some basic definitions and one Lemma from [PS95].

Definition 10 *The* depth *of an agent P, $\mathrm{d}(P)$, is defined inductively as follows:*

$\mathrm{d}(\mathbf{0}) = 0$, $\mathrm{d}(\alpha\,.\,P) = 1 + \mathrm{d}(P)$, $\mathrm{d}((\tilde{x})P) = \mathrm{d}(MP) = \mathrm{d}(P)$, $\mathrm{d}(P \mid Q) = \mathrm{d}(P) + \mathrm{d}(Q)$, $\mathrm{d}(P + Q) = \max(\mathrm{d}(P), \mathrm{d}(Q))$.

Definition 11 *A match sequence $\tilde{M}$ is* complete on *a set of names V if for some equivalence relation $\mathcal{R}$ on V, called the equivalence relation* corresponding to *$\tilde{M}$, it holds that $\tilde{M} \Rightarrow [x = y]$ iff $x \mathrel{\mathcal{R}} y$, and $\tilde{M} \Rightarrow [x \neq y]$ iff $\neg(x \mathrel{\mathcal{R}} y)$*

A complete match sequence $\tilde{M}$ is maximally consistent *in the sense that adding something not already implied by $\tilde{M}$ makes $\tilde{M}$ inconsistent.*

Lemma 5 ([PS95]) *Let V be a set of names and let $\tilde{M}$ be complete on V.*

1. *If $\tilde{N}$ is another match sequence with names in V, then either $\tilde{M}\tilde{N}$ is unsatisfiable or $\tilde{M}\tilde{N} \Leftrightarrow \tilde{M}$.*

2. *If $\tilde{N}$ is another match sequence complete on V such that $\tilde{M}$ and $\tilde{N}$ both agree with the same substitution σ, then $\tilde{M} \Leftrightarrow \tilde{N}$.*

The completeness of our axiom system says that if two agents are hyper-equivalent, we can prove this using our axioms. We show that this is true in a standard way: first we define a head normal form for agents, and show that any agent can be written on this form. We can then find exactly which part of one agent that was used to simulate the transition of the other.

Definition 12 *An agent P is in* head normal form (HNF) on *V (a finite set of names) if P is on the form*

$$\sum_{i \in I} \tilde{M}_i(\tilde{x}_i)\alpha_i\,.\,P_i$$

where for all i, $\tilde{x}_i \cap V = \emptyset$, $\tilde{x}_i = \emptyset$ or $\tilde{x}_i = \{\mathrm{obj}(\alpha_i)\}$ and $\tilde{M}_i$ is complete on V.

Lemma 6 *For all agents P and finite V such that $\mathrm{fn}(P) \subseteq V$, there is an agent H such that $\mathrm{d}(H) \leq \mathrm{d}(P)$, H is in HNF on V, and $\vdash P = H$ from the axioms of Tables 2 and 3.*

Proof: By structural induction on P. $\square$

Theorem 7 *If $P \sim Q$, then $\vdash P = Q$ from the axioms of Tables 2 and 3.*
PROOF: By Lemma 6 we can assume P and Q to be in HNF on $\mathrm{fn}(P,Q)$. The proof is by induction on the depth of $P+Q$, showing that each summand of P is provably equal to a summand of Q.

If the depth is 0, $P \equiv Q \equiv \mathbf{0}$ and we are done. Otherwise, let $\tilde{M}(\tilde{x})\alpha\,.\,P'$ be a summand of P, σ a substitution agreeing with $\tilde{M}$, and $\sigma(w) = w$ for all $w \notin \mathrm{fn}(P,Q)$. Then $P\sigma \stackrel{((\tilde{x})\alpha)\sigma}{\longrightarrow} P'\sigma$ (here $\tilde{x}$ contains one or zero names). By alpha-conversion $\tilde{x} \cap \mathrm{fn}(Q) = \emptyset$, and by $P \sim Q$, $Q\sigma \stackrel{((\tilde{x})\alpha)\sigma}{\longrightarrow} Q''$, and

$$\begin{array}{rcll} P'\sigma & = & Q'' & \text{by induction, if } \alpha \text{ not an update action} \qquad (1) \\ P'\sigma\sigma_\alpha & = & Q''\sigma_\alpha & \text{by induction, if } \alpha \text{ an update action} \qquad (2) \end{array}$$

where σ_α is $\{x/y\}$ if $\alpha\sigma$ is $[x/y]$.

By alpha-conversion, we can write the summand of Q used to simulate the action of $P\sigma$ as $\tilde{N}(\tilde{x})\beta\,.\,Q'$. Then $\tilde{N}$ agrees with σ and $(\tilde{N}(\tilde{x})\beta\,.\,Q')\sigma \stackrel{((\tilde{x})\beta)\sigma}{\longrightarrow} Q'\sigma$, so $Q'' \equiv Q'\sigma$ and $((\tilde{x})\alpha)\sigma = ((\tilde{x})\beta)\sigma$, which implies

$$\alpha\sigma = \beta\sigma \qquad (3)$$

since σ is identity for names not free in P, Q.

$\tilde{M}$ and $\tilde{N}$ are complete, so by Lemma 5(2) we know $\tilde{M} \Leftrightarrow \tilde{N}$. Since σ agrees with $\tilde{M}$

$$\tilde{M}(\tilde{x})\alpha\,.\,P' \stackrel{\mathbf{DM3}}{=} \tilde{M}((\tilde{x})\alpha\,.\,P')\sigma$$

In the case that α is not an update action,

$$\begin{array}{rl} \tilde{M}((\tilde{x})\alpha\,.\,P')\sigma & = \tilde{M}(\tilde{x})(\alpha\sigma)\,.\,(P'\sigma) \stackrel{(3)}{=} \tilde{M}(\tilde{x})(\beta\sigma)\,.\,(P'\sigma) \\ \stackrel{(1)}{=} & \tilde{M}(\tilde{x})(\beta\sigma)\,.\,(Q'\sigma) \stackrel{\mathbf{M1}}{=} \tilde{N}(\tilde{x})(\beta\sigma)\,.\,(Q'\sigma) = \tilde{N}((\tilde{x})\beta\,.\,Q')\sigma \\ \stackrel{\mathbf{DM3}}{=} & \tilde{N}(\tilde{x})\beta\,.\,Q' \end{array}$$

In the case that α is an update action,

$$\begin{array}{rl} \tilde{M}((\tilde{x})\alpha\,.\,P')\sigma \stackrel{\mathbf{DU1}}{=} & \tilde{M}(\tilde{x})(\alpha\sigma)\,.\,(P'\sigma\sigma_\alpha) \stackrel{(2)}{=} \tilde{M}(\tilde{x})(\alpha\sigma)\,.\,(Q'\sigma\sigma_\alpha) \\ \stackrel{\mathbf{DU1}}{=} & \tilde{M}(\tilde{x})(\alpha\sigma)\,.\,(Q'\sigma) \stackrel{(3)}{=} \tilde{M}(\tilde{x})(\beta\sigma)\,.\,(Q'\sigma) \stackrel{\mathbf{M1}}{=} \tilde{N}(\tilde{x})(\beta\sigma)\,.\,(Q'\sigma) \\ = & \tilde{N}((\tilde{x})\beta\,.\,Q')\sigma \stackrel{\mathbf{DM3}}{=} \tilde{N}(\tilde{x})\beta\,.\,Q' \end{array}$$

Thus we have proven that each summand of P is provably equal to a summand of Q, and then it suffices to use the **S** rules to infer that P and Q are provably equal. $\square$

By dropping axioms **M4** and **M5**, and promoting **DM5** to an axiom, we get an axiomatization of hyperequivalence for the up-calculus without mismatch, which can be proved complete along the lines of [San96].

5 Conclusion

We have presented a new calculus, the *update calculus*, which is a step towards a unifying framework for concurrent actions with local and global effect. The calculus is constructed by on the one hand adding to the π-calculus *update* actions written $[y/z]$, meaning "z should everywhere be replaced by y" with a potentially unbounded extent, and on the other hand simplifying the π-calculus by replacing the two binding operators by one *scope* operator (z) explicitly controlling the extent of updates of z. In the resulting calculus input and output are symmetric, and the π-calculus can be recovered as a subcalculus by simple syntactic restrictions.

By providing a framework for global updates without a separate global state, we hope that the update calculus is well suited to model the global updates of imperative languages, as well as assignment to shared structures in concurrent constraint languages and the local binding in functional concurrent languages and formalisms.

In the present paper we have defined a bisimulation congruence, called *hyperequivalence*. It is strictly stronger than bisimulation equivalences previously studied for the π-calculus, and we show that it is the *only* reasonable congruence for the up-calculus. We have also given equational laws constituting a complete axiomatization of hyperequivalence. The axiomatization highlights the semantic difference between the scope operator of the up-calculus and the restriction operator of the π-calculus.

The idea to separate scope from input has occurred independently to Fu in encodings of the λ-calculus and of cut elimination in linear logic [Fu97]. Fu's semantics is significantly more complex since it lacks update actions.

This paper is a starting point for much further work. To give confidence that the up-calculus is computationally feasible, we are working on a translation into the polyadic π-calculus. To use the new calculus for modelling concurrent constraint programming we are constructing type systems for differentiating between different uses of names; these systems can also be used to relate the up-calculus to other calculi such as the γ-calculus [Smo94]. A polyadic version of the up-calculus will also be constructed; here we anticipate the interesting task of defining the action, if any, from $a\langle xx\rangle \mid \overline{a}\langle bc\rangle$. On the more practical side, we want to construct a tool for reasoning about up-calculus. We aim at modifying the Mobility Workbench [VM94], an automated tool for the π-calculus, for this purpose.

References

[Bar84] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1984.

[BKS88] R. J. R. Back and R. Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513–554, Oct. 1988.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[Dam93] M. Dam. Model checking mobile processes. In E. Best, editor, *CONCUR'93: Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 22–36. Springer-Verlag, 1993.

[Fu97] Y. Fu. A proof theoretical approach to communication. In P. Degano and R. Gorrieri, editors, *ICALP'97: Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer-Verlag, 1997.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, Oct. 1991.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[PS95] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation*, 120(2):174–197, 1995.

[PS96] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In R. Alur and T. A. Henzinger, editors, *Proceedings of CAV'96*, volume 1102 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[PV97] J. Parrow and B. Victor. The Update Calculus (full version). Available electronically, 1997. URL: http://www.docs.uu.se/~victor/tr/upd-full.ps.gz.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, Department of Computer Science, University of Edinburgh, UK, 1992.

[San95a] D. Sangiorgi. On the proof method for bisimulation (extended abstract). In J. Wiederman and P. Hájek, editors, *Mathematical Foundations of Computer Science 1995 (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 479–488. Springer-Verlag, 1995.

[San95b] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995.

[San96] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

[Smo94] G. Smolka. A foundation for higher-order concurrent constraint programming. In J.-P. Jouannaud, editor, *Proc. 1st International Conference on Constraints in Computational Logics*, volume 845 of *Lecture Notes in Computer Science*, pages 50–72. Springer-Verlag, Sept. 1994.

[VM94] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *CAV'94: Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.

[VP96] B. Victor and J. Parrow. Constraints as processes. In U. Montanari and V. Sassone, editors, *CONCUR'96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 389–405. Springer-Verlag, 1996.