Basic Research in Computer Science

Characteristic Formulae: From Automata to Logic

Luca Aceto
Anna Ingólfsdóttir

BRICS Report Series **RS-07-2**

ISSN 0909-0878 **January 2007**

**See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
IT-parken, Aabogade 34
DK–8200 Aarhus N
Denmark
Telephone: +45 8942 9300
Telefax: +45 8942 5601
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/07/2/`

Characteristic Formulae: From Automata to Logic

Luca Aceto Anna Ingolfsdottir
Department of Computer Science, Reykjavík University
Kringlan 1, 103 Reykjavík, Iceland
and
BRICS (**B**asic **R**esearch **i**n **C**omputer **S**cience)
Centre of the Danish National Research Foundation
Department of Computer Science, Aalborg University
Fr. Bajersvej 7B, 9220 Aalborg Ø, Denmark
Email: luca@ru.is, annai@ru.is

Abstract

This paper discusses the classic notion of characteristic formulae for processes using variations on Hennessy-Milner logic as the underlying logical specification language. It is shown how to characterize logically (states of) finite labelled transition systems modulo bisimilarity using a single formula in Hennessy-Milner logic with recursion. Moreover, characteristic formulae for timed automata with respect to timed bisimilarity and the faster-than preorder of Moller and Tofts are offered in terms of the logic L_ν of Laroussinie, Larsen and Weise.

1 Motivation

The aim of this paper is to introduce the basic ideas and results on a piece of classic concurrency theory that is, perhaps, not as well known as it deserves to be, namely the notion of *characteristic formulae* for processes. Characteristic formulae are neither a new nor a particularly hot topic in concurrency theory at the time of writing. However, we believe that the notion of characteristic formula sheds such a natural connection between automata-based formalisms as a way of describing the actual behaviour of processes and logics—for instance, modal or temporal logics—as formalisms for writing down specifications of the expected behaviour of processes that it is worth surveying some of the developments on this branch of concurrency theory here. We state at the outset that our aim is *not* to give a comprehensive account of all of the work that has been carried out on characteristic formulae and their use in concurrency theory. Rather, we shall present the basic ideas underlying the notion of characteristic formulae

for two specific models of concurrent computation—namely, (finite) labelled transition systems [24] and timed automata [4]—, and we shall refer the reader to the literature for further developments on, and applications of, this notion. Apart from the research papers we shall refer to in this survey, an introduction to characteristic formulae suitable for classroom use in courses on concurrency theory may be found in the forthcoming book [1].

What are Characteristic Formulae? Various types of automata are fundamental formalisms for the description of the behaviour of computing systems. For instance, a widely used model of computation is that of *labelled transition systems* (LTSs) [24]. LTSs underlie Plotkin's Structural Operational Semantics [36, 37] and, following Milner's pioneering work on CCS [32], are by now the formalism of choice for describing the semantics of various process description languages.

Since automata like LTSs can be used for describing specifications of process behaviours as well as their implementations, an important ingredient in their theory is a notion of behavioural equivalence or preorder between (states of) LTSs. A behavioural equivalence describes formally when (states of) LTSs afford the same 'observations', in some appropriate technical sense. On the other hand, a behavioural preorder is a possible formal embodiment of the idea that (a state in) an LTS affords at least as many 'observations' as another one. Taking the classic point of view that an implementation correctly implements a specification when each of its observations is allowed by the specification, behavioural preorders may therefore be used to establish the correctness of implementations with respect to their specifications, and to support the stepwise refinement of specifications into implementations.

The lack of consensus on what constitutes an appropriate notion of observable behaviour for reactive systems has led to a large number of proposals for behavioural equivalences and preorders for concurrent processes. In his by now classic papers [16, 17, 18], van Glabbeek presented a taxonomy of extant behavioural preorders and equivalences for processes.

The approach to the specification and verification of reactive systems in which automata like LTSs are used to describe both implementations and specifications of reactive systems is often referred to as *implementation verification* or *equivalence checking*.

An alternative approach to the specification and verification of reactive systems is that of *model checking* [8, 10, 39]. In this approach, automata are still the formalism of choice for the description of the actual behaviour of a concurrent system. However, specifications of the expected behaviour of a system are now expressed using a suitable logic, for instance, a modal or temporal logic [14, 38]. Verifying whether a concurrent process conforms to its specification expressed as a formula in the logic amounts to checking whether the automaton describing the behaviour of the process is a model of the formula.

It is natural to wonder what the connection between these two approaches to the specification and verification of concurrent computation is. A classic,

and most satisfying, result in the theory of concurrency is the characterization theorem of bisimulation equivalence [32, 35] in terms of Hennessy-Milner logic (HML) due to Hennessy and Milner [21]. This theorem states that two bisimilar processes satisfy the same formulae in Hennessy-Milner logic, and if the processes satisfy a technical finiteness condition, then they are also bisimilar when they satisfy the same formulae in the logic. This means that, for bisimilarity and HML, logical equivalence coincides with behavioural equivalence, and that, whenever two processes are *not* equivalent, we can always find a formula in HML that witnesses a reason why they are not. This distinguishing formula is useful for debugging purposes, and can be algorithmically constructed for finite processes—see, e.g., [26, 31]. (Algorithms for computing such distinguishing formulae for strong and weak bisimilarity are implemented in tools like the Edinburgh Concurrency Workbench.)

The characterization theorem of Hennessy and Milner is, however, less useful if we are interested in using it directly to establish when two processes are behaviourally equivalent using model checking. Indeed, that theorem seems to indicate that to show that two processes are equivalent we need to check that they satisfy the same formulae expressible in the logic, and there are countably many such formulae, even modulo logical equivalence. Is it possible to find a *single* formula that characterizes the bisimulation equivalence class of a process p—in the sense that any process is bisimilar to p if, and only if, it affords that property? Such a formula, if it exists, is called a *characteristic formula*. When a characteristic formula for a process modulo a given notion of behavioural equivalence or preorder can be algorithmically constructed, implementation verification can be reduced to model checking, and, as the sub-title of our paper indicates, we can translate automata to logic. (An investigation of the model checking problems that can be reduced to implementation verification may, for instance, be found in the paper [6].)

To sum up the above discussion, characteristic formulae provide a very elegant connection between automata and logic, and between implementation verification and model checking. But, can they be constructed for natural, and suitably expressive, automata-based models and known logics of computation? To the best of our knowledge, this natural question was first addressed in the literature on concurrency theory in the paper [20]. In that paper, Graf and Sifakis offered a translation from recursion-free terms of Milner's CCS [32] into formulae of a modal language representing their equivalence class with respect to observational congruence.

Can one characterize the equivalence class of an arbitrary finite process—for instance one described in the regular fragment of CCS—up to bisimilarity using HML? The answer is negative because each formula in that logic can only describe a finite fragment of the initial behaviour of a process—see, for instance, [1] for a textbook presentation. However, in the rest of this survey, we shall show that adding a facility for the recursive definition of formulae to (variants of) HML yields a logic that is powerful enough to support the construction of characteristic formulae for various types of finite processes modulo notions of behavioural equivalence or preorder. We shall focus on bisimilarity as

a notion of behavioural equivalence between processes, but the formalism that we consider is powerful enough to handle a wealth of other semantics from van Glabbeek's spectrum.

Roadmap of the Paper The rest of the paper is organized as follows. Section 2 discusses the construction of characteristic formulae for finite LTSs modulo bisimilarity using HML with a facility for the recursive definition of formulae as the logical specification language. We then proceed to show that both the logic and the approach used in that section in the setting of finite LTSs apply equally well to the formalism of timed automata. We conclude the paper with suggestions for further reading and further research on the topic of characteristic formulae (Section 4).

2 Characteristic Formulae for Finite LTSs Modulo Bisimilarity

As our first concrete example of characteristic formula construction, we now proceed to show how to build characteristic formulae for finite labelled transition systems modulo bisimilarity, using an extension of HML with a facility for the recursive definition of formulae.

Definition 2.1 [Labelled transition system] A *labelled transition system (LTS)* is a triple $(\mathsf{Proc}, \mathsf{Act}, \{\stackrel{a}{\rightarrow} | \ a \in \mathsf{Act}\})$, where:

- Proc is a set of *states* (or *processes*);
- Act is a set of *actions* (or *labels*);
- $\stackrel{a}{\rightarrow} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is a *transition relation*, for every $a \in \mathsf{Act}$. As usual, we shall use the more suggestive notation $s \stackrel{a}{\rightarrow} s'$ in lieu of $(s, s') \in \stackrel{a}{\rightarrow}$, and write $s \stackrel{a}{\not\rightarrow}$ (read 's refuses a') iff $s \stackrel{a}{\rightarrow} s'$ for no state s'.

A labelled transition system is finite if its sets of states and actions are both finite.

In this section, LTSs and their states will be considered modulo the classic notion of bisimulation equivalence [32, 35].

Definition 2.2 [Bisimulation and bisimilarity] A binary relation $\mathcal{R}$ over the set of states of an LTS is a *bisimulation* iff whenever $s_1 \, \mathcal{R} \, s_2$ and a is an action:

- if $s_1 \stackrel{a}{\rightarrow} s_1'$, then there is a transition $s_2 \stackrel{a}{\rightarrow} s_2'$ such that $s_1' \, \mathcal{R} \, s_2'$;
- if $s_2 \stackrel{a}{\rightarrow} s_2'$, then there is a transition $s_1 \stackrel{a}{\rightarrow} s_1'$ such that $s_1' \, \mathcal{R} \, s_2'$.

Two states s and s' are *bisimilar*, written $s \sim s'$, iff there is a bisimulation that relates them. Henceforth the relation $\sim$ will be referred to as *bisimulation equivalence* or *bisimilarity*.

Consider a finite LTS $(\mathsf{Proc}, \mathsf{Act}, \{\stackrel{a}{\rightarrow} \mid\ a \in \mathsf{Act}\})$. Our order of business will now be to show how to associate with each process $p \in \mathsf{Proc}$ a formula F_p in a suitable logic such that, for each process $q \in \mathsf{Proc}$,

$$p \text{ is bisimilar to } q \text{ if, and only if, } q \text{ 'affords the property } F_p\text{'}.$$

Such a formula F_p will be called the *characteristic formula* for process p.

The logic that we shall use to define such characteristic formulae is an extension of HML with recursively defined formulae.

Let $\mathcal{X}$ be a countably infinite collection of formula variables. The collection of formulae in Hennessy-Milner logic with recursion, denoted by $\mathcal{M}_{\mathcal{X}}$, is given by the following grammar:

$$F ::= X \mid tt \mid ff \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \langle a \rangle F \mid [a]F \ ,$$

where X ranges over $\mathcal{X}$ and $a \in \mathsf{Act}$.

The meaning of formula variables is specified by means of a declaration. A *declaration* is a function $D : \mathcal{X} \rightarrow \mathcal{M}_{\mathcal{X}}$ that associates a formula $D(X)$ with each variable X. Intuitively, if $D(X) = F$, then X stands for the largest solution of the equation $X = F$. In general, we shall only be interested in the restriction of a declaration to a finite collection of formula variables. In that case, we write

$$\begin{array}{rcl} X_1 & = & F_{X_1} \\ & \vdots & \\ X_n & = & F_{X_n} \ , \end{array}$$

where $\{X_1, \ldots, X_n\}$ is a set of variables in $\mathcal{X}$, and, for $1 \leq i \leq n$, the formula $D(X_i) = F_{X_i}$ can only contain variables from $\{X_1, \ldots, X_n\}$.

Definition 2.3 [Satisfaction relation] The binary relation $\models$ relating processes in Proc to formulae in $\mathcal{M}_{\mathcal{X}}$ is the largest relation satisfying the following clauses:

- $p \models tt$, for each p,
- $p \models ff$, for no p,
- $p \models F \wedge G$ implies $p \models F$ and $p \models G$,
- $p \models F \vee G$ implies $p \models F$ or $p \models G$,
- $p \models \langle a \rangle F$ implies $p \stackrel{a}{\rightarrow} p'$ for some p' such that $p' \models F$,
- $p \models [a]F$ implies $p' \models F$, for each p' such that $p \stackrel{a}{\rightarrow} p'$, and
- $p \models X$ implies $p \models D(X)$.

The existence of the relation $\models$ is guaranteed by classic fixed point theory [25, 41].

Semantically, a formula F that may contain occurrences of a finite subset $\{X_1, \ldots, X_n\}$ of variables in $\mathcal{X}$ is interpreted as a function $\mathcal{O}_F$ that, given a vector of sets of processes $(S_1, \ldots, S_n)$ that are assumed to satisfy the formulae $X_1, \ldots, X_n$, returns the set of processes that satisfy F. Similarly, a mutually recursive system of equations of the form

$$\begin{aligned} X_1 &= F_{X_1} \\ &\vdots \\ X_n &= F_{X_n} \ , \end{aligned}$$

where $\{X_1, \ldots, X_n\}$ is a set of variables in $\mathcal{X}$, and the formula F_{X_i} $(1 \leq i \leq n)$ can only contain variables from $\{X_1, \ldots, X_n\}$, is interpreted as the largest vector of sets of processes $(S_1, \ldots, S_n)$ such that

$$\begin{aligned} S_1 &= \mathcal{O}_{F_{X_1}}(S_1, \ldots, S_n) \\ &\vdots \\ S_n &= \mathcal{O}_{F_{X_n}}(S_1, \ldots, S_n) \ . \end{aligned}$$

This means that the logic we have just introduced enriches classic HML with greatest fixed points, and is thus a fragment of Kozen's μ-calculus [27]. We refer the reader to, for instance, [1] for more details on the semantics of Hennessy-Milner logic with recursion.

We are now ready to define the characteristic formula for each process $p \in \mathsf{Proc}$. A characteristic formula for a process has to describe both which actions the process *can perform*, which actions it *cannot perform* and what happens to it *after it has performed* each action. Let

$$Der(a, p) = \{p' \mid p \xrightarrow{a} p'\}$$

be the set of states that can be reached from p by performing action a. If $p' \in Der(a, p)$ and p' has a characteristic property $X_{p'}$, then p has the property $\langle a \rangle X_{p'}$. We therefore have that

$$p \models \bigwedge_{a, p' . p \xrightarrow{a} p'} \langle a \rangle X_{p'} \ .$$

Furthermore, if $p \xrightarrow{a} p'$ then $p' \in Der(a, p)$. Therefore p has the property

$$[a]\Big(\bigvee_{p' . p \xrightarrow{a} p'} X_{p'} \Big) \ ,$$

for each action a. The above property states that, by performing action a, process p (and any other process that is bisimilar to it) must become a process

satisfying the characteristic property of a state in $Der(a,p)$. (Note that if $p \stackrel{a}{\not\rightarrow}$, then $Der(a,p)$ is empty. In that case, since an empty disjunction is just the formula ff, the above formula becomes simply $[a]ff$—which is what we would expect.)

Since action a is arbitrary, we have that

$$p \models \bigwedge_a [a]\Big(\bigvee_{p'.p \stackrel{a}{\rightarrow} p'} X_{p'} \Big) \ .$$

If we summarize the above requirements, we may conclude that

$$p \models \bigwedge_{a,p'.p \stackrel{a}{\rightarrow} p'} \langle a \rangle X_{p'} \ \wedge \ \bigwedge_a [a]\Big(\bigvee_{p'.p \stackrel{a}{\rightarrow} p'} X_{p'} \Big) \ .$$

As this property is apparently a complete description of the behaviour of process p, this is our candidate for its characteristic property. X_p is therefore defined as a solution to the equational system obtained by giving the following equation for each $q \in \mathsf{Proc}$:

$$X_q = \bigwedge_{a,q'.q \stackrel{a}{\rightarrow} q'} \langle a \rangle X_{q'} \ \wedge \ \bigwedge_a [a]\Big(\bigvee_{q'.q \stackrel{a}{\rightarrow} q'} X_{q'} \Big) \ .$$

Theorem 2.1 Let $(\mathsf{Proc}, \mathsf{Act}, \{\stackrel{a}{\rightarrow} \mid a \in \mathsf{Act}\})$ be a finite transition system and, for each $p \in \mathsf{Proc}$, let X_p be defined by

$$X_p = \bigwedge_{a,p'.p \stackrel{a}{\rightarrow} p'} \langle a \rangle X_{p'} \ \wedge \ \bigwedge_a [a]\Big(\bigvee_{p'.p \stackrel{a}{\rightarrow} p'} X_{p'} \Big) \ .$$

Then X_p is the characteristic property for p—that is, $q \models X_p$ iff $p \sim q$, for each $q \in \mathsf{Proc}$.

To the best of our knowledge, the above theorem was first proved in the MSc thesis [23]. The results in that study were later generalized to a family of bisimulation-like preorders and equivalences by Steffen and the second author in [40], which is by now the standard reference for characteristic formulae for finite labelled transition systems.

Remark 2.1 The above construction and theorem apply equally well to finitely branching LTSs with countably many states. However, in that case, the resulting characteristic formulae will use infinite systems of recursive equations.

3 Characteristic Formulae for Timed Automata

The approach to (automated) verification where the problem of checking behavioral relations between finite LTSs is reduced to model checking is advocated

by Cleaveland and Steffen in [11, 12]. In their approach, the language being model checked is a logic equivalent in expressive power to the alternation-free fragment of the modal μ-calculus [27]. The efficiency of this approach hinges on the following two facts:

1. the characteristic formula associated with a finite labelled transition system has size that is linear in that of the original LTS, and

2. the time complexity of determining whether a process satisfies a formula is proportional to the product of the sizes of the process and the formula.

The resulting algorithm offered in [12] is still considered to be one of the most efficient for checking behavioural preorders.

In the setting of modelling and verification for real-time systems, a characteristic formula construction for timed bisimulation equivalence over timed automata [4] has been offered in [29]. In *op. cit.*, Laroussinie, Larsen and Weise have proposed the logic L_ν, a real-time version of Hennessy-Milner Logic [21] with greatest fixed points. Moreover, they have shown that its associated model checking problem is decidable, and that this logic is sufficiently expressive for representing any timed automaton as a single characteristic L_ν formula. Such a formula uniquely characterizes the timed automaton up to timed bisimilarity.

The characteristic formula construction offered in [29], together with a model checking algorithm for the logic L_ν, yields an algorithm for checking whether two timed automata are timed bisimilar, which may be seen as an implementation of the approach advocated in [12] in a real-time setting. Unfortunately, however, the characteristic formula construction for timed automata proposed in [29] produces formulae whose size is exponential in that of the original automaton, and this makes its use in checking timed bisimilarity for timed automata infeasible. The exponential blow-up involved in the characteristic formula construction from *op. cit.* is due to the fact that the formula is essentially constructed by applying the standard, untimed construction presented in Section 2 to the region graph associated with the timed automaton [4]. As shown by Alur and Dill in [4], the size of the region graph is exponential in that of the original timed automaton.

Our order of business in this section will be to present characteristic formula constructions for timed automata using the logic L_ν that, like those in the untimed setting and unlike that offered in [29], yield formulae whose size is linear with respect to that of the timed automaton they characterize.

We limit ourselves to presenting characteristic formula constructions for timed bisimilarity [42], and for the faster-than preorder proposed by Moller and Tofts in [33]. Constructions of characteristic formulae for some other behavioural relations over timed automata, as well as proofs of the results we mention in what follows, may be found in [2].

Timed Labelled Transition Systems Let Act be a finite set of *actions*, ranged over by a, and let $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ denote the sets of natural and non-negative

real numbers, respectively. We use $\mathcal{L}$ to stand for the union of Act and $\mathbb{R}_{\geq 0}$. The meta-variable α will range over $\mathcal{L}$.

Definition 3.1 A *timed labelled transition system* (TLTS) is a structure $\mathcal{T} = (\mathcal{S}, \mathcal{L}, s^0, \longrightarrow)$ where $\mathcal{S}$ is a set of *states*, $s^0 \in \mathcal{S}$ is the initial state, and $\longrightarrow \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is a transition relation satisfying the following properties:

- (Time Determinism) for every $s, s', s'' \in \mathcal{S}$ and $d \in \mathbb{R}_{\geq 0}$, if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$, then $s' = s''$;
- (Time Additivity) for every $s, s'' \in \mathcal{S}$ and $d_1, d_2 \in \mathbb{R}_{\geq 0}$, $s \xrightarrow{d_1+d_2} s''$ iff $s \xrightarrow{d_1} s' \xrightarrow{d_2} s''$, for some $s' \in \mathcal{S}$;
- (0-Delay) for every $s, s' \in \mathcal{S}$, $s \xrightarrow{0} s'$ iff $s = s'$.

The axioms of time determinism, time additivity and 0-delay are standard in the literature on Yi's TCCS (see, for instance, [42]).

Timed Automata Let C be a set of clocks. We use $\mathcal{B}(C)$ to denote the set of boolean expressions over atomic formulae of the form $x \bowtie m$ and $x - y \bowtie m$, with $x, y \in C$, $m \in \mathbb{N}$, and $\bowtie \in \{<, >, =\}$. Expressions in $\mathcal{B}(C)$ are interpreted over the collection of time assignments. A *time assignment*, or *valuation*, v for C is a function from C to $\mathbb{R}_{\geq 0}$. Given an expression $g \in \mathcal{B}(C)$ and a time assignment v, we write $v \models g$ if v satisfies g. Note that $\mathcal{B}(C)$ is closed under negation. For every time assignment v and $d \in \mathbb{R}_{\geq 0}$, we use $v + d$ to denote the time assignment which maps each clock $x \in C$ to the value $v(x) + d$. Two assignments u and v are said to agree on the set of clocks C' iff they assign the same real number to every clock in C'. For every subset C' of clocks, $v[C' \mapsto 0]$ denotes the assignment for C which maps each clock in C' to the value 0 and agrees with v over $C \backslash C'$.

Definition 3.2 A *timed automaton* is a quintuple $A = (\mathsf{Act}, N, n_0, C, E)$ where N is a finite set of *nodes*, n_0 is the *initial node*, C is a finite set of *clocks*, and $E \subseteq N \times N \times \mathsf{Act} \times 2^C \times \mathcal{B}(C)$ is a set of *edges*. The quintuple $e = (n, n_e, a, r_e, g_e) \in E$ stands for an edge from node n to node n_e (the *target* of e) with action a, where r_e denotes the set of clocks to be reset to 0 and g_e is the enabling condition (or *guard*) over the clocks of A.

A *state* of a timed automaton A is a pair (n, v) where n is a node of A and v is a time assignment for C. The initial state of A is $(n_0, [C \mapsto 0])$ where n_0 is the initial node of A, and $[C \mapsto 0]$ is the time assignment mapping all clocks in C to 0.

The operational semantics of a timed automaton A is given by the timed labelled transition system $\mathcal{T}_A = (\mathcal{S}_A, \mathcal{L}, s^0_A, \longrightarrow)$, where $\mathcal{S}_A$ is the set of states

of A, s^0_A is the initial state of A, and $\longrightarrow$ is the transition relation defined as follows:

$$(n, v) \xrightarrow{a} (n', v') \text{ iff } \exists e = (n, n', a, r_e, g_e) \in E.\ v \models g_e \wedge v' = v[r_e \mapsto 0]$$
$$(n, v) \xrightarrow{d} (n', v') \text{ iff } n = n' \text{ and } v' = v + d \ ,$$

where $a \in \mathsf{Act}$ and $d \in \mathbb{R}_{\geq 0}$.

The Logic L_ν The logic L_ν is a real-time version of Hennessy-Milner Logic with greatest fixed points that stems from [29]. We now briefly review its syntax and semantics for the sake of completeness.

Definition 3.3 [Syntax of L_ν] Let K be a finite set of formula clocks, **Id** a finite set of identifiers and k a non-negative integer. The set L_ν of formulae over K, **Id** and largest constant k is generated by the abstract syntax below, with φ and ψ ranging over L_ν:

$$\varphi ::= \ tt \mid ff \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \exists\varphi \mid \forall\varphi \mid \langle a\rangle\varphi \mid [a]\varphi \mid$$
$$x \ \underline{\texttt{in}} \ \varphi \mid x \bowtie m \mid x + m \bowtie y + \ell \mid Z \ ,$$

where $a \in \mathsf{Act}$, $x, y \in K$, $\ell, m \in \{0, \ldots, k\}$, $\bowtie \in \{=, <, \leq, >, \geq\}$ and $Z \in \mathbf{Id}$.

The logic L_ν allows for the recursive definition of formulae by including a finite set **Id** of identifiers. The formula associated with each of the identifiers is specified by a declaration D, i.e. D assigns a formula of L_ν to each identifier. For an identifier Z we let $Z = \varphi$ denote $D(Z) = \varphi$. Intuitively Z will stand for the largest solution of the equation $Z = \varphi$. We refer the interested reader to [1, 29] for more information on L_ν.

Given a timed automaton A, whose set of clocks C is disjoint from K, we interpret the formulae in L_ν over extended states. An *extended state* of A is a pair (n, vu), where (n, v) is a state of A, u is a time assignment for K, and we use vu for the assignment over $C \cup K$ that agrees with v over C and with u over K.

Definition 3.4 [Semantics of L_ν] Let A be a timed automaton and D a declaration. The satisfaction relation $\models_D$ is the largest relation satisfying the following

implications:

$$
\begin{array}{rcl}
(n, vu) \models_D tt & \Rightarrow & \text{true} \\
(n, vu) \models_D ff & \Rightarrow & \text{false} \\
(n, vu) \models_D \varphi \wedge \psi & \Rightarrow & (n, vu) \models_D \varphi \text{ and } (n, vu) \models_D \psi \\
(n, vu) \models_D \varphi \vee \psi & \Rightarrow & (n, vu) \models_D \varphi \text{ or } (n, vu) \models_D \psi \\
(n, vu) \models_D \exists\varphi & \Rightarrow & \exists d \in \mathbb{R}_{\geq 0}.\ (n, (v+d)(u+d)) \models_D \varphi \\
(n, vu) \models_D \forall\varphi & \Rightarrow & \forall d \in \mathbb{R}_{\geq 0}.\ (n, (v+d)(u+d)) \models_D \varphi \\
(n, vu) \models_D \langle a \rangle\varphi & \Rightarrow & \exists (n', v').\ (n, v) \xrightarrow{a} (n', v') \text{ and } (n', v'u) \models_D \varphi \\
(n, vu) \models_D [a]\varphi & \Rightarrow & \forall (n', v').\ (n, v) \xrightarrow{a} (n', v') \text{ implies } (n', v'u) \models_D \varphi \\
(n, vu) \models_D x \bowtie m & \Rightarrow & u(x) \bowtie m \\
(n, vu) \models_D x + m \bowtie y + \ell & \Rightarrow & u(x) + m \bowtie u(y) + \ell \\
(n, vu) \models_D x \ \underline{\texttt{in}}\ \varphi & \Rightarrow & (n, vu') \models_D \varphi \text{ where } u' = u[\{x\} \mapsto 0] \\
(n, vu) \models_D Z & \Rightarrow & (n, vu) \models_D D(Z)\ .
\end{array}
$$

Again, the existence of $\models_D$ follows from standard fixed point theory [25, 41].

In the remainder of this section, we shall use the logic L_ν to construct characteristic formulae for timed automata with respect to timed bisimilarity [42], and for the faster-than preorder [33].

Two Timed Behavioural Relations Timed bisimulation stems from [42]. It is the obvious adaptation to the timed setting of the classic definition presented in Definition 2.2.

Definition 3.5 [Timed bisimulation] Let $\mathcal{T} = (\mathcal{S}, \mathcal{L}, s^0, \longrightarrow)$ be a TLTS. A *timed bisimulation* is a relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ such that whenever $s_1 \mathcal{R} s_2$ and $\alpha \in \mathcal{L}$, then:

- If $s_1 \xrightarrow{\alpha} s_1'$ then $s_2 \xrightarrow{\alpha} s_2'$ for some s_2' such that $s_1' \mathcal{R} s_2'$.
- If $s_2 \xrightarrow{\alpha} s_2'$ then $s_1 \xrightarrow{\alpha} s_1'$ for some s_1' such that $s_1' \mathcal{R} s_2'$.

For states s_1, s_2, we write $s_1 \sim_T s_2$ iff there exists a timed bisimulation $\mathcal{R}$ with $s_1 \mathcal{R} s_2$.

Moller and Tofts [33] have proposed a preorder on processes that distinguishes functionally behaviourally equivalent processes which operate at different speed. Their original proposal applied to their version of timed CCS, but it is simple enough to adapt it to the setting of TLTSs.

Definition 3.6 [Faster-than bisimulation] Let $\mathcal{T} = (\mathcal{S}, \mathcal{L}, s^0, \longrightarrow)$ be a TLTS. A *faster-than bisimulation* is a relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ such that whenever $s_1 \mathcal{R} s_2$, $a \in \mathsf{Act}$ and $d \in \mathbb{R}_{\geq 0}$ then:

1. if $s_1 \stackrel{a}{\rightarrow} s_1'$ then there are $d \in \mathbb{R}_{\geq 0}, s_1'', s_2'$ and s_2'' such that $s_1' \stackrel{d}{\rightarrow} s_1''$, $s_2 \stackrel{d}{\rightarrow} s_2' \stackrel{a}{\rightarrow} s_2''$, and $s_1' \mathcal{R} s_2'$;

2. If $s_2 \stackrel{a}{\rightarrow} s_2'$ then $s_1 \stackrel{a}{\rightarrow} s_1'$ for some s_1' such that $s_1' \mathcal{R} s_2'$;

3. If $s_1 \stackrel{d}{\rightarrow} s_1'$ then $s_2 \stackrel{d}{\rightarrow} s_2'$ for some s_2' such that $s_1' \mathcal{R} s_2'$;

4. If $s_2 \stackrel{d}{\rightarrow} s_2'$ then $s_1 \stackrel{d}{\rightarrow} s_1'$ for some s_1' such that and $s_1' \mathcal{R} s_2'$.

For states s_1, s_2, we write $s_1 \sqsubseteq_{FT} s_2$ iff there exists a faster-than bisimulation $\mathcal{R}$ with $s_1 \mathcal{R} s_2$.

It is well-known that $\sqsubseteq_{FT}$ is a preorder, and is the largest faster-than bisimulation. Similarly, $\sim_T$ is an equivalence relation, and is the largest timed bisimulation. Both of the previously defined behavioural relations can be lifted to the setting of timed automata as follows.

Definition 3.7 Let A, B be two timed automata. We write $A \sim_T B$ iff $s_A^0 \sim_T s_B^0$ in the TLTS that results by taking the disjoint union of $\mathcal{T}_A$ and $\mathcal{T}_B$. Similarly, we write $A \sqsubseteq_{FT} B$ iff $s_A^0 \sqsubseteq_{FT} s_B^0$ in the TLTS that results by taking the disjoint union of $\mathcal{T}_A$ and $\mathcal{T}_B$.

Characteristic Formula Constructions To increase the readability of the characteristic formulae we make use of some derived constructs in the logic L_ν. These we now present for the sake of clarity.

For a reset set $r = \{x_1, \ldots, x_k\}$, we use the abbreviation $r \ \underline{\mathtt{in}} \ \varphi$ to stand for the formula inductively defined thus:

$$\begin{aligned} \emptyset \ \underline{\mathtt{in}} \ \varphi &= \varphi \\ \{x_1, \ldots, x_k\} \ \underline{\mathtt{in}} \ \varphi &= x_1 \ \underline{\mathtt{in}} \ (\{x_2, \ldots, x_k\} \ \underline{\mathtt{in}} \ \varphi) \qquad (k \geq 1) \ . \end{aligned}$$

Note that the order of the clocks is arbitrary because $x \ \underline{\mathtt{in}} \ (y \ \underline{\mathtt{in}} \ \varphi)$ is logically equivalent to $y \ \underline{\mathtt{in}} \ (x \ \underline{\mathtt{in}} \ \varphi)$.

The expression $g \Rightarrow \varphi$ will stand for $\overline{g} \vee \varphi$, where $\overline{g}$ is the negation of the guard g. This is a formula in L_ν because the collection of guards is closed under negation.

In the remainder of this section, we shall implicitly assume a given timed automaton A, for which the characteristic formulae will be defined. Given a node n in A, and action a, we use $E(n, a)$ to stand for the set of a-labelled edges stemming from node n.

We first consider timed bisimilarity. A formula characterizing a node of a timed automaton up to timed bisimilarity should offer a description of:

1. all the actions that are enabled in the node,

2. which node is entered by taking a given transition, together with the resets associated with it, and

3. the fact that arbitrary delays are allowed in the node.

The resulting characteristic formula is presented below, where we consider each $\Phi(n)^{\sim_T}$ to be an identifier. The formula consists of three sets of conjuncts, each associated to one of the above properties, for each node n of a timed automaton A:

$$\begin{aligned}\Phi^{\sim_T}(n) \quad = \quad & (\bigwedge_{a\in \mathsf{Act}} \bigwedge_{e\in E(n,a)} g_e \Rightarrow (\langle a\rangle\ r_e\ \underline{\mathtt{in}}\ \Phi^{\sim_T}(n_e))) \wedge \\ & \bigwedge_{a\in \mathsf{Act}} [a](\bigvee_{e\in E(n,a)} g_e \wedge (r_e\ \underline{\mathtt{in}}\ \Phi^{\sim_T}(n_e))) \wedge \\ & \mathbb{W}\Phi^{\sim_T}(n)\ ,\end{aligned}$$

where n is a node of A, $e = (n, n_e, a, r_e, g_e)$, and we recall that $E(n, a)$ denotes the set of a-labelled edges from node n. We shall use $D_A^{\sim_T}$ to denote the declaration that consists of the equations above, one for each node of A.

Theorem 3.1 Let A, B be timed automata with disjoint sets of clocks. Let n be a node of A and m be a node of B. Assume that u and v are valuations for the clocks of A and B, respectively. Then

$$(n, u) \sim_T (m, v) \text{ iff } (m, vu) \models \Phi^{\sim_T}(n) \quad ,$$

where $(m, vu) \models \Phi^{\sim_T}(n)$ holds with respect to the declaration $D_A^{\sim_T}$.

In the characteristic formula construction for timed bisimilarity, no use was made of the existential modality $\exists$ over delay transitions. The use of the $\exists$ modality will instead play a crucial role in the definition of the characteristic property for the faster-than bisimulation preorder. This we now proceed to present.

For every node n in a timed automaton A, we define:

$$\begin{aligned}\Phi^{\sqsubset}_{\sim FT}(n) \quad = \quad & (\bigwedge_{a\in \mathsf{Act}} \bigwedge_{e\in E(n,a)} g_e \Rightarrow (r_e\ \underline{\mathtt{in}}\ \exists\langle a\rangle\Phi^{\sqsubset}_{\sim FT}(n_e))) \wedge \\ & (\bigwedge_{a\in \mathsf{Act}} [a](\bigvee_{e\in E(n,a)} g_e \wedge (r_e\ \underline{\mathtt{in}}\ \Phi^{\sqsubset}_{\sim FT}(n_e))) \wedge \\ & \mathbb{W}\Phi^{\sqsubset}_{\sim FT}(n)\ ,\end{aligned}$$

where $e = (n, n_e, a, r_e, g_e)$ and $E(n, a)$ denotes the set of a labelled edges from node n. We shall use $D_A^{\sqsubset}{}_{\sim FT}$ to denote the declaration that consists of the equations above, one for each node of A.

Theorem 3.2 Let A, B be timed automata with disjoint sets of clocks. Let n be a node of A and m be a node of B. Assume that u and v are valuations for the clocks of A and B, respectively. Then

$$(n, u) \sqsubset_{\sim FT} (m, v) \text{ iff } (m, vu) \models \Phi^{\sqsubset}_{\sim FT}(n)\ ,$$

where $(m, vu) \models \Phi^{\sqsubset}_{\widetilde{\sim} FT}(n)$ holds with respect to the declaration $D^{\sqsubset}_{A\widetilde{\sim} FT}$.

It is interesting to remark that no other characteristic formula construction presented in [2] uses the the existential modality $\exists$ over delay transitions.

4 Suggestions for Further Reading

It is an instructive exercise to construct characteristic formulae for (states of) variations on finite LTSs in variants of HML with greatest fixed points for other (bi)simulation based behavioural relations in van Glabbeek's spectrum. This bears witness to the naturalness of this logic for the specification of behavioural properties of reactive systems modelled as LTSs. Examples of such constructions may be found in, for instance, [15, 34]. The former reference offers a characteristic formula in terms of the μ-calculus for each finite underspecified transition system—essentially a transition system where transitions may have sets of states as their target. The latter shows how to derive characteristic formulae in the μ-calculus for finite LTSs up to strong or weak bisimilarity directly from the characterization of those relations in terms of greatest fixed points.

All of the results we have surveyed in this paper show that, in light of its beautiful connection with bisimilarity, HML and its variations are prime candidates for logics in which to express characteristic properties for bisimulation-like relations. However, there are other options.

A classic, early result on characteristic formulae was obtained in the paper [7]. That paper shows that each finite Kripke structure can be characterized by a formula in Computation Tree Logic (CTL) [9] up to the natural variation on bisimilarity over Kripke structures.

Another characteristic formula result is presented in that paper for an equivalence between states that takes 'stuttering' into account. (This equivalence is closely related to van Glabbeek's and Weijland's branching bisimilarity [19], for which logical characterizations have been offered by De Nicola and Vaandrager in the paper [13].) Browne, Clarke and Grümberg show that equivalence classes of states in a finite Kripke structure modulo stuttering equivalence are completely characterized by next-time-free CTL formulae. (The absence of the next-time operator is expected in light of the inability of stuttering equivalence to 'count' the number of steps in a stuttering sequence.) Kučera and Schnoebelen have presented a refinement of the above classic theorem by Browne, Clarke and Grümberg in the paper [28]. To the best of our knowledge, it is not known whether the timed version of CTL presented in [3] is sufficiently expressive to characterize timed bisimilarity.

Larsen and Skou present a characteristic formula construction in a probabilistic variation on HML for a recursion-free calculus of probabilistic processes in [30].

Recently, Berger, Honda and Yoshida have been investigating the notion of *descriptive completeness* for logics of higher-order functions. For instance, in their paper [22], they show that, given a program in call-by-value PCF, one

can construct a Hoare triple representing the program's behaviour up to observational equivalence. This notion is the counterpart of characteristic formulae in the setting of program logics. Our readers will find further information on the, by now very substantial, body of work on this topic by Berger, Honda and Yoshida at the URL

`http://www.dcs.qmul.ac.uk/~kohei/logics/index.html`.

Despite all of the aforementioned, classic studies on the notion of characteristic formula, we feel that there is still scope for further investigation, both from the point of view of theory and from that of applications. We hope that this small paper will contribute to a renewed interest in this topic.

References

[1] L. Aceto, A. Ingolfsdottir, K. G. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007. To appear.

[2] L. Aceto, A. Ingolfsdottir, M. L. Pedersen, and J. Poulsen. Characteristic formulae for timed automata. *RAIRO, Theoretical Informatics and Applications*, 34(6):565–584, 2000.

[3] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Comput. Sci.*, 126(2):183–235, 25 Apr. 1994. Fundamental Study.

[5] J. C. Baeten and J. W. Klop, editors. *Proceedings CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[6] G. Boudol and K. G. Larsen. Graphical versus logical specifications. *Theoretical Comput. Sci.*, 106(1):3–20, 30 Nov. 1992.

[7] M. Browne, E. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Comput. Sci.*, 59(1,2):115–131, 1988.

[8] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[9] E. Clarke, E. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Prog. Lang. Syst.*, 8(2):244–263, 1986.

[10] E. Clarke, O. Gruemberg, and D. Peled. *Model Checking.* MIT Press, December 1999.

[11] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in Systems Design*, 2:121–147, 1993.

[12] R. Cleaveland and B. Steffen. Computing behavioural relations, logically. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings* 18^{th} *ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 127–138. Springer-Verlag, 1991.

[13] R. De Nicola and F. W. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 32(2):458–487, 1995.

[14] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Vol. B*, pages 995–1072. Elsevier, Amsterdam, 1990.

[15] H. Fecher and M. Steffen. Characteristic μ-calculus formula for an underspecified transition system. In *EXPRESS'04*, volume 128 of *Electronic Notes in Theoretical Computer Science*, pages 103–116. Elsevier Science Publishers, 2005.

[16] R. van Glabbeek. The linear time – branching time spectrum. In Baeten and Klop [5], pages 278–297.

[17] R. van Glabbeek. The linear time – branching time spectrum II: the semantics of sequential processes with silent moves. In E. Best, editor, *Proceedings CONCUR 93,* Hildesheim, Germany, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1993.

[18] R. van Glabbeek. The linear time–branching time spectrum. I. The semantics of concrete, sequential processes. In J. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001.

[19] R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

[20] S. Graf and J. Sifakis. A modal characterization of observational congruence on finite terms of CCS. *Information and Control*, 68(1–3):125–145, Jan./Feb./Mar. 1986.

[21] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[22] K. Honda, M. Berger, and N. Yoshida. Descriptive and relative completeness of logics for higher-order functions. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 360–371. Springer-Verlag, 2006.

[23] A. Ingolfsdottir, J. C. Godskesen, and M. Zeeberg. Fra Hennessy-Milner logik til CCS-processer. Master's thesis, Department of Computer Science, Aalborg University, 1987. In Danish.

[24] R. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.

[25] B. Knaster. Un théorème sur les fonctions d'ensembles. *Annales Societatis Mathematicae Polonae*, 6:133–134, 1928. In French.

[26] H. Korver. Computing distinguishing formulas for branching bisimulation. In K. Larsen and A. Skou, editors, *Proceedings of the Third Workshop on Computer Aided Verification,* Aalborg, Denmark, July 1991, volume 575 of *Lecture Notes in Computer Science*, pages 13–23. Springer-Verlag, 1992.

[27] D. Kozen. Results on the propositional mu-calculus. *Theoretical Comput. Sci.*, 27:333–354, 1983.

[28] A. Kučera and P. Schnoebelen. A general approach to comparing infinite-state systems with their finite-state specifications. *Theoretical Comput. Sci.*, 358(2-3):315–333, 2006.

[29] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic - and back. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science 1995, 20th International Symposium*, volume 969 of *Lecture Notes in Computer Science*, pages 529–539, Prague, Czech Republic, 28 Aug.–1 Sept. 1995. Springer.

[30] K. G. Larsen and A. Skou. Compositional verification of probabilistic processes. In R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 456–471. Springer-Verlag, 1992.

[31] T. Margaria and B. Steffen. Distinguishing formulas for free. In *Proc. EDAC–EUROASIC'93: IEEE European Design Automation Conference, Paris (France)*. IEEE Computer Society Press, February 1993.

[32] R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.

[33] F. Moller and C. Tofts. Relating processes with respect to speed. In J. Baeten and J. F. Groote, editors, *Proceedings CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 424–438. Springer-Verlag, 1991.

[34] M. Müller-Olm. Derivation of characteristic formulae. In *MFCS'98 Workshop on Concurrency (Brno, 1998)*, volume 18 of *Electron. Notes Theor. Comput. Sci.*, page 12 pp. (electronic). Elsevier, Amsterdam, 1998.

[35] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5^{th} *GI Conference,* Karlsruhe, Germany, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

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

[37] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004. This is a revised version of the original DAIMI memo [36].

[38] A. Pnueli. The temporal logic of programs. In *Proceedings 18^{th} Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[39] J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

[40] B. Steffen and A. Ingolfsdottir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

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

[42] W. Yi. Real-time behaviour of asynchronous agents. In Baeten and Klop [5], pages 502–520.

Recent BRICS Report Series Publications

RS-07-2 **Luca Aceto and Anna Ingólfsdóttir.** ***Characteristic Formulae: From Automata to Logic*****. January 2007. 18 pp.**

RS-07-1 **Daniel Andersson.** ***HIROIMONO is NP-complete.*** **January 2007. 8 pp.**

RS-06-19 **Michael David Pedersen.** ***Logics for The Applied π Calculus*****. December 2006. viii+111 pp.**

RS-06-18 **Małgorzata Biernacka and Olivier Danvy.** ***A Syntactic Correspondence between Context-Sensitive Calculi and Abstract Machines*****. dec 2006. iii+39 pp. Extended version of an article to appear in TCS. Revised version of BRICS RS-05-22.**

RS-06-17 **Olivier Danvy and Kevin Millikin.** ***A Rational Deconstruction of Landin's J Operator*****. December 2006. ii+37 pp. Revised version of BRICS RS-06-4. A preliminary version appears in the proceedings of IFL 2005, LNCS 4015:55–73.**

RS-06-16 **Anders Møller.** ***Static Analysis for Event-Based XML Processing*****. October 2006. 16 pp.**

RS-06-15 **Dariusz Biernacki, Olivier Danvy, and Kevin Millikin.** ***A Dynamic Continuation-Passing Style for Dynamic Delimited Continuations*****. October 2006. ii+28 pp. Revised version of BRICS RS-05-16.**

RS-06-14 **Giorgio Delzanno, Javier Esparza, and Jiří Srba.** ***Monotonic Set-Extended Prefix Rewriting and Verification of Recursive Ping-Pong Protocols*****. July 2006. 31 pp. To appear in ATVA '06.**

RS-06-13 **Jiří Srba.** ***Visibly Pushdown Automata: From Language Equivalence to Simulation and Bisimulation*****. July 2006. 21 pp. To appear in CSL '06.**

RS-06-12 **Kristian Støvring.** ***Higher-Order Beta Matching with Solutions in Long Beta-Eta Normal Form*****. June 2006. 13 pp. To appear in *Nordic Journal of Computing*, 2006.**

RS-06-11 **Kim G. Larsen, Ulrik Nyman, and Andrzej Wasowski.** ***An Interface Theory for Input/Output Automata*****. June 2006. 40 pp. Appears in Misra, Nipkow and Sekerinski, editors, *Formal Methods: 14th International Symposium*, FM '06 Proceedings, LNCS 4085, 2006, pages 82–97.**