Stuttering Congruence for χ

Bas Luttik[1,2] and Nikola Trčka[1]*

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
[2] CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands

Abstract. The language χ is a modeling and simulation language which is currently mainly used to analyse and optimize the performance of industrial systems. To be able to also verify functional properties of a system using a χ model, part of the language has been given a formal semantics. Rather than implementing a new model checker for χ, the philosophy is to provide automatic translations from χ into the specification languages of existing state-of-the-art model checkers such as, e.g., SPIN and UPPAAL.
In this paper, we propose for χ a notion of stuttering congruence, which is an adaptation of the notion of stuttering equivalence. We prove that our notion preserves the validity of $\text{CTL}^*_{-\text{X}}$ formulas, that it preserves deadlock, and that it is indeed a congruence with respect to the constructs of χ. We also indicate how our notion is to be used to establish confidence in the correctness of a translation from χ into PROMELA.

1 Introduction

The language χ [19] is a modeling language developed for detecting design flaws and for optimizing performance of industrial systems (machines, manufacturing lines, warehouses, factories, etc.) by simulation. Quite a few case studies have shown the usefulness of χ in an industrial context [12, 6, 10, 21]; simulation turns out to be a powerful technique for doing performance analysis such as approximating throughput and cycle time. However, for the verification of functional properties such as, e.g., deadlock freedom, simulation is less suitable. To be able to also do verification with χ, either verification tools have to be developed especially for χ, or existing verification tools and techniques have to be made available for use with χ. Currently, the latter approach is pursued.
The idea is to extend χ with facilities for doing formal verification by establishing a connection with other verification tools on the level of the specification language. That is, formal verification of a χ model is done by first translating it into the input language of some model checker and then performing the actual verification. Preferably, the translation closely resembles the original, so that a counterexamples produced by the model checker can be related to the original specification. The suitability of this approach was shown in [2, 3], where a χ

* Research supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.064.205

model of a turntable machine was translated to PROMELA [14], μCRL [1] and UPPAAL timed automata [15], and then verified in SPIN, CADP [9] and UPPAAL, respectively.

In [20], the translation of χ specifications into PROMELA is discussed in more generality. The translation proceeds in two phases. The first phase, which we call the *preprocessing phase*, consists of a transformation of the χ model in an attempt to eliminate all constructs that do not directly map to PROMELA constructs. For instance, χ has an explicit construct for parallel composition which facilitates nested parallelism, whereas PROMELA only allows the (implicit) parallel composition of sequential PROMELA processes; so in the preprocessing phase the nested parallelism in the χ model is eliminated. If the result after the preprocessing phase is a χ model that only has constructions with a direct translation into PROMELA, then it can be translated to a PROMELA model; this phase is called the *translation phase*.

The main difficulty for establishing the correctness of the whole translation is that usually the two languages do not have a formal semantics in common. An advantage of the two-phase approach sketched above then is that the preprocessing phase of the translation, which is usually the most involved part, takes place entirely within the realm of χ. Therefore, a correctness proof for this phase only involves the formal semantics of χ. An additional advantage of the two-phase approach is that the preprocessing phase (and its correctness proof) is potentially reusable, e.g., when defining a translation from χ to some other language.

The appropriate correctness criterion for a translation depends of course on the application. If the purpose is to establish that a χ model is deadlock-free, then the translation should preserve deadlock. If the purpose is to do LTL/CTL model checking, then the translation should preserve the validity of LTL/CTL formulas. In all cases, establishing the desired preservation of properties directly is usually cumbersome. It is often more convenient to relate the χ model and its transformation by establishing that they are related according to some behavioral equivalence pertaining to the operational semantics of χ. The purpose of this paper is to define a behavioral equivalence that can be used to establish the correctness of the preprocessing phase of translations of χ models into the language of state-based model checkers such as, e.g., SPIN or UPPAAL.

Of course, such an equivalence should then preserve the relevant properties; in our case it will preserve deadlock and the validity of (state-based) $\mathrm{CTL}^*_{-\mathrm{X}}$ formulas. (We do not require that the validity of formulas containing the next time operator X be preserved, in order to achieve sufficient flexibility for translations.) Its intended application for establishing the correctness of syntactic translations puts some further requirements on the notion. For instance, since the transformations are generally defined in a compositional manner, it is particularly convenient if the equivalence is a congruence with respect to the syntactic constructs of χ. Also, it is desirable that it is defined on the operational semantics of χ as directly as possible, i.e., it should be bisimulation-like. We propose a notion of stuttering congruence that meets these requirements. We present it in

the context of the language χ, but since the constructs of χ are fairly standard, we think that our notion can be of use for other languages too.
The paper is organized as follows. In Section 2 we present the syntax and the operational semantics of the discrete-event and untimed part of the language χ. We use the operational semantics to define when a χ-process has a deadlock, and we give the semantics of $\text{CTL}^*_{-\text{X}}$ formulas with respect to χ processes. In Section 3, we propose an adaptation of divergence blind stuttering bisimilarity [18]. We add to it a termination condition, which takes care of the distinction between successful and unsuccessful termination present in χ, and a divergence condition, which is needed both for the preservation of deadlock and preservation of $\text{CTL}^*_{-\text{X}}$. We prove that our version of stuttering bisimilarity is an equivalence relation and that it indeed preserves deadlock and the validity of (state-based) $\text{CTL}^*_{-\text{X}}$ formulas. In Section 4 we argue that stuttering bisimilarity as defined in Section 3 is not a congruence. So we adapt it further by excluding send and receive transitions as stuttering steps and by adding a root condition. The resulting notion we call stuttering congruence and we prove that it is indeed a congruence with respect to the syntactic constructs of the discrete-event, untimed part of χ. In Section 5 we briefly indicate how our notion of stuttering congruence can be used to establish part of the correctness of the translation proposed in [20]. The paper ends with a conclusion. For detailed proofs of the results in this paper we refer to the full version [16].

2 The language χ

In this section we present the syntax and operational semantics of χ. We also define the notion of deadlock and the semantics of $\text{CTL}^*_{-\text{X}}$ for χ processes. We use the formalization of χ proposed in [4], but without the time support and with a few minor differences that we shall mention on the fly.

2.1 Syntax and semantics

There are several predefined data types in χ, but they are not relevant for the present paper. For our purposes, it is enough to presuppose a set of variables V, a set of data values D, a set of data expressions E that includes D and V, and a set of boolean expressions B that includes the set of truth values $\{\mathit{true}, \mathit{false}\}$.

Definition 1. *A partial mapping $\sigma : V \rightharpoonup D$ with a finite domain (denoted* $\text{dom}(\sigma)$*) is called a* state. *The set of all states is denoted Σ.*

To correctly override global variables by local ones of the same name, we use the function $\gamma : \Sigma \times \Sigma \to \Sigma$ defined as:

$$\text{dom}(\gamma(\sigma_1, \sigma_2)) = \text{dom}(\sigma_1) \cup \text{dom}(\sigma_2)$$
$$\gamma(\sigma_1, \sigma_2)(x) = \begin{cases} \sigma_1(x), & \text{if } x \in \text{dom}(\sigma_1) \\ \sigma_2(x), & \text{if } x \in \text{dom}(\sigma_2) \backslash \text{dom}(\sigma_1). \end{cases}$$

We assume that σ also extends to data expressions ($\sigma : E \rightharpoonup D$) and to boolean expressions ($\sigma : B \to \{\mathit{true}, \mathit{false}\}$). In the latter case we require σ to be total.
We now give the syntax of χ. The set of atomic processes A, and the set of all χ process terms P, are generated by the following grammar:

$$\begin{array}{lll} a & ::= & \varepsilon \mid \delta \mid \mathit{skip} \mid x := e \mid m!e \mid m?x \\ p & ::= & a \mid b :\to p \mid p\,;p \mid p \,[\!]\, p \mid p^* \mid p \parallel p \mid [\![s \mid p]\!] \mid \partial(p) \ . \end{array}$$

Here $a \in A$, $p \in P$, $x \in V$, $e \in E$, $b \in B$, $s \in \Sigma$ and $m \in M$, where M is a set of channel names.
Elements of the set $C = P \times \Sigma$ we call *configurations*. They represent processes together with their context. If $c = \langle p, \sigma \rangle$, then σ is the *state* of c. The semantics of χ is given in terms of configurations.
We make a distinction between successful and unsuccessful termination. The statement $c\!\downarrow$ denotes that $c \in C$ successfully terminates. The statement $c \xrightarrow{a} c'$ denotes that $c \in C$ can execute the action a and transform into the configuration c'. The set of actions that can be performed (denoted $\mathcal{A}$) consists of the internal action τ, the assignment action $aa(x, d)$, the send action $sa(m, d)$, the receive action $ra(m, d)$ and the communication action $ca(m, d)$, where $x \in V$, $m \in M$ and $d \in D$.

Atomic processes We explain each atomic process informally; the operational rules are given in Table 1.
The constant δ stands for the deadlock process. It cannot execute an action nor terminate successfully. The empty process ε cannot do an action either, but it is considered successfully terminated. The *skip* process performs the internal action τ (and terminates successfully). The assignment process $x := e$ assigns to x the value of the expression e according to the current state. The send process $m!e$ outputs the value of e (in the current state) along channel m. The receive process $m?x$ inputs a value along channel m and assigns it to x.

Table 1. Operational semantics for atomic processes

$$\frac{}{\langle \varepsilon, \sigma \rangle\!\downarrow}\ 1 \qquad \frac{}{\langle \mathit{skip}, \sigma \rangle \xrightarrow{\tau} \langle \varepsilon, \sigma \rangle}\ 2 \qquad \frac{\sigma(e) = d}{\langle x := e, \sigma \rangle \xrightarrow{aa(x,d)} \langle \varepsilon, \gamma(\{x \mapsto d\}, \sigma) \rangle}\ 3$$

$$\frac{\sigma(e) = d}{\langle m!e, \sigma \rangle \xrightarrow{sa(m,d)} \langle \varepsilon, \sigma \rangle}\ 4 \qquad \frac{}{\langle m?x, \sigma \rangle \xrightarrow{ra(m,d)} \langle \varepsilon, \gamma(\{x \mapsto d\}, \sigma) \rangle}\ 5$$

Compound processes Here we give an informal explanation for each of the seven operators; the operational rules are given in Table 2.
The guarded process $b :\to p$ behaves as p when the value of the guard $b \in B$ is *true* (in the current state). The sequential composition $p\ ;\ q$ behaves as p

followed by the process q. The alternative composition $p \mathbin{[\!]} q$ stands for a non-deterministic choice between p and q. The process p^* behaves as p, executed zero (successful termination), or more times. The parallel composition operator $\parallel$ executes p and q concurrently in an interleaved fashion. In addition, if one of the processes can execute a send action and the other one can execute a receive action on the same channel, then they can also communicate, i.e. $p \parallel q$ can also execute the communication action on this channel. The scope operator is used for declarations of local variables. The process $\llbracket s \mid p \rrbracket$ behaves as p in a local state s. In contrast to [4] and [19], channel declarations are not allowed, i.e. channels are global. Finally, the encapsulation operator ∂ disables all send and receive actions of a process. This is slightly more restrictive than in [4] and [19] where ∂ is parameterized by a set of actions that should be blocked, but corresponds to current practice.

Table 2. Operational semantics for composed processes

$$\frac{\sigma(b) = \mathit{true},\ \langle p, \sigma\rangle{\downarrow}}{\langle b :\to p, \sigma\rangle{\downarrow}}\ 6 \qquad \frac{\sigma(b) = \mathit{true},\ \langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle b :\to p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}\ 7$$

$$\frac{\langle p, \sigma\rangle{\downarrow},\ \langle q, \sigma\rangle{\downarrow}}{\langle p\,;q, \sigma\rangle{\downarrow}}\ 8 \qquad \frac{\langle p, \sigma\rangle{\downarrow},\ \langle q, \sigma\rangle \xrightarrow{a} \langle q', \sigma'\rangle}{\langle p\,;q, \sigma\rangle \xrightarrow{a} \langle q', \sigma'\rangle}\ 9 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle p\,;q, \sigma\rangle \xrightarrow{a} \langle p'\,;q, \sigma'\rangle}\ 10$$

$$\frac{\langle p, \sigma\rangle{\downarrow}}{\langle p \mathbin{[\!]} q, \sigma\rangle{\downarrow},\ \langle q \mathbin{[\!]} p, \sigma\rangle{\downarrow}}\ 11 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle p \mathbin{[\!]} q, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle,\ \langle q \mathbin{[\!]} p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}\ 12$$

$$\frac{}{\langle p^*, \sigma\rangle{\downarrow}}\ 13 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle p^*, \sigma\rangle \xrightarrow{a} \langle p'\,;p^*, \sigma'\rangle}\ 14$$

$$\frac{\langle p, \sigma\rangle{\downarrow},\ \langle q, \sigma\rangle{\downarrow}}{\langle p \parallel q, \sigma\rangle{\downarrow},\ \langle q \parallel p, \sigma\rangle{\downarrow}}\ 15 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle p \parallel q, \sigma\rangle \xrightarrow{a} \langle p' \parallel q, \sigma'\rangle,\ \langle q \parallel p, \sigma\rangle \xrightarrow{a} \langle q \parallel p', \sigma'\rangle}\ 16$$

$$\frac{\langle p, \sigma\rangle \xrightarrow{sa(m,d)} \langle p', \sigma\rangle,\ \langle q, \sigma\rangle \xrightarrow{ra(m,d)} \langle q', \sigma'\rangle}{\langle p \parallel q, \sigma\rangle \xrightarrow{ca(m,d)} \langle p' \parallel q', \sigma'\rangle,\ \langle q \parallel p, \sigma\rangle \xrightarrow{ca(m,d)} \langle q' \parallel p', \sigma'\rangle}\ 17$$

$$\frac{\langle p, \gamma(s, \sigma)\rangle{\downarrow}}{\langle \llbracket s \mid p \rrbracket, \sigma\rangle{\downarrow}}\ 18 \qquad \frac{\langle p, \gamma(s, \sigma)\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle \llbracket s \mid p \rrbracket, \sigma\rangle \xrightarrow{a} \langle \llbracket \sigma' \restriction \mathrm{dom}(s) \mid p' \rrbracket, \gamma(\sigma, \sigma' \restriction \mathrm{dom}(\sigma) \backslash \mathrm{dom}(s))\rangle,}\ 19$$

$$\frac{\langle p, \sigma\rangle{\downarrow}}{\langle \partial(p), \sigma\rangle{\downarrow}}\ 20 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle,\ a \notin \{sa(m,d), ra(m,d)\}}{\langle \partial(p), \sigma\rangle \xrightarrow{a} \langle \partial(p'), \sigma'\rangle}\ 21$$

2.2 Deadlock and CTL$^*_{-\mathrm{X}}$ in χ

First we give some abbreviations: $c \longrightarrow c'$ denotes that there exists $a \in A$ such that $c \xrightarrow{a} c'$; $c \not\longrightarrow$ denotes that there does not exist a c' such that $c \longrightarrow c'$.
Now we define when a configuration has a deadlock.

Definition 2. *A configuration c has a* deadlock *iff there exist $c_0, \ldots, c_n \in C$ such that*

$$c_0 = c,\ \ c_0 \longrightarrow \cdots \longrightarrow c_n,\ \ c_n \not\longrightarrow \ \text{and}\ c_n \not\downarrow.$$

Next, we recall the formulas of the logic CTL$^*_{-\mathrm{X}}$ [7] and give their semantics.
Let AP be a set that we call the set of atomic propositions.

Definition 3. *The formulas of the logic* CTL$^*_{-\mathrm{X}}$ *are defined as follows:*

1. *every atomic proposition is a state formula;*
2. *if φ is a state formula, then $\neg\varphi$ is a state formula;*
3. *if φ_1 and φ_2 are state formulas, then $\varphi_1 \wedge \varphi_2$ is a state formula;*
4. *if ψ is a path formula, then $\exists\psi$ is a state formula;*
5. *if φ is a state formula, then φ is a path formula;*
6. *if ψ is a path formula, then $\neg\psi$ is a path formula;*
7. *if ψ_1 and ψ_2 are path formulas, then $\psi_1 \wedge \psi_2$ is a path formula;*
8. *if ψ_1 and ψ_2 are path formulas, then $\psi_1 \mathcal{U} \psi_2$ is a path formula.*

For the satisfaction relation we need the notion of a path.

Definition 4. *A* path *(from a configuration c_0) is an infinite sequence of configurations $c_0, c_1, c_2, \ldots$ such that either:*

- $c_0 \longrightarrow c_1 \longrightarrow c_2 \longrightarrow \cdots$ *or*
- $c_0 \longrightarrow \cdots \longrightarrow c_n$, $c_n \not\longrightarrow$ *and* $c_{i+1} = c_i$ *for all* $i \geq n$.

If π is a path $c_0, c_1, c_2, \ldots$ then π^i denotes the path $c_i, c_{i+1}, c_{i+2}, \ldots$.

We require that two configurations with the same state satisfy the same atomic propositions by assuming that for each state $\sigma \in \Sigma$ there is a mapping $val_\sigma : AP \longrightarrow \{true, false\}$.

Definition 5. *We simultaneously define the satisfaction of a state formula φ by a configuration c (notation: $c \models \varphi$) and the satisfaction of a path formula ψ by a path π (notation: $\pi \models \psi$) as follows:*

1. $c \models \alpha \in AP$ *iff* $val_c(\alpha) = true$
2. $c \models \neg\varphi$ *iff* $c \not\models \varphi$,
3. $c \models \varphi_1 \wedge \varphi_2$ *iff* $c \models \varphi_1$ *and* $c \models \varphi_2$,
4. $\pi \models \varphi$ *iff* $c \models \varphi$ *where c is the first configuration in path π,*
5. $c \models \exists\psi$ *iff there is a path π from c such that* $\pi \models \psi$,
6. $\pi \models \neg\psi$ *iff* $\pi \not\models \psi$,
7. $\pi \models \psi_1 \wedge \psi_2$ *iff* $\pi \models \psi_1$ *and* $\pi \models \psi_2$,
8. $\pi \models \psi_1 \mathcal{U} \psi_2$ *iff there exists $j \geq 0$ such that* $\pi^j \models \psi_2$ *and* $\pi^i \models \psi_1$ *for all* $i < j$.

3 Stuttering bisimilarity

Stuttering equivalence was originally proposed and proved to preserve the validity of $\text{CTL}^*_{-\text{X}}$ formulas by Browne, Clarke and Grumberg [5]. They define the notion on maximal paths associated with total Kripke structures, i.e., Kripke structures without deadlocked states. De Nicola and Vaandrager [18] drop the requirement that Kripke structures are total, and provide a definition of stuttering equivalence that proceeds via a notion of divergence blind stuttering bisimilarity defined on the Kripke structures themselves. Groote and Vaandrager [13] give an efficient algorithm for deciding this equivalence. For alternative definitions of stuttering equivalence see also [8, 17].
We take divergence blind stuttering bisimilarity of [18] as a starting point, and add to it two conditions:

1. a *termination condition* that ensures a proper handling of the distinction between successful and unsuccessful termination as it is present in χ; and
2. a *divergence condition* similar to one that appears in [11] to ensure the preservation of deadlock and the preservation of $\text{CTL}^*_{-\text{X}}$.

Remark 1. To obtain a notion that coincides with the notion of [5], instead of adding a divergence condition, de Nicola and Vaandrager define a divergence sensitive version of stuttering bisimilarity by extending Kripke structures with a fresh state that serves as a sink-state for deadlocked or divergent states. This approach is not suitable in our case, because it identifies deadlock and livelock, and because it is in conflict with our requirement, mentioned in the introduction, that the equivalence is defined directly on the operational semantics of χ.

Definition 6. *A symmetric relation $R \subseteq C \times C$ is a* stuttering bisimulation *iff, for all $(c, d) \in R$, c and d have the same state and:*

1. *if $c{\downarrow}$, then there exist $d_0, \ldots, d_n \in C$ such that*
$$d_0 = d,\ d_0 \rightarrow \cdots \rightarrow d_n,\ d_n{\downarrow} \text{ and } cRd_i \text{ for all } i \leq n,$$
2. *if $c \rightarrow c'$ for some $c' \in C$, then there exist $d_0, \ldots, d_n \in C$ such that*
$$d_0 = d,\ d_0 \rightarrow \cdots \rightarrow d_n,\ cRd_i \text{ for all } i \leq n-1, \text{ and } c'Rd_n,$$
3. *if there exists an infinite sequence $c_0, c_1, c_2, \ldots \in C$ such that*
$$c_0 = c,\ c_0 \rightarrow c_1 \rightarrow c_2 \rightarrow \cdots \text{ and } c_iRd \text{ for all } i \geq 0,$$
then there exist $d' \in C$ and $j > 0$ such that
$$d \rightarrow d' \text{ and } c_jRd'.$$

We refer to condition 1 as the termination condition*, to condition 2 as the* transfer condition*, and to condition 3 as the* divergence condition*.*

A non-empty and finite sequence of configurations we call a *block*. If $B = c_0, \ldots, c_n$ and $C = d_0, \ldots, d_m$ are blocks and R is a stuttering bisimulation, we write BRC when $c_0 R d_0$, $c_n R d_m$ and when, for all $i < n, j < m$, $c_i R d_j$ implies $c_{i+1} R d_j$ or $c_i R d_{j+1}$. Note that, BRC implies CRB.

Definition 7. *Let R be a stuttering bisimulation. Two sequences of configurations Ω_1 and Ω_2 are* R-corresponding *if:*

- *they are both finite and can be partitioned as $\Omega_1 = B^0, \ldots, B^K$ and $\Omega_2 = C^0, \ldots, C^K$ where $B^k R C^k$ for all $0 \le k \le K$; or*
- *they are both infinite and can be partitioned as $\Omega_1 = B^0, B^1, B^2 \ldots$ and $\Omega_2 = C^0, C^1, C^2, \ldots$ where $B^k R C^k$ for all $k \ge 0$.*

It is clear that R-correspondence is a symmetric relation. Also note that, if $c_0, \ldots, c_n$ and $d_0, \ldots, d_m$ (similarly for the infinite case) are R-corresponding then for all $i \le n$ there exists $j \le m$ and for all $j \le m$ there exists $i \le n$ such that $c_i R d_j$.
We now present some properties of a stuttering bisimulation R.

Lemma 1. *If cRd, then for every sequence of configurations $c_0, \ldots, c_n$ such that $c_0 = c$ and $c_0 \to \cdots \to c_n$ there exists an R-corresponding sequence $d_0, \ldots, d_m$ such that $d_0 = d$ and $d_0 \to \cdots \to d_m$.*

Proof. By induction on n. □

Lemma 2. *If cRd, then*

a. for every sequence of configurations $c_0, \ldots, c_n$ such that $c_0 = c$, $c_0 \to \cdots \to c_n$ and $c_n\downarrow$ there exists an R-corresponding sequence $d_0, \ldots, d_m$ such that

$$d_0 = d,\ d_0 \to \cdots \to d_m \text{ and } d_m\downarrow; \quad \text{and}$$

b. for every sequence of configurations $c_0, \ldots, c_n$ such that $c_0 = c$, $c_0 \to \cdots \to c_n$ and $c_n \not\to$ there exists an R-corresponding sequence $d_0, \ldots, d_m$ such that

$$d_0 = d,\ d_0 \to \cdots \to d_m,\ \text{and } d_m \not\to.$$

Proof. For both cases, use Lemma 1 to obtain a sequence $d_0, \ldots, d_l$ that R-corresponds to $c_0, \ldots, c_n$ and then extend it to $d_0, \ldots, d_l, \ldots, d_m$ by using Definition 6. (Use the termination condition for the first case, and the transfer and the divergence condition for the second case.) □

Lemma 3. *If cRd, then for every infinite sequence of configurations $c_0, c_1, c_2, \ldots$ such that $c_0 = c$ and $c_0 \to c_1 \to c_2 \to \cdots$, there exists an R-corresponding sequence of configurations $d_0, d_1, d_2, \ldots$ such that $d_0 = d$, $d_0 \to d_1 \to d_2 \to \cdots$.*

Proof. Construct infinite sequences of blocks $C^0, C^1, C^2, \ldots$ and $D^0, D^1, D^2, \ldots$ such that $C^k R D^k$ for all $k \ge 0$, with $C^0, C^1, C^2, \ldots$ a partitioning of $c_0, c_1, c_2, \ldots$ and $D^0, D^1, D^2, \ldots$ a partitioning of $d_0, d_1, d_2, \ldots$ such that $d_0 = d$ and $d_0 \to d_1 \to d_2 \to \cdots$. The construction is by induction on k, where each step delivers the blocks C^k and D^k and the first configurations of C^{k+1} and D^{k+1}. □

Definition 8. *Two configurations c and d are* stuttering bisimilar*, denoted $c \sim_{st} d$, if there exists a stuttering bisimulation R such that cRd.*

We now prove that stuttering bisimilarity is an equivalence relation. The usual way to prove that a bisimulation-like equivalence $\sim$ is transitive, is to suppose that $c \sim d$ and $d \sim e$ are witnessed by bisimulation relations R_1 and R_2 respectively, and then show that $R_1 \circ R_2$ is again a bisimulation relation. However, this method fails here, due to the nature of the divergence condition. We prove transitivity by showing that the transitive closure of a stuttering bisimulation is a stuttering bisimulation.

Lemma 4. *If R is a stuttering bisimulation, then so is $R^+ = \bigcup_{n\in\omega} R^n$.*

Proof. The transitive closure of any symmetric relation is symmetric so R^+ is symmetric. Prove first that for all n and all $(c, d) \in R^n$, c and d have the same state, (c, d) satisfies the termination and transfer condition and if $c_0 = c$, $c_0 \to c_1 \to c_2 \to \cdots$ then there exist $d_0, d_1, d_2, \ldots \in C$ such that $d_0 = d$, $d_0 \to d_1 \to d_2 \to \cdots$ and for all $j \geq 0$ there is $i \geq 0$ such that $c_i\ R^n\ d_j$. Since $R^+ = \bigcup_{n\in\omega} R^n$, it now follows immediately that R^+ satisfies the termination and transfer condition and that, for all $(c, d) \in R^+$, c and d have the same state. Prove that R^+ also satisfies the divergence condition by using that R^+ is symmetric and transitive. □

Theorem 1. *Stuttering bisimilarity on configurations is an equivalence relation.*

Proof. The set $\{(c, c) \mid c \in C\}$ is clearly a stuttering bisimulation, so $\sim_{st}$ is reflexive. Furthermore, that $\sim_{st}$ is symmetric follows directly from the required symmetry of a stuttering bisimulation. It remains to prove transitivity.
Suppose $c \sim_{st} d$ and $d \sim_{st} e$. Then, there exist stuttering bisimulations R_1 and R_2 such that $c\,R_1\,d$ and $d\,R_2\,e$. Let $R = R_1 \cup R_2$. It is not hard to show that R is also a stuttering bisimulation. By Lemma 4, so is R^+. Since $R \subseteq R^+$, $c\,R^+\,d$ and $d\,R^+\,e$. By the transitivity of R^+, we conclude $c\,R^+\,e$, and hence $c \sim_{st} e$. □

Corollary 1. *$\sim_{st}$-correspondence is an equivalence relation.*

In the remainder of this section we establish that stuttering bisimilarity preserves deadlock and the validity of $\text{CTL}^*_{-\text{X}}$ formulas.

Theorem 2. *If $c \sim_{st} d$ then c has a deadlock iff d has a deadlock.*

Proof. Suppose c has a deadlock (when d has a deadlock the proof is similar). This means that there exist $c_0, \ldots, c_n \in C$ such that

$$c_0 = c,\ c_0 \to \cdots \to c_n,\ c_n \not\to \text{ and } c_n \not\downarrow.$$

By Lemma 2b, there exist $d_0, \ldots, d_m \in C$ such that

$$d_0 = d,\ d_0 \to \cdots \to d_m,\ d_m \not\to \text{ and } c_n R d_m.$$

Suppose $d_m\downarrow$. Then, there exist $c_n^0, \ldots, c_n^k \in C$ such that

$$c_n^0 \to \cdots \to c_n^k,\ \text{ and } c_n^k\downarrow.$$

This is however not possible (even when $k = 0$) because $c_n \not\to$ and $c_n \not\downarrow$. □

The following lemma plays a crucial role in the proof that stuttering bisimilarity preserves the validity of $\text{CTL}^*_{-\text{X}}$ formulas.

Lemma 5. *If $c \sim_{st} d$, then for every path from c there is a $\sim_{st}$-corresponding path from d.*

Proof. Let $c_0, c_1, c_2, \ldots$ be a path from c. There are two cases:

– if $c_0 \to \cdots \to c_n,\ c_n \not\to$ and $c_{i+1} = c_i$ for all $i \geq n$, then the statement follows directly from Lemma 2b;
– if $c_0 \to c_1 \to c_2 \to \cdots$, then the statement follows directly from Lemma 3.

□

Now we present the main theorem.

Theorem 3. *If $c \sim_{st} d$ then for all $\text{CTL}^*_{-\text{X}}$ formulas φ, $c \models \varphi$ iff $d \models \varphi$.*

Proof. The proof is a straightforward adaptation of the one of Theorem 3.2.3 in [18], replacing the calls to Lemma 3.2.2 in that proof by calls to Lemma 5. □

4 Stuttering congruence

First we extend the definition of stuttering bisimilarity to the level of χ processes.

Definition 9. *Two processes p and q are stuttering bisimilar, denoted $p \sim_{st} q$, if for all $\sigma \in \Sigma$, $\langle p, \sigma\rangle \sim_{st} \langle q, \sigma\rangle$.*

To see that stuttering bisimilarity is not a congruence on χ processes, consider the following example.

Example 1. Note that the execution of a send action does not affect the state (see Rule 4 in Table 1), so $a!0 \sim_{st} b!0$. However, $a!0 \parallel a?x \not\sim_{st} b!0 \parallel a?x$, for the process on the left can do a communication action and change the value of x, whereas the process on the right cannot. It follows that $\sim_{st}$ is not a congruence for parallel composition. Also note that $a!0 \sim_{st} \mathit{skip}$, whereas $\partial(a!0) \not\sim_{st} \partial(\mathit{skip})$ ($\partial(a!0)$ is deadlocked; $\partial(\mathit{skip})$ does a τ-transition and terminates successfully). So $\sim_{st}$ is not a congruence for encapsulation either.

The example shows that for an equivalence on χ processes to be a congruence, it should not be completely action insensitive. We adapt the definition of stuttering bisimilarity in such a way that it distinguishes the send and receive actions from the other actions.
Let $\mathcal{A}^{com}$ be the set of all send and receive actions. Let $c \hookrightarrow c'$ mean that there is an $a \in \mathcal{A}\backslash\mathcal{A}^{com}$ such that $c \xrightarrow{a} c'$. Furthermore, let $c \xrightarrow{(a)} c'$ denote $c \xrightarrow{a} c'$ when $a \in \mathcal{A}^{com}$, and $c \hookrightarrow c'$ when $a \in \mathcal{A}\backslash\mathcal{A}^{com}$.

Definition 10. *A symmetric relation $R \subseteq C \times C$ is an* interaction sensitive stuttering bisimulation *iff, for all $(c, d) \in R$, c and d have the same state and:*

1. *if $c\downarrow$, then there exist $d_0, \dots, d_n \in C$ such that*

$$d_0 = d,\ d_0 \hookrightarrow \cdots \hookrightarrow d_n,\ d_n\downarrow \text{ and } cRd_i \text{ for all } i \le n,$$

2. *if $c \xrightarrow{a} c'$, then there exist $d_0, \dots, d_n \in C$ such that*

$$d_0 = d,\ d_0 \hookrightarrow \cdots \hookrightarrow d_{n-1} \xrightarrow{(a)} d_n,\ cRd_i \text{ for all } i \le n-1, \text{ and } c'Rd_n$$

   (we allow n to be 0 only if $a \in \mathcal{A} \backslash \mathcal{A}^{com}$ and c and c' have the same state),

3. *if there exists an infinite sequence $c_0, c_1, c_2, \ldots \in C$ such that*

$$c_0 = c,\ c_0 \hookrightarrow c_1 \hookrightarrow c_2 \hookrightarrow \cdots \text{ and } c_iRd \text{ for all } i \ge 0,$$

   then there exist $d' \in C$ and $j > 0$ such that $d \hookrightarrow d'$ and c_jRd'.

Two configurations c and d are interaction sensitive stuttering bisimilar*, denoted $c \sim_{isst} d$, if there exists an interaction sensitive stuttering bisimulation R such that cRd.*

Theorem 4. *Interaction sensitive stuttering bisimilarity is an equivalence.*

Proof. Reflexivity and symmetry are proved as before. For the transitivity proof, it can be easily seen that Lemmas 1, 2a, and 3 hold when $\rightarrow$ is replaced by $\hookrightarrow$. Then, the proof goes similarly as for Theorem 1. □

To show that all the previous results hold, we prove the following theorem.

Theorem 5. *Interaction sensitive stuttering bisimilarity is a stuttering bisimulation.*

Proof. Suppose that $c \sim_{isst} d$. To show that the termination and transfer condition hold is trivial since $c \hookrightarrow c'$ implies $c \rightarrow c'$ for all $c, c' \in C$. We verify the divergence condition in detail.
Suppose $c_0 \rightarrow c_1 \rightarrow c_2 \rightarrow \cdots$ and $c_i \sim_{isst} d$ for all $i \ge 0$.
If every arrow in this sequence is a '$\hookrightarrow$' arrow, we have that there exist d' and $i > 0$ such that $d \hookrightarrow d'$ and $c_i \sim_{isst} d'$. In this case we are done because $d \hookrightarrow d'$ implies $d \rightarrow d'$.
Suppose now $c_0 \hookrightarrow \cdots \hookrightarrow c_n \xrightarrow{a} c_{n+1}$ and $a \in \mathcal{A}^{com}$.
Since $c_n \sim_{isst} d$, there exist $d_0, \ldots, d_m \in C$ such that $d_0 = d$,

$$d_0 \hookrightarrow \cdots \hookrightarrow d_{m-1} \xrightarrow{a} d_m, c_n \sim_{isst} d_i \text{ for all } i \le m-1 \text{ and } c_{n+1} \sim_{isst} d_m.$$

Because $a \in \mathcal{A}^{com}$, $m > 0$. Then, $c_n \sim_{isst} d_1$ or $c_{n+1} \sim_{isst} d_1$. If $c_n \sim_{isst} d_1$ and $n = 0$ then, since $c_0 \sim_{isst} d_0$, $c_1 \sim_{isst} d_0$ and $\sim_{isst}$ is symmetric and transitive, we conclude that $c_1 \sim_{isst} d_1$. □

The following example shows that interaction sensitive stuttering bisimulation is still not a congruence.

Example 2. Note that *skip* $\sim_{isst}$ *skip* and $\delta \sim_{isst}$ *skip* ; δ. However, *skip* $[]$ $\delta \not\sim_{isst}$ *skip* $[]$ *skip* ; δ, for the right-hand side process can execute *skip* and then deadlock, while the left-hand side process never deadlocks. So, interaction sensitive stuttering bisimulation is not a congruence for alternative composition.

Both the problem illustrated in the above example, and its solution are well known; we need to add a root condition.

Definition 11. *Two configurations c and d are* rooted interaction sensitive stuttering bisimilar *(notation: $c \sim_{riss} d$) iff*

1. $c\!\downarrow$ *iff* $d\!\downarrow$
2. *if* $c \xrightarrow{a} c'$*, then there exists* $d' \in C$ *such that* $d \xrightarrow{(a)} d'$ *and* $c' \sim_{isst} d'$*,*
3. *if* $d \xrightarrow{a} d'$*, then there exists* $c' \in C$ *such that* $c \xrightarrow{(a)} c'$ *and* $c' \sim_{isst} d'$*,*

It is clear that $c \sim_{riss} d$ implies $c \sim_{isst} d$. Also, that rooted interaction sensitive stuttering bisimilarity is an equivalence relation is easily proved. The notion induces a congruence on χ processes.

Definition 12. *Two processes p and q are* stuttering congruent*, denoted $p \cong_{st} q$, if $\langle p, \sigma \rangle \sim_{riss} \langle q, \sigma \rangle$ for all $\sigma \in \Sigma$.*

Clearly, $\cong_{st}$ is an equivalence relation and $p \cong_{st} q$ implies $p \sim_{st} q$. We show that it is a congruence for the constructs of χ.

Theorem 6. *For all $p, q, \overline{p}, \overline{q} \in P$, if $p \cong_{st} q$ and $\overline{p} \cong_{st} \overline{q}$, then:*

1. $b :\rightarrow p \cong_{st} b :\rightarrow q$,
2. $p\ ;\ \overline{p} \cong_{st} q\ ;\ \overline{q}$,
3. $p\ []\ \overline{p} \cong_{st} q\ []\ \overline{q}$,
4. $p^* \cong_{st} q^*$,
5. $p \parallel \overline{p} \cong_{st} q \parallel \overline{q}$,
6. $[\![s \mid p]\!] \cong_{st} [\![s \mid q]\!]$ *for all states s,*
7. $\partial(p) \cong_{st} \partial(q)$.

5 Application

In [20], the translation from χ to PROMELA is discussed in detail. It is pointed out that the translation is straightforward for some constructs of χ (e.g., for assignments and alternative composition), since they also exist in PROMELA. However, the translation of guards, nested scopes and nested parallelism is less straightforward, since they have no direct equivalents in PROMELA. In the preprocessing phase of the proposed translation, nested scopes and certain occurrences of nested parallelism are eliminated, and guards are pushed down to the level of atomic processes. In this section we indicate how this preprocessing phase can be proved correct using the notion of stuttering congruence.

To be able to state some of the results pertaining to the scope operator, we first need to define the notion of *free* occurrence of a variable x in a process p. An occurrence of a variable x in a process p is called *free* if there is no subprocess of p of the form $[\![s \mid q]\!]$ with $x \in \mathrm{dom}(s)$ and q containing the occurrence of x. We denote by *free*(p) the set of all variables with a free occurrence in p.
The following theorem explains how scope operators can be pulled out.

Theorem 7. *Let* $p, q \in P$, *let* $s, s_1, s_2 \in \Sigma$ *and let* $b \in B$ *such that* $\mathrm{dom}(b) \cap \mathrm{dom}(s) = \mathit{free}(p) \cap \mathrm{dom}(s) = \mathit{free}(q) \cap \mathrm{dom}(s) = \emptyset$. *Then:*

1. $b :\rightarrow [\![s \mid p]\!] \cong_{st} [\![s \mid b :\rightarrow p]\!]$,
2. $[\![s \mid p]\!] \,;\, q \cong_{st} [\![s \mid p \,;\, q]\!]$,
3. $p \,;\, [\![s \mid q]\!] \cong_{st} [\![s \mid p \,;\, q]\!]$,
4. $[\![s \mid p]\!] \;[\!]\; q \cong_{st} [\![s \mid p \;[\!]\; q]\!]$,
5. $[\![s \mid p]\!] \parallel q \cong_{st} [\![s \mid p \parallel q]\!]$,
6. $[\![s_1 \mid [\![s_2 \mid p]\!]]\!] \cong_{st} [\![\gamma(s_1, s_2) \mid p]\!]$,
7. $[\![s \mid \partial(p)]\!] \cong_{st} \partial([\![s \mid p]\!])$.

In case of a repetition, we need to be careful: $[\![s \mid p]\!]^*$ is not stuttering congruent with $[\![s \mid p^*]\!]$, for in $[\![s \mid p]\!]^*$ the state s is applied before each repetition of p, while in $[\![s \mid p^*]\!]$ it is only applied before the first repetition. The solution is to incorporate in $[\![s \mid p^*]\!]$ the effect of the state s after each repetition of p by a sequential composition of assignments of the form $x := s(x)$, one for every $x \in \mathrm{dom}(s)$. That is, if $\mathrm{dom}(s) = \{x_1, \ldots, x_n\}$, then we propose to replace $[\![s \mid p]\!]^*$ by $[\![s \mid (p \,;\, x_1 := s(x_1) \,;\, \ldots \,;\, x_n := s(x_n))^*]\!]$. Note, however, that this only works if p does not have the option to terminate immediately (i.e., if $\langle p, \sigma\rangle \not\downarrow$ for all states σ). For, if $\langle p, \gamma(s, \sigma)\rangle\downarrow$, then $\langle [\![s \mid (p \,;\, x := c)^*]\!], \sigma\rangle$ can do the action $aa(x, c)$, whereas $\langle [\![s \mid p]\!]^*, \sigma\rangle$ cannot do the same action (unless p can do it), and thus the root condition is violated.
Let $\overline{P}$ be the set of processes that do not contain ε and in which every occurrence of the star operator is immediately followed by the sequential composition operator. Then $\langle p, \sigma\rangle \not\downarrow$ for all $\sigma \in \Sigma$ and all $p \in \overline{P}$, as is easily proved by structural induction on p.

Theorem 8. *Let* $p \in \overline{P}$, *let* $x \in V$, *and let* $c \in C$. *If* s *is a state such that* $\mathrm{dom}(s) = \{x_1, \ldots, x_n\}$, *then*

$$[\![s \mid p]\!]^* \cong_{st} [\![s \mid (p \,;\, x_1 := s(x_1) \,;\, \cdots \,;\, x_n := s(x_n))^*]\!].$$

The next theorem explains how guards can be distributed over all operators except parallel composition.

Theorem 9. *Let* $p, q \in P$, *let* $s \in \Sigma$ *and let* $b, b_1, b_2 \in B$, *then:*

1. $\mathit{true} :\rightarrow p \cong_{st} p$
2. $b_1 :\rightarrow b_2 :\rightarrow p \cong_{st} (b_1 \wedge b_2) :\rightarrow p$
3. $b :\rightarrow (p \,;\, q) \cong_{st} (b :\rightarrow p) \,;\, q$
4. $b :\rightarrow (p \;[\!]\; q) \cong_{st} (b :\rightarrow p) \;[\!]\; (b :\rightarrow q)$

5. $b :\rightarrow p^* \cong_{st} (b :\rightarrow p) \,;\, p^* \,[]\, b :\rightarrow \varepsilon$
6. $b :\rightarrow \partial(p) \cong_{st} \partial(b :\rightarrow p)$

Note that the process $b :\rightarrow (p \parallel q)$ is not stuttering congruent with the process $(b :\rightarrow p) \parallel (b :\rightarrow q)$. For suppose the processes are executed in a state in which b is *true* and an action from p changes the state in such a way that the value of b becomes *false*. Then the process $b :\rightarrow (p \parallel q)$ proceeds as $p' \parallel q$ and the process $(b :\rightarrow p) \parallel (b :\rightarrow q)$ as $p' \parallel (b :\rightarrow q)$, and in the latter process the option q is blocked.
We finish this section with a theorem that allows us to simplify nested parallelism in the context of sequential composition. For similar reasons as before it is formulated in terms of the set $\overline{P}$.

Theorem 10. *Let $p, q, r \in \overline{P}$, and let $w \in V$ such that $w \notin free(p) \cup free(q) \cup free(r)$. Then,*

1. $(p \parallel q) \,;\, r \cong_{st} [\![w \mapsto 0 \mid p \,;\, w := w + 1 \parallel q \,;\, w := w + 1 \parallel (w = 2) :\rightarrow r]\!]$,
2. $p \,;\, (q \parallel r) \cong_{st} [\![w \mapsto 0 \mid p \,;\, w := w + 1 \parallel (w = 1) :\rightarrow q \parallel (w = 1) :\rightarrow r]\!]$.

6 Conclusion

In this paper we have proposed the notion of stuttering congruence for the modeling and simulation language χ. We have proved that if two χ processes are stuttering congruent, then, with respect to any state σ, they satisfy the same $\text{CTL}^*_{-\text{X}}$ formulas and either both have a deadlock or neither of them. Stuttering congruence is a behavioral congruence for the constructs of discrete-event, untimed part of χ, i.e., it is defined directly on the operational semantics of the language and it is a congruence for its constructions. Therefore, it is suitable for establishing the correctness of syntactic transformations on χ models.
We have illustrated the use of stuttering congruence in correctness proofs of syntactic transformations by indicating how (a part of) the preprocessing phase of the translation from χ to PROMELA can be proved correct. It is explained in detail in [20] that if a χ model satisfies a few general restrictions, then the preprocessing phase yields a χ model that can be straightforwardly translated into PROMELA. The resulting PROMELA specification can then be verified with SPIN. Incidentally, note that for Theorems 8 and 10 it is essential that stuttering congruence allows 'stuttering'; the transformations in these theorems do not preserve the validity of full CTL^*.
Currently, a translation from χ to UPPAAL is being developed, and it also involves a preprocessing phase. We think that stuttering congruence will be suitable for proving the correctness of that preprocessing phase too.
As future work, we mention the extension of the results obtained in this paper to full timed χ [19], proving the preservation of the validity of a timed variant of $\text{CTL}^*_{-\text{X}}$.

Acknowledgements The authors would like to thank Jos Baeten for commenting on a draft of this paper, and the members of the TIPSy project for discussions.

References

1. S. Blom, W. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser, and J.C. van de Pol. μCRL: A toolset for analysing algebraic specifications. In *Proceedings of CAV2001*, LNCS 2102, pages 250–254, 2001.
2. E. Bortnik, N. Trčka, A. J. Wijs, S. P. Luttik, J. M. van de Mortel-Fronczak, J. C. M. Baeten, W. J. Fokkink, and J. E. Rooda. Analyzing a χ model of a turntable system using SPIN, CADP and UPPAAL. *Journal of Logic and Algebraic Programming*, 2005. To appear.
3. V. Bos and J. J. T. Kleijn. Automatic verification of a manufacturing system. *Robotics and Computer Integrated Manufacturing*, 17:185–198, 2001.
4. V. Bos and J.J.T. Klein. *Formal specification and analysis of industrial systems*. PhD thesis, Eindhoven University of Technology, 2002.
5. M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theor. Comput. Sci.*, 59:115–131, 1988.
6. E.J.J. van Campen. *Design of a multi-process multi-product wafer fab*. PhD thesis, Eindhoven University of Technology, 2000.
7. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. MIT Press, Cambridge, Massachusetts, 1999.
8. D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Eindhoven University of Technology, 1996.
9. J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP - a protocol validation and verification toolbox. In *Proceedings 8th of CAV'96*, LNCS 1102, pages 437–440, 1996.
10. J.J.H. Fey. *Design of a fruit juice blending and packaging plant*. PhD thesis, Eindhoven University of Technology, 2000.
11. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2):132–152, 1999.
12. J. A. Govaarts. Efficiency in a lean assembly line: a case study at NedCar born. Master Thesis, 1997.
13. J. F. Groote and F. W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M. S. Paterson, editor, *Proceedings of 17th ICALP*, LNCS 443, pages 626–638, 1990.
14. G.J. Holzmann. *The SPIN model checker*. Addison-Wesley, 2003.
15. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
16. B. Luttik and N. Trčka. Stuttering congruence for χ. Computer Science Report 05/13, Eindhoven University of Technology, 2005.
17. K. S. Namjoshi. A simple characterization of stuttering bisimulation. In *Proceedings of 17th FST & TCS*, LNCS 1346, pages 284–296, 1997.
18. R. De Nicola and F. W. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995.
19. R.R.H. Schiffelers, D.A. van Beek, K.L. Man, M.A. Reniers, and J.E. Rooda. Syntax and semantics of timed Chi. Computer Science Report 05/09, Eindhoven University of Technology, 2005.
20. N. Trčka. Verifying χ models of industrial systems with Spin. Computer Science Report 05/12, Eindhoven University of Technology, 2005.
21. D. A. van Beek, A. van der Ham, and J. E. Rooda. Modelling and control of process industry batch production systems. In *Proceedings of 15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, 2002.