A PRIORI VERIFICATION OF REACTIVE SYSTEMS

On Simultaneous Syntactic Action Refinement for TCSP and the Modal Mu-Calculus

Mila Majster-Cederbaum, Frank Salger and Maria Sorea
Universität Mannheim
Fakultät für Mathematik und Informatik,
D7, 27, 68131 Mannheim, Germany
{mcb, fsalger,msorea }@pi2.informatik.uni-mannheim.de

Abstract In this paper we present a refinement operator $\cdot[a \rightsquigarrow Q]$, defined both on $TCSP$-like process terms P and formulas φ of the *Modal Mu-Calculus*. We show that

> the system induced by a term P satisfies a specification φ if and only if the system induced by the refined term $P[\alpha \rightsquigarrow Q]$ satisfies the refined specification $\varphi[\alpha \rightsquigarrow Q]$

where Q is a process term from an appropriate sublanguage of $TCSP$. We explain how this result can be used to reason about reactive systems. In particular it supplies a method to verify systems a priori: Provided $P \models \varphi$ holds, the refinement of φ into $\varphi[a \rightsquigarrow Q]$ induces a transformation of P into $P[a \rightsquigarrow Q]$ such that $P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q]$. The above result holds under the restriction that the set of actions that occur in the term Q has to be disjoint from the set of actions occurring in the term P and the formula φ. Though such restrictions on alphabet disjointness are commonly adopted in approaches to syntactic action refinement for process term languages they preclude the possibility to carry out certain refinement steps that might be necessary in the stepwise development of reactive systems. We show, that while dropping the above restriction, the validity of the two implications comprised in the above result can still be established independently for different interesting fragments of the Modal Mu-Calculus.

Keywords: Verification, Syntactic Action Refinement, Process Algebra, Reactive Systems, Temporal Logic.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35533-7_26

Tommaso Bolognesi and Diego Latella (Eds.), *Formal Methods for Distributed System Development.*

1. INTRODUCTION

One possibility to model *reactive systems* is provided through various *process calculi* like e.g. CCS [16] or TCSP [2, 17]. On the other hand *temporal logics* [9] can be used to reason about attributes of such systems. A particular expressive temporal logic is the *Modal Mu-Calculus* [14] as it subsumes many logics that are commonly used to specify properties of reactive systems [10, 7]. The basic building blocks of process calculi and modal logics are *(atomic) actions* considered to be the conceptual entities at a chosen level of abstraction. The concept of action refinement (see e.g. [1, 11, 12]) is sometimes regarded as one of the fundamental methods to develop reactive systems. A particularly interesting concept of action refinement is *syntactic action refinement* (e.g. [1]) where an action a which occurs in a process term P is refined by a more complex term Q thereby yielding a more detailed process description $P[a \rightsquigarrow Q]$. Due to its definitional clarity, syntactic action refinement can easily be used by system developers without a thorough knowledge about reactive system semantics.

Until recently, the connection between syntactic action refinement in process languages and syntactic action refinement in logics has rarely been adressed in the literature (see the last section for related work). However, syntactic action refinement in logics has various interesting applications to the verification of reactive systems as already discussed in [15]. We show the validity of the assertion

$$P \models \varphi \Leftrightarrow P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q] \quad (*)$$

where $\cdot[a \rightsquigarrow Q]$ denotes the refinement operator in the considered process languages and logics. P is a process term from a $TCSP$-like language, Q is a term from an appropriate sublanguage and φ is a Modal Mu-Calculus-formula that in addition might contain refinement operators.

The simultaneous application of a refinement operator to process terms and formulas (as in assertion $(*)$) will be called *simultaneous syntactic action refinement* (SSAR for short). Assertion $(*)$ says, that the system induced by a term P[1] satisfies a specification φ if and only if the system induced by the refined term $P[\alpha \rightsquigarrow Q]$ satisfies the refined specification $\varphi[\alpha \rightsquigarrow Q]$. As an application of assertion $(*)$ we consider the following 'assembly line' process in a car factory shown in Figure 1 which can be 'implemented' by a $TCSP$-like process expression P. The job of P is to adjust the motor and the gear of a car and to mount the

[1] In assertion $(*)$, we identified the program (process term) P with its semantics. This convention will be continued throughout the paper if the context avoids ambiguity.

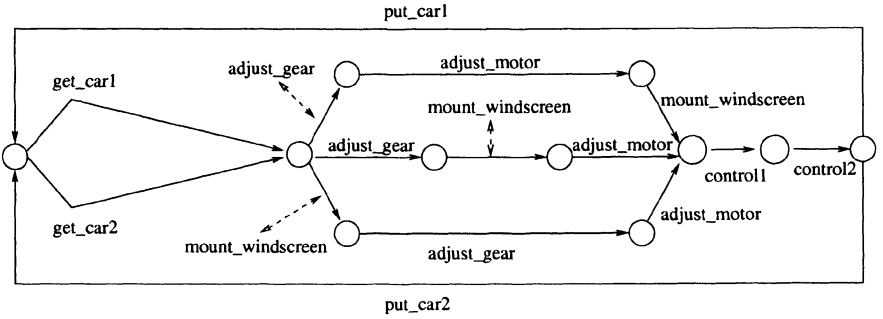

Figure 1 The Process P

windscreen. To reach a defined system status before the car is carried back to the conveyer band two control actions are executed by P. The process P has the temporal property that 'whenever a car is taken from the conveyer band (either *get_car*1 or *get_car*2 is executed), the control actions will eventually be executed' which can be denoted by a formula φ in the Modal Mu-Calculus. We propose that the verification task be facilitated by integrating verification into the procedure of program development via assertion $(*)$. To this end we observe, that the process P arises from the process P_s (shown in Figure 2) by the application of four successive refinement steps i.e.

$$\begin{aligned} P_1 &= P_s[control \rightsquigarrow (control1; control2)] \\ P_2 &= P_1[put\_car \rightsquigarrow (put\_car1 + put\_car2)] \\ P_3 &= P_2[adjust \rightsquigarrow (adjust\_gear; adjust\_motor)] \\ P &= P_3[get\_car \rightsquigarrow (get\_car1 + get\_car2)] \end{aligned}$$

Now let us assume that we had already established $P_s \models \varphi_s$ where the Modal Mu-Calculus formula φ_s denotes the property 'whenever a car is taken from the conveyer band, the control action will eventually be executed'.

In this paper we show how to define a *logical refinement* operation $\cdot[a \rightsquigarrow Q]$ on the Modal Mu-Calculus where

$$\begin{aligned} \varphi_1 &= \varphi_s[control \rightsquigarrow (control1; control2)] \\ \varphi_2 &= \varphi_1[put\_car \rightsquigarrow (put\_car1 + put\_car2)] \\ \varphi_3 &= \varphi_2[adjust \rightsquigarrow (adjust\_gear; adjust\_motor)] \\ \varphi &= \varphi_3[get\_car \rightsquigarrow (get\_car1 + get\_car2)] \end{aligned}$$

such that $P_s \models \varphi_s$ iff $P \models \varphi$.

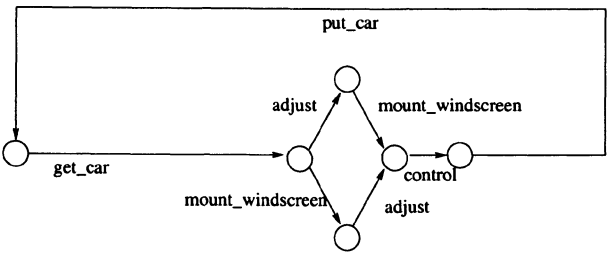

Figure 2 The Process P_s

As indicated by the above example, assertion $(*)$ can be used for the following:

- It offers the possibility of simplifying the verification task by repeatedly applying $(*)$ as was shown in the example. Instead of checking that

$$P = P_s[a_1 \rightsquigarrow Q_1] \dots [a_n \rightsquigarrow Q_n] \models \varphi_s[a_1 \rightsquigarrow Q_1] \dots [a_n \rightsquigarrow Q_n] = \varphi$$

  we only have to check that $P_s \models \varphi_s$.

- Let us assume that the specification of a system is developed incrementally and that φ_s is the actual specification which is now given more details by refining it to $\varphi = \varphi_s[a_1 \rightsquigarrow Q_1] \dots [a_n \rightsquigarrow Q_n]$. When we look for an implementation of φ, we only have to supply an implementation P_s for φ_s thereby automatically obtaining a process $P = P_s[a_1 \rightsquigarrow Q_1] \dots [a_n \rightsquigarrow Q_n]$ such that $P \models \varphi$. This can be seen as a concept of *a priori-verification*. For the example above, suppose the car factory was equipped with the assembly line P_s and $P_s \models \varphi_s$ has already been established. Due to new requirements on the productivity (speed) of P_s the conveyor band *get_car* needs to be replaced by two conveyor bands get_car1, get_car2. This change of resources can be specified by refining φ_s to the specification $\phi = \varphi_s[get\_car \rightsquigarrow (get\_car1 + get\_car2)]$. The system $\tilde{P} = P_s[get\_car \rightsquigarrow (get\_car1 + get\_car2)]$ which is given via assertion (1) is (a priori) correct with respect to ϕ, i.e. $\tilde{P} \models \phi$. The other refinement steps in the example can easily been interpreted accordingly.

- As a determination of the logical consequences of process term refinements: If $P \models \varphi$ then the process refinement $P[a \rightsquigarrow Q]$ automatically yields a refined specification $\varphi[a \rightsquigarrow Q]$ via assertion $(*)$ such that $P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q]$.

Some restrictions have to be obeyed to guarantee that assertion (∗) holds, one of which is *alphabet disjointness*[2] of P from Q. Alphabet disjointness of P and Q for terms $P[a \rightsquigarrow Q]$ is commonly used in approaches to syntactic action refinement[3]. However alphabet disjointness rules out the possibility to conduct particular refinement steps that can become important in the development of reactive systems:
Suppose the system P can execute the atomic actions a, b. At the current level of abstraction, the action a (b) is considered to be the name of a procedure Q_a (Q_b resp.) which is not yet implemented. In an intermediate development step, Q_a and Q_b are implemented making use of a common subsystem S which we might assume has been provided by a system library. Hence, alphabet disjointness of Q_a and Q_b does not hold. Consequently we cannot use assertion (∗) to establish

$$P \models \varphi \Leftrightarrow P[\rho \rightsquigarrow Q_\rho][\varrho \rightsquigarrow Q_\varrho] \models \varphi[\rho \rightsquigarrow Q_\rho][\varrho \rightsquigarrow Q_\varrho]$$

for $\rho, \varrho \in \{a, b\}$, $\rho \neq \varrho$ since alphabet disjointness with respect to the terms $P[\rho \rightsquigarrow Q_\rho]$ and Q_ϱ does not hold.

However it is possible to extract an interesting fragment of the Modal Mu-Calculus for which (under some particular restrictions) it is possible to guarantee the validity of the assertion

$$P \models \varphi \Rightarrow P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q].$$

Furtheron, a fragment for which the assertion

$$P \models \varphi \Leftarrow P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q]$$

holds will be presented.

In Section 2 we present a $TCSP$-like process calculus which contains an operator for syntactic action refinement. The concept of logical refinement for the Modal Mu-Calculus is defined in Section 3. Section 4 provides the link between those two refinement concepts and contains the main results as well as their applications to system verification. A summary, a short discussion on related papers and work in progress are given in Section 5. Section 6 contains some standard definitions.

[2]The alphabet of a process term P is the set of actions which occur in P.
[3]One argues that this restriction avoids the undesired mixture of different levels of abstraction [11]. Moreover as atomic actions are merely 'action names' it is possible to rename them without losing any generality.

2. SYNTACTIC ACTION REFINEMENT IN THE SYSTEM MODEL

In this section we fix the framework used to model reactive systems. Let $Act := \{a, b, \ldots\}$ be a set of actions and $Var_{Act} := \{v_1, v_2, \ldots\}$ be a set of distinguished action variables[4] such that $Act \cap Var_{Act} = \emptyset$. We let $\alpha, \beta, \gamma, \ldots$ range over the set $\mathcal{A} := Act \cup Var_{Act}$, the elements of which are called *atomic performances* or simply *performances*. Furtheron we let $Idf := \{x, y, \ldots\}$ be a fixed set of *identifiers*. As usual the process expression 0 is used to denote a process that is unable to perform any atomic performance.

Let $R\Delta$ be the language of process terms generated by the grammar

$$Q ::= \alpha \mid (Q + Q) \mid (Q; Q) \mid Q[\alpha \rightsquigarrow Q].$$

Let $R\Sigma$ be the language of process terms generated by the grammar

$P ::= 0 \mid \alpha \mid x \mid (P + P) \mid (P; P) \mid (P \|_A P) \mid fix(x = P) \mid P[\alpha \rightsquigarrow Q]$

where $Q \in R\Delta$ and $A \subseteq \mathcal{A}$. Let Σ, Δ be the languages of process expressions generated by the grammars for $R\Sigma$, $R\Delta$ respectively, without the rule $P ::= P[\alpha \rightsquigarrow Q]$.

An identifier x is *guarded* in a term $P \in R\Sigma$ iff each free occurrence of x only occurs in subexpressions F where F lies in a subexpression $(E; F)$ such that $E \notin \surd$, i.e. E can execute a performance (see Appendix A1). A term $P \in R\Sigma$ is called *guarded* iff in each subexpression $fix(x = Q)$ of P the identifier x is guarded in Q. We let $GR\Sigma$ ($G\Sigma$) be the set of all guarded $R\Sigma$ (Σ)-expressions.

The function $\xi : R\Sigma \to 2^{\mathcal{A}}$ that gives the set of performances occurring in a process expression is defined by induction on the structure of $R\Sigma$, e.g. $\xi(\alpha) := \{\alpha\}$ and $\xi((P_1 \|_A P_2)) := \xi(P_1) \cup \xi(P_2)$. The set of synchronisation performances of a process expression $P \in R\Sigma$ is given by the function $\chi : R\Sigma \to 2^{\mathcal{A}}$, e.g. $\chi(\alpha) := \emptyset$ and $\chi((P_1 \|_A P_2)) := \chi(P_1) \cup \chi(P_2) \cup A$.

A term $P_1 \in R\Sigma$ is called *ξ-disjoint* from a term $P_2 \in R\Sigma$ iff $\xi(P_1) \cap \xi(P_2) = \emptyset$. A term $P_1 \in R\Sigma$ is called *$\chi\xi$-disjoint*[5] from a term $P_2 \in R\Sigma$ iff $\chi(P_1) \cap \xi(P_2) = \emptyset$.

A term $P_1 \in R\Sigma$ is called *alphabet-disjoint* from a term $P_2 \in R\Sigma$ iff it is ξ-disjoint and $\chi\xi$-disjoint from P_2. A term $P \in R\Sigma$ is called *uniquely synchronized* iff for all terms $(P_1 \|_A P_2)$ that occur in P, $A = \chi(P_i)$ holds for $i = 1, 2$.

To give a meaning to refined terms $P[\alpha \rightsquigarrow Q]$, we make use of a *reduction function red* : $R\Sigma \to \Sigma$ that removes all occurrences of refinement

[4] Action variables will be used later to define the refinement of formulas (cf. section 3).
[5] Please note, that $\chi\xi$-disjointness is no symmetric relation on the set $R\Sigma$.

operators in a process expression illustrated by Figure 3. The function *red* uses syntactic substitution $\cdot\{Q/\alpha\}$ (see Appendix A2).

Let $P, P_1, P_2 \in R\Sigma$ and $Q \in R\Delta$ be process expressions. The function $red : R\Sigma \to \Sigma$ is defined as follows:

$red(*) := *$ for $* \in \{0\} \cup Idf \cup \mathcal{A}$ $\quad red(fix(x = P)) := fix(x = red(P))$

$red((P_1\ op\ P_2)) := (red(P_1)\ op\ red(P_2))$ where $op \in \{+, ;, \|_A\}$

$red(P[\alpha \rightsquigarrow Q]) := (red(P))\{red(Q)/\alpha\}$

Figure 3 The Reduction Function

The operational semantics of the language Σ is given as usual (see Appendix A3). The semantics of a process expression P is a *labelled transition system with termination* (*LTST*) i.e. a tuple $\mathcal{T}(P) = (P, \Sigma, \mathcal{A}, \to, \surd)$[6]. Since the terms $P[\alpha \rightsquigarrow Q]$ and $red(P[\alpha \rightsquigarrow Q])$ are supposed to behave identically[7], we define $\mathcal{T}(P) := \mathcal{T}(red(P))$ to supply semantics for terms $P \in R\Sigma$ (cf. [1]). In what follows we sometimes identify the term P with the transition system $\mathcal{T}(P)$ if the context avoids ambiguity. The fact that the parallel operator cannot occur in terms $Q \in R\Delta$ is no severe restriction: due to the absence of the fix operator in all terms $Q \in R\Delta$ any such term induces a *finite state system* whence it is always possible to construct for every term Q in the language[8] $R\Delta^+$ a term $\tilde{Q} \in \Delta$ such that $\mathcal{T}(Q) \equiv_b \mathcal{T}(\tilde{Q})$ ($\equiv_b$ denotes strong bisimulation equivalence). If $Q \in R\Delta^+$ is deadlock free, the meaning of 'replacing' performances α in some term P by Q is the same as for the replacement of every α in P by $\tilde{Q}$. The exclusion of the empty process term 0 means that we disallow 'forgetful refinement'[9]. As the refinement of a (terminating) performance α by some infinite behaviour violates the intuition [12], no expression of the form $fix(x = P)$ is allowed to occur in a term $Q \in R\Delta$. Let $LTST_{fin} := \{\mathcal{T}(P) | P \in R\Sigma$ and the set of reachable states of $\mathcal{T}(P)$ is finite$\}$. The set of reach-

[6] $P \in \Sigma$ is the initial state and $\to \subseteq \Sigma \times \mathcal{A} \times \Sigma$ is the set of transitions, determined via the rules of the operational semantics. $\surd$ is the set of terminated states as defined in the Appendix.

[7] The syntactic difference between the terms $P[\alpha \rightsquigarrow Q]$ and $red(P[\alpha \rightsquigarrow Q])$ is that in the former information about α is scattered over several levels of abstraction whereas only one level of abstraction exists in the latter.

[8] $R\Delta^+$ is the language generated by the grammar for $R\Delta$ augmented with the rule $Q ::= (Q \| _A Q))$.

[9] Such refinements cannot be explained by a change in the level of abstraction [20] and are usually avoided (see e.g. [1]).

able states of $\mathcal{T}(P)$ consists of all states which can be reached from the state $red(P)$.

3. SYNTACTIC ACTION REFINEMENT IN THE SPECIFICATION LOGIC

Let $\mu\mathcal{L}$ be the (negation-free form of the) *Modal Mu-Calculus* [14] generated by the grammar

$$\Phi ::= \top \mid \bot \mid Z \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid [\alpha]\Phi \mid \langle\alpha\rangle\Phi \mid \nu Z.\Phi \mid \mu Z.\Phi$$

where α ranges over $\mathcal{A}$ and Z over a fixed set Var of *variables*. Let $R\mu\mathcal{L}$ be the language generated by the above grammar with the additional rule $\Phi ::= \Phi[\alpha \rightsquigarrow Q]$ where $Q \in R\Delta$. Let $R\mu\mathcal{L}_{\langle\cdot\rangle}$ ($R\mu\mathcal{L}_{[\cdot]}$) be the language generated by the grammar for $R\mu\mathcal{L}$ without the rule $\Phi ::= [\alpha]\Phi$ ($\Phi ::= \langle\alpha\rangle\Phi$ resp.). We let σ range over the set $\{\mu, \nu\}$. A *fixed point formula* has the form $\sigma Z.\varphi$ in which σZ *binds* free occurrences of Z in φ. A variable Z is called *free* iff it is not bound. A $R\mu\mathcal{L}$-formula φ is called *closed* iff every variable Z which occurs in φ is bound. A $R\mu\mathcal{L}$-formula φ is called *guarded* iff every occurrence of a variable Z in φ lies in the scope of a modality $[\alpha]$ or $\langle\alpha\rangle$. For a language $L \subseteq R\mu\mathcal{L}$ we define the closed and guarded fragment of L by $CGL := \{\varphi \in L \mid \varphi \text{ is closed and guarded}\}$. The function $\xi : R\mu\mathcal{L} \to 2^{\mathcal{A}}$ gives the set of performances that occur in a formula. A formula $\varphi \in R\mu\mathcal{L}$ is called ξ-*disjoint* from a term $P \in R\Sigma$ iff $\xi(\varphi) \cap \xi(P) = \emptyset$.

We now give the concept of logical substitution by which we are able to define the reduction of formulas (see Figure 4):

Let $Q, Q_1, Q_2 \in \Delta$ and $\varphi, \varphi_1, \varphi_2 \in \mu\mathcal{L}$ and $n, m \in \mathbb{N}$. The operation of logical substitution, $(\varphi)\{\alpha \rightsquigarrow Q\}$ is defined as follows:

$$(*)\{\alpha \rightsquigarrow Q\} := * \quad \text{if } * \in \{\top, \bot\} \cup Var$$

$$((\varphi_1 \odot \varphi_2))\{\alpha \rightsquigarrow Q\} := ((\varphi_1)\{\alpha \rightsquigarrow Q\} \odot (\varphi_2)\{\alpha \rightsquigarrow Q\}) \quad \text{if } \odot \in \{\wedge, \vee\}$$

$$(\Delta_\beta\varphi)\{\alpha \rightsquigarrow Q\} := \Delta_\beta(\varphi)\{\alpha \rightsquigarrow Q\} \quad \text{if } \alpha \neq \beta$$

$$(\Delta_\alpha\varphi)\{\alpha \rightsquigarrow Q\} :=$$

$$\begin{cases} \Delta_\beta(\varphi)\{\alpha \rightsquigarrow Q\} & \text{if } Q = \beta \\ ((\Delta_{v_n}(\varphi)\{\alpha \rightsquigarrow Q\})\{v_n \rightsquigarrow Q_1\} \wedge (\Delta_{v_m}(\varphi)\{\alpha \rightsquigarrow Q\})\{v_m \rightsquigarrow Q_2\}) & \text{if } Q = (Q_1 + Q_2) \\ (\Delta_{v_n}(\Delta_{v_m}(\varphi)\{\alpha \rightsquigarrow Q\})\{v_m \rightsquigarrow Q_2\})\{v_n \rightsquigarrow Q_1\} & \text{if } Q = (Q_1; Q_2) \end{cases}$$

$$(\sigma Z.\varphi)\{\alpha \rightsquigarrow Q\} := \sigma Z.(\varphi)\{\alpha \rightsquigarrow Q\}$$

where in each clause Δ_γ means throughout either $\langle\gamma\rangle$ or $[\gamma]$ for all γ. For

Let $Q \in R\Delta$ be a process expression and $\varphi, \varphi_1, \varphi_2 \in R\mu\mathcal{L}$ be formulas. We define the logical reduction function $\mathcal{R}ed : R\mu\mathcal{L} \to \mu\mathcal{L}$ as follows:

$$\mathcal{R}ed(*) := * \quad \text{if } * \in \{\top, \bot\} \cup Var \qquad \mathcal{R}ed(\varphi[\alpha \rightsquigarrow Q]) := (\mathcal{R}ed(\varphi))\{\alpha \rightsquigarrow red(Q)\}$$

$$\mathcal{R}ed((\varphi_1 \odot \varphi_2)) := (\mathcal{R}ed(\varphi_1) \odot \mathcal{R}ed(\varphi_2)) \quad \text{if } \odot \in \{\wedge, \vee\}$$

$$\mathcal{R}ed([\beta]\varphi) := [\beta]\mathcal{R}ed(\varphi)\ , \quad \mathcal{R}ed(\langle\beta\rangle\varphi) := \langle\beta\rangle\mathcal{R}ed(\varphi)$$

$$\mathcal{R}ed(\sigma Z.\varphi) := \sigma Z.\mathcal{R}ed(\varphi)$$

Figure 4 The Logical Reduction Function

the introduction of variables v_n, v_m we require $v_n, v_m \notin \xi([\alpha]\varphi) \cup \xi(Q)$ and $n \neq m$ ($v_n, v_m \notin \xi(\langle\alpha\rangle\varphi) \cup \xi(Q)$ respectively).

To supply semantics for refined formulas $\varphi[\alpha \rightsquigarrow Q]$, we extend the standard satisfaction relation (see Sect. 6, Def. A4) which is defined relative to valuation functions[10] $\vartheta : Var \to 2^{R\Sigma}$ by means of the clause $P \models_\vartheta \varphi[\alpha \rightsquigarrow Q]$ iff $P \models_\vartheta \mathcal{R}ed(\varphi[\alpha \rightsquigarrow Q])$. We say P *satisfies* φ (with respect to ϑ) iff $P \models_\vartheta \varphi$. For a closed $R\mu\mathcal{L}$-formula φ we simply write $P \models \varphi$. For a fixed point formula $\sigma Z.\varphi$ we observe that least and greatest fixed points (denoted μ, ν resp.) always exist by the results of [19].

4. SAFETY FOR SYNTACTIC ACTION REFINEMENT AND ABSTRACTION

The first result shows that the concept of simultaneous syntactic action refinement is safe provided a condition of alphabet disjointness is met.

Theorem 1 *Let $P \in GR\Sigma$ be a process term, such that $\mathcal{T}(P) \in LTST_{fin}$ and $\varphi \in CGR\mu\mathcal{L}$ be a formula. Further let $Q \in R\Delta$, such that P is alphabet-disjoint from Q and φ is ξ-disjoint from Q. Then $P \models \varphi \Leftrightarrow P[\alpha \rightsquigarrow Q] \models \varphi[\alpha \rightsquigarrow Q]$.*

Proof Idea:
The proof of Theorem 1 is by induction on the structure of $\varphi \in CG\mu\mathcal{L}$ and a subsidiary induction on the structure of $Q \in R\Delta$. We first observe that $\mathcal{T}(P[\alpha \rightsquigarrow Q])$ is finite. The fact that an approximation over at most

[10]The customary updating notation is used: $\vartheta[\mathcal{E}/Z]$ is the valuation ϑ' which agrees with ϑ on all variables $Z \in Var$ except Z, and $\vartheta'(Z) = \mathcal{E}$.

$|\mathcal{T}_r(P)|_S$ steps [11] suffices to capture the actual fixed point on the system P can thus be used in the induction. In order to show Theorem 1, a serial of lemmata which relate the behaviour of a term P with the behaviour of $P[\alpha \rightsquigarrow Q]$ have to be proved.

Alphabet-disjointness is commonly used in approaches to syntactic action refinement. In order to meet the condition of Theorem 1, renaming of performances can often be applied successfully[12]. However alphabet-disjointness is too strong a condition in a situation as the second one described in the introduction. Alphabet-disjointness is necessary in Theorem 1 as can be seen by the following simple example. Consider the process expression $P = b$ and the formula $\varphi = [c]\langle d\rangle\top$. We have $P \models \varphi$ but $red(P[b \rightsquigarrow c]) \not\models \mathcal{R}ed(\varphi[b \rightsquigarrow c])$. Note that we have $\xi(\varphi) \cap \xi(Q) \neq \emptyset$, i.e. φ is not ξ-disjoint from Q. It is clear that dropping the condition of alphabet disjointness of the process terms P and Q only makes sense in conjunction with abandoning the ξ-disjointness of φ from Q. Dropping the latter restriction however leads to fundamental problems: Without it, repeated SSAR can transform an originally satisfiable formula into an unsatisfiable one[13]. The above examples show that we cannot hope for a result like Theorem 1 for any fragment $L \subseteq R\mu\mathcal{L}$ in which it is allowed to compose formulas $\varphi \in L$ containing both types of modalities, i.e. $\langle\alpha\rangle$ and $[\alpha]$ without accepting any restrictions on alphabet disjointness. This is the reason why we consider the logics $R\mu\mathcal{L}_{\langle\cdot\rangle}$ and $R\mu\mathcal{L}_{[\cdot]}$ where only one modality type can occur in the formulas. The logic $R\mu\mathcal{L}_{[\cdot]}$ can be used to express interesting properties of reactive systems, like e.g. the unless-property 'φ remains true in every computation unless ψ holds' or safety properties such as 'φ never holds again whenever ψ has become true'. Moreover, $R\mu\mathcal{L}_{[\cdot]}$ can be used to express liveness-properties under fairness and cyclic-properties (see [18]).

$R\mu\mathcal{L}_{\langle\cdot\rangle}$-formulas can be used to formalize properties like e.g. 'there exists a computation sequence of P in which φ holds infinitely often' or 'there exists a computation sequence of P along which φ is always attainable.' With techniques similar to those applied for Theorem 1 we obtain the following.

Theorem 2 *Let $P \in GR\Sigma$ be a uniquely synchronized process term, such that $\mathcal{T}(P) \in LTST_{fin}$ and $\varphi \in CGR\mu\mathcal{L}_{\langle\cdot\rangle}$ be a formula. Further let $Q \in R\Delta$, such that P is $\chi\xi$-disjoint from Q. If $(\alpha \notin \chi(P) \vee \xi(\varphi) \subseteq \chi(P))$ then $P \models \varphi \Rightarrow P[\alpha \rightsquigarrow Q] \models \varphi[\alpha \rightsquigarrow Q]$.*

[11] $|\mathcal{T}(P)|_S$ denotes the number of states of $\mathcal{T}(P)$.
[12] Cf. the assembly line example of the introduction where no renaming was necessary.
[13] Let $\varphi = (\langle\alpha_1\rangle\top \wedge [\alpha_2]\bot)$. Clearly, φ is satisfiable. On the other hand, the formula $\varphi[\alpha_1 \rightsquigarrow \alpha][\alpha_2 \rightsquigarrow \alpha]$ is not satisfiable.

Theorem 3 *Let $P \in GR\Sigma$ be a uniquely synchronized process term, such that $\mathcal{T}(P) \in LTST_{fin}$ and $\varphi \in CGR\mu\mathcal{L}_{[\cdot]}$ be a formula. Further let $Q \in R\Delta$, such that P is $\chi\xi$-disjoint from Q. If $(\alpha \notin \chi(P) \vee \xi(\varphi) \subseteq \chi(P))$ then $P \models \varphi \Leftarrow P[\alpha \rightsquigarrow Q] \models \varphi[\alpha \rightsquigarrow Q]$.*

As an example, we consider a ('lock-step') solution of a two process mutual exclusion problem.

Example: The process terms P_α and P_β are given by □

$$fix(x = (((\alpha_1; (\alpha; \alpha_2)) + (\beta_1; (\beta; \beta_2))); x))$$

and

$$fix(y = (((\beta_1; (\beta; \beta_2)) + (\alpha_1; (\alpha; \alpha_2))); y))$$

respectively. It is easy to see that the process $ME_{\alpha\beta} = (P_\alpha \|_{\{\alpha_i, \beta_i, \alpha, \beta\}} P_\beta)$ where $i = 1, 2$ enters the (abstract) *critical sections* α and β in mutual exclusion. For $A \subseteq \mathcal{A}$ we let $\diamond_A\varphi$ abbreviate the formula $\bigvee_{\alpha \in A}\langle\alpha\rangle\varphi$ and $\Box_A\varphi$ abrreviate the formula $\bigwedge_{\alpha \in A}[\alpha]\varphi$. We have that

$$ME_{\alpha\beta} \models \varphi = \nu Y.(\mu Z.(\langle\alpha\rangle\top \vee \diamond_{\xi(ME_{\alpha\beta})}Z) \wedge \diamond_{\xi(ME_{\alpha\beta})}Y),$$

i.e. there exists a computation sequence of $ME_{\alpha\beta}$ along which it is always possible to reach a state where the performance α can be executed. Now let $Q = (\gamma_1 + \gamma_2)$. Then we have that

$$ME_{\alpha\beta}[\alpha \rightsquigarrow Q][\beta \rightsquigarrow Q] \models \phi[\alpha \rightsquigarrow Q][\beta \rightsquigarrow Q]$$

via Theorem 2 which says that $ME_{\alpha\beta}[\alpha \rightsquigarrow Q][\beta \rightsquigarrow Q]$ can execute a computation sequence along which it is always possible to reach a state where all performances of Q can be executed. This holds, though another performance (β in our case) is also refined by Q. □

Theorem 2 and Theorem 3 can both be established without the need to consider restrictions of alphabet disjointness whence we might use the two logics $CGR\mu\mathcal{L}_{\langle\cdot\rangle}$ and $CGR\mu\mathcal{L}_{[\cdot]}$ in the stepwise development of systems as demonstrated in the introduction.

Amongst the applications of Theorem 1 to the verification of reactive systems (as described in the introduction), the concept of a priori-verification might be the most interesting one. There, every property φ of a system P which can be expressed in $R\mu\mathcal{L}$ 'carries over' (in its refined form $\varphi[\alpha \rightsquigarrow Q]$) to the refined system $P[\alpha \rightsquigarrow Q]$, in particular this holds for all safety and liveness properties. Strong safety properties (which involve the box modality $[\alpha]$) might not be carried over (in the above sense) when dropping the restriction of alphabet disjointness. Theorem 2 shows however, that it is still possible to verify systems 'a

priori' with respect to properties expressible in the logic $R\mu\mathcal{L}_{\langle\cdot\rangle}$, without the need to ensure alphabet disjointness of the considered process terms and formulas. In essence, these properties are 'existential' properties e.g. weak safety and liveness properties (according to [18]).

In its contrapositive form, Theorem 3 can be used to 'debug' a (concrete) reactive system by means of debugging an abstract system (where the abstraction is based on syntactic action refinement between those systems): If $P \not\models \varphi \in R\mu\mathcal{L}_{[\cdot]}$ then $P[\alpha \rightsquigarrow Q] \not\models \varphi[\alpha \rightsquigarrow Q]$. In the case we cannot disprove φ for P, no information about satisfaction of $\varphi[\alpha \rightsquigarrow Q]$ by $P[\alpha \rightsquigarrow Q]$ can be inferred. However, Theorem 3 might also be used to support model checking techniques for systems that otherwise would remain unfeasable due to the size of their state spaces: If P is such a system then no information at all about satisfaction with respect to any property φ can be established by means of model checking techniques. In particular we could not show $P \not\models \varphi$. However, if we could establish appropriate abstractions P_s and φ_s, i.e. $P = P_s[\alpha_1 \rightsquigarrow Q_1] \ldots [\alpha_n \rightsquigarrow Q_n]$ and $\varphi = \varphi_s[\alpha_1 \rightsquigarrow Q_1] \ldots [\alpha_n \rightsquigarrow Q_n]$ then P_s might well become manageable by a model checker since the state space of P_s might be exponentially smaller then the state space of P due to the well known state explosion problem[14]. Then we could apply the model checker to prove $P_s \not\models \varphi_s$ and conclude $P \not\models \varphi$ via Theorem 3. Various interesting properties can be expressed in $R\mu\mathcal{L}_{[\cdot]}$ (and therefore be used in the debug-procedure described above), in particular strong safety properties of a system P like e.g. $\nu Z.(\varphi \wedge \Box_{\xi(P)} Z)$ meaning 'φ holds in every state of P'.

5. CONCLUSION AND RELATED WORK

We showed that it is possible to establish the validity of the assertion $P \models \varphi$ iff $P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q]$ $(*)$ where P and Q are terms from appropriate $TCSP$-like languages, φ is a *Modal Mu-Calculus*-formula and $\cdot[a \rightsquigarrow Q]$ denotes a refinement operator both on process terms and formulas. As we explained in the introduction, assertion $(*)$ can be used to simplify the verification task. Furthermore, the logical meaning of syntactic action refinement on process terms becomes evident. Most interestingly however is the method of a priori verification supplied by assertion $(*)$: Provided we have $P \models \varphi$, the specification refinement $\varphi[a \rightsquigarrow Q]$ automatically yields an a priori correct process $P[a \rightsquigarrow Q]$ via assertion $(*)$, i.e. $P[a \rightsquigarrow Q] \models \varphi[a \rightsquigarrow Q]$.

[14] A linear reduction of the number of performances in a term $P \in R\Sigma$ can entail an exponential reduction of the number of reachable states of $\mathcal{T}(P)$. Please note that model checking algorithms are based on the investigation of the involved system state spaces, regardless whether those spaces are represented explicitly or implicitly (e.g. using $BDDs$ [3]).

We demonstrated that assertion (∗) does not hold in general but under the condition of alphabet-disjointness. We showed the validity of the assertions $P \models \varphi \Rightarrow P[\alpha \rightsquigarrow Q] \models \varphi[\alpha \rightsquigarrow Q]$ (†) and $P \models \varphi \Leftarrow P[\alpha \rightsquigarrow Q] \models \varphi[\alpha \rightsquigarrow Q]$ (‡) where $P \in GR\Sigma$ is uniquely synchronized and $\varphi \in CGR\mu\mathcal{L}_{\langle\cdot\rangle}$ in assertion (†), $\varphi \in CGR\mu\mathcal{L}_{[\cdot]}$ in assertion (‡). The important point is that the restriction of alphabet disjointness is not necessary to prove the correctness of the two assertions above. Consequently, assertion (∗), (†) and (‡) show that the expressive power of the specification logic determines the applicability of SSAR to the verification of reactive systems. A future topic of research is to further investigate this 'expressiveness-applicability' tradeoff.

We explained that SSAR can also be conceived as a concept of abstraction simplifying the verification of reactive systems. In this sense our approach is related to the method of *abstract interpretations* [6]. There, one can conclude that properties hold for a (concret) system $\mathcal{C}$ if they hold for a (abstract) system $\mathcal{A}$ where the state spaces of $\mathcal{A}$ and $\mathcal{C}$ are related with each other via appropriate abstractions thereby reducing the costs of *model checking* techniques (see e.g. [4, 8]). There however, the formula under consideration remains 'unchanged' whereas it is changed (through refinement) in our approach. In [13], *semantic action refinement* for *synchronisation structures* is used to induce refinement functions for formulas of the considered specification logic whereas we use *syntactic action refinement* for process terms and formulas. In the above paper a *linear time* logic is used whereas our approach is based on the *branching time* Modal Mu-Calculus.

The concept of syntactic action refinement can easily be used by a system developer and the transition system semantics used is closely related to the well known concept of automata. The Modal Mu-Calculus is often conceived as the generic process-logic (see e.g. [8]). We thus believe that our framework satisfies the requirement that formal methods should be easy to comprehend to be of any use in practice [5].

Work is in progress that extends the above results: We study the consequences of introducing the 'hiding'-operator (see e.g. [17]) to the process algebra $R\Sigma$. Further, we investigate how the reduction of formulas can be determined efficiently. Investigations concerning infinite state systems are a future topic as well as investigations to what extend the presented abstraction technique can be automatized. We are currently performing some case studies to determine the practical applicability of our results.

6. AUXILIARY DEFINITIONS

A1 Terminated States:

To evaluate the semantics of the operator ';' it is common to use a special predicate $\surd$: Let $\surd \subseteq R\Sigma$ be the least set which contains the term 0 and is closed under the rules $(P_1 \in \surd \wedge P_2 \in \surd) \Rightarrow (P_1\ op\ P_2) \in \surd$ where $op \in \{\|_A, +, ;\}$ and $(P \in \surd) \Rightarrow fix(x = P) \in \surd$ and $(P \in \surd) \Rightarrow P[\alpha \rightsquigarrow Q] \in \surd$.

A2 Syntactic Substitution:

Let $P, P_1, P_2 \in \Sigma$ and $Q \in \Delta$ be process expressions. Syntactic substitution, denoted $(P)\{Q/\alpha\}$ is defined as follows:

$$(*)\{Q/\alpha\} := * \text{ where } * \in \{0\} \cup Idf \qquad (\alpha)\{Q/\beta\} := \begin{cases} Q & \text{if } \alpha = \beta \\ \alpha & \text{otherwise} \end{cases}$$

$$((P_1\ op\ P_2))\{Q/\alpha\} := ((P_1)\{Q/\alpha\}\ op\ (P_2)\{Q/\alpha\}) \text{ where } op \in \{+, ;\}$$

$$((P_1 \|_A P_2))\{Q/\alpha\} := \begin{cases} ((P_1)\{Q/\alpha\} \|_{A\setminus\{\alpha\}\cup\xi(Q)} (P_2)\{Q/\alpha\}) & \text{if } \alpha \in A \\ ((P_1)\{Q/\alpha\} \|_A (P_2)\{Q/\alpha\}) & \text{if } \alpha \notin A \end{cases}$$

$$(fix(x = P))\{Q/\alpha\} := fix(x = P\{Q/\alpha\})$$

A3 Operational Semantics:

Let $P, Q \in \Sigma$ be process expressions.

$$\frac{}{\alpha \xrightarrow{\alpha} 0} \qquad \frac{P \xrightarrow{\alpha} P'}{(P+Q) \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{(P+Q) \xrightarrow{\alpha} Q'}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{(P;Q) \xrightarrow{\alpha} Q'} \text{ if } P \in \surd \qquad \frac{P \xrightarrow{\alpha} P'}{(P;Q) \xrightarrow{\alpha} (P';Q)}$$

$$\frac{P \xrightarrow{\alpha} P'}{(P \| _A Q) \xrightarrow{\alpha} (P' \| _A Q)} \text{ if } \alpha \notin A \qquad \frac{Q \xrightarrow{\alpha} Q'}{(P \| _A Q) \xrightarrow{\alpha} (P \| _A Q')} \text{ if } \alpha \notin A$$

$$\frac{P[fix(x=P)/x] \xrightarrow{\alpha} Q}{fix(x=P) \xrightarrow{\alpha} Q} \qquad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\alpha} Q'}{(P \| _A Q) \xrightarrow{\alpha} (P' \| _A Q')} \text{ if } \alpha \in A$$

A4 Extended Satisfaction Relation:

Let $P \in R\Sigma$, $Q \in R\Delta$, $\varphi, \psi \in R\mu\mathcal{L}$ and $Z \in Var$.

$$P \models_\vartheta \top, \qquad P \not\models_\vartheta \bot \ , \qquad P \models_\vartheta Z \text{ iff } P \in \vartheta(Z)$$

$P \models_\vartheta (\varphi \wedge \psi)$ iff $P \models_\vartheta \varphi$ and $P \models_\vartheta \psi$

$P \models_\vartheta (\varphi \vee \psi)$ iff $P \models_\vartheta \varphi$ or $P \models_\vartheta \psi$

$P \models_\vartheta [\alpha]\varphi$ iff $P \in \{E \in R\Sigma | \forall E' \in R\Sigma(E \xrightarrow{\alpha} E' \Rightarrow E' \models_\vartheta \varphi)\}$

$P \models_\vartheta \langle\alpha\rangle\varphi$ iff $P \in \{E \in R\Sigma | \exists E' \in R\Sigma(E \xrightarrow{\alpha} E' \text{ and } E' \models_\vartheta \varphi)\}$

$P \models_\vartheta \mu Z.\varphi$ iff $P \in \bigcap \{\mathcal{E} \subseteq R\Sigma | \{E \in R\Sigma | E \models_{\vartheta[\mathcal{E}/Z]} \varphi\} \subseteq \mathcal{E}\}$

$P \models_\vartheta \nu Z.\varphi$ iff $P \in \bigcup \{\mathcal{E} \subseteq R\Sigma | \mathcal{E} \subseteq \{E \in R\Sigma | E \models_{\vartheta[\mathcal{E}/Z]} \varphi\}\}$

$P \models_\vartheta \varphi[\alpha \rightsquigarrow Q]$ iff $P \models_\vartheta \mathcal{R}ed(\varphi[\alpha \rightsquigarrow Q])$

References

[1] Aceto, L. and Hennessy, M. (1991). Adding action refinement to a finite process algebra. *Lecture Notes in Computer Science*, 510:506–519.

[2] Brookes, S. D., Hoare, C. A. R., and Roscoe, A. W. (1984). A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599.

[3] Bryant, R. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[4] Clarke, E., Grumberg, D., and Long, D. (1994). Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542.

[5] Clarke, E. M. and Wing, J. M. (1996). Formal methods: State of the art and future directions. Technical Report CMU-CS-96-178. ftp://reports.adm.cs.cmu.edu/usr/anon/1996/CMU-CS-96-178.ps.

[6] Cousot, P. and Cousot, R. (1992). Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547.

[7] Dam, M. (1994). CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126(1):77–96.

[8] Dams, D., Gerth, R., and Grumberg, O. (1997). Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291.

[9] Emerson, E. A. (1990). Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam. Elsevier Science Publishers.

[10] Emerson, E. A. and Lei, C. L. (1986). Efficient model checking in fragments of the propositional μ-calculus. In *Symposium on Logic in Computer Science (LICS '86)*, pages 267–278, Washington, D.C., USA. IEEE Computer Society Press.

[11] Goltz, U., Gorrieri, R., and Rensink, A. (1994). On syntactic and semantic action refinement. *Lecture Notes in Computer Science*, 789:385–404.

[12] Gorrieri, R. and Rensink, A. (1999). Action refinement. Technical Report UBLCS-99-9, University of Bologna (Italy). Department of Computer Science.

[13] Huhn, M. (1996). Action refinement and property inheritance in systems of sequential agents. In Montanari, U. and Sassone, V., editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 639–654, Pisa, Italy. Springer-Verlag.

[14] Kozen, D. (1983). Results on the propositional mu -calculus. *Theoretical Computer Science*, 27(3):333–354.

[15] Majster-Cederbaum, M. and Salger, F. (1999). A verification technique based on syntactic action refinement in a TCSP-like process algebra and the Hennessy-Milner-Logic. In *Advances in Computing Science-ASIAN'99*, volume 1742 of *LNCS*, pages 379–380. Springer.

[16] Milner, R. (1980). *A Calculus of Communicating Systems.* Springer, Berlin, 1 edition.

[17] Olderog, R. (1986). TCSP: Theory of communicating sequential processes. In *Advances in Petri Nets 1987, ed. Grzegorz Rozenberg, LNCS 266; Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, LNCS 254-255, 1987, LNCS 188 (1984), LNCS 340 (1988), LNCS 483 (1991).*

[18] Stirling, C. (1996). Modal and temporal logics for processes. *Lecture Notes in Computer Science*, 1043:149–237.

[19] Tarski, A. (1955). A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309.

[20] van Glabbeek, R. and Goltz, U. (1989). Equivalence notions for concurrent systems and refinement of actions. In Kreczmar, A. and Mirkowska, G., editors, *Proceedings of the Conference on Mathematical Foundations of Computer Science*, volume 379 of *LNCS*, pages 237–248, Berlin. Springer.