Compositional state space reduction using *untangled* actions

Xu Wang   Marta Kwiatkowska[1]

School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, UK

Abstract

We propose a compositional technique for efficient verification of networks of parallel processes. It is based on an automatic analysis of LTSs of individual processes (using a failure-based equivalence which preserves divergences) that determines their sets of "conflict-free" actions, called untangled actions. Untangled actions are compositional, i.e. synchronisation on untangled actions will not destroy their "conflict-freedom". For networks of processes, using global untangled actions derived from local ones, efficient reduction algorithms have been devised for systems with a large number of small processes running in parallel.

Keywords: Untangled action, Conflict-freedom, Partial order reduction, Process algebra, Compositionality, Determinism, and Partial confluence.

1 Introduction

Informally, an untangled action[2] is a special action in a discrete event system of causality and conflict [23]. At any state of the system the action, if enabled, shall not be entangled through any conflict with the rest of the system, and its only contribution to the system dynamics is by causality. Therefore, if an untangled action is not observed (due to hiding or other operations), its occurrence becomes time irrelevant[3]. This gives us the opportunity to reduce the search space by considering only one possibility of its occurrence time.

The notion of untangled actions is closely related to similar ideas in true concurrency semantics [23], partial order reduction [12,21], and Petri net unfolding [10]. Since our work will be developed in the framework of process algebra (in contrast to state-based formalisms or Petri nets) and concentrates on state space reduction, the closest work to ours is that of τ-confluence reduction by Groote, van de Pol and Blom [8,7,2,3].

[1] {X.Wang,M.Z.Kwiatkowska}@cs.bham.ac.uk

[2] We prefer to use here the term "action" instead of "event" so as to distinguish between actions and their occurrences. But in the rest of the paper they may be used interchangeably.

[3] Some type of progress/maximality assumption is needed to guarantee that the action will eventually occur.

2 Motivations

This work is motivated by our experience in using process algebra (e.g. CSP and FDR2 [6]) to verify asynchronous circuits [22], where high concurrency in gate-level circuits induces serious state explosion problems. A well-known example is the tree arbiter [5]. A tree arbiter consists of a tree of arbiter cells. Each arbiter cell behaves as a two-way arbiter for its sons while at the same time acting as one of the clients of its father node. In this way, a tree arbiter implements multi-way arbitration through a hierarchy of two-way arbitration cells.

The state space of tree arbiters blows up exponentially with the increase of tree size, and it is not readily amenable to reduction due to the conflicts inherent to arbitration. Previously, Petri net unfolding techniques [10] and partial-order reduction enhanced BDD methods [1] had been applied to it, with limited success. In this paper, we will propose untangled actions as a viable solution to verify this and similar systems.

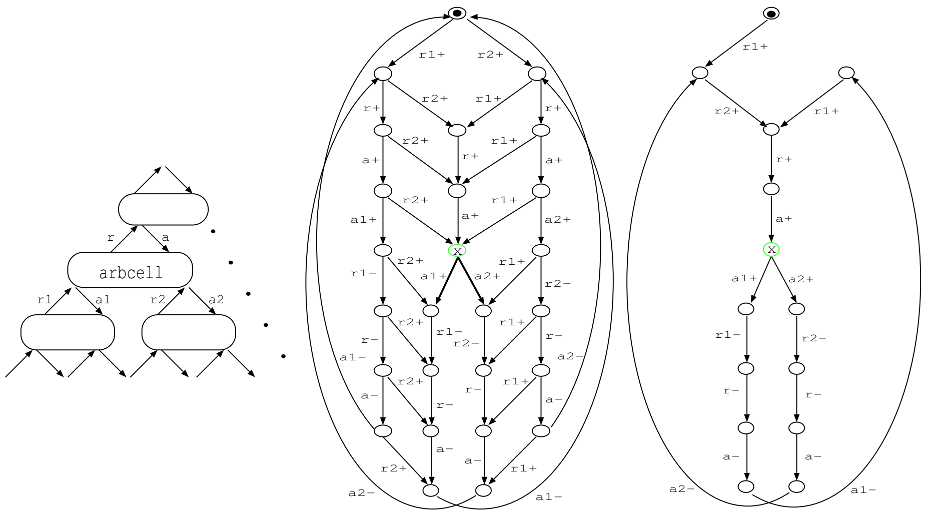

Figure 1. An arbiter tree, and the original and reduced LTSs of an arbiter cell

Untangledness is a simple idea. We will defer the theoretical justification to later sections. For the arbiter cell example, it is not difficult to see that only two actions are entangled in conflicts, i.e. $a1+$ and $a2+$. These tangled actions coincide with so called *output choice* signal transitions [4]. With this information, it is straightforward to give a state-space reduction algorithm by prioritising untangled actions (similar to the *chase* reduction in FDR2) in the exploration of the state space. That is, in a depth-first search, given a state with a non-empty set of untangled outgoing transitions, we use some *strategy* to pick and prioritise one from the set to explore; all the other transitions from the same state, untangled or tangled, will be completely ignored in the exploration. In case a state has no untangled transition, all the transitions from that state will need to be explored. It is not difficult to apply the algorithm to reduce the LTS of the arbiter cell manually. Figure 1 gives the reduced state space based on one possible prioritisation strategy.

However, the above reduction is correct only if we treat the arbiter cell as a closed system and all the untangled actions are not observable to the checked properties.

Synchronisation with the environment may introduce new conflicts that can destroy the untangledness of the actions. The previous works on τ-confluence solve these problems by considering only τ action [7,2,3] or *locally visible and globally invisible* (lvgi) actions[4] without synchronisation [11].

Secondly, untangledness analysis with the environment factors taken into account is difficult, since the analysis must avoid explicitly working on the global state space, which is intractable in our context. In τ-confluence reduction, the proposed solution is to use theorem proving on a symbolic representation (so called *linear processes*) of global state spaces [3], or to use compositionality [11] as we have adopted. However, since the involved actions must be synchronisation-free, their compositionality does not apply to the tree arbiter, whose lvgi actions, e.g. r, a, $r1$, etc., need further synchronisation.

In this paper, we propose a compositional technique for concurrent systems such that untangledness analysis is done at a local level. A compositionality theorem automatically calculates global untangledness information from the local information. Using thus obtained results, state space reduction can be applied on-the-fly on the global systems.

Structure of the paper. After the introduction of basic notations (Section 3) and concurrent systems (Section 4), two important (partial) determinacy notions on LTSs with lvgi actions, one stronger than the other, are proposed in Section 5. The former is compositional on the lvgi actions without synchronisation potential and induces a simple and efficient on-the-fly reduction procedure (Section 6). The latter removes the synchronisation restriction and becomes compositional on all lvgi actions, and thus enables compositional reductions (Section 7). Preliminary experiment results are given and the paper is summarised in Section 8.

3 Basic Notation

A *LTS* (*Labelled Transition System*) is a tuple (A, S, T, s^0), consisting of a (finite or countably infinite) set A of visible events called the alphabet, a (finite or infinite) set S of states, a transition relation $T : S \times (A \cup \{\tau\}) \leftrightarrow S$ and the initial state $s^0 \in S$.

(**Event, Sequence and Trace**) Let e be a visible event, a be a τ or visible event, and Δ be a subset of A. Δ^τ and A^τ denote $\Delta \cup \{\tau\}$ and $A \cup \{\tau\}$. k and l are finite sequences of events (including the empty sequence, ϵ)[5]. t and u are finite traces, i.e. finite sequences of non-τ events. $\overline{k}$, $\overline{l}$, $\overline{t}$ and $\overline{u}$ are the infinite variants, while $\tilde{k}$, $\tilde{l}$, $\tilde{t}$ and $\tilde{u}$ can denote both finite and infinite ones.

(**Sequence operations**) Juxtaposition is used for sequence concatenation, e.g. $k\tilde{l}$. $|\tilde{k}|$ gives us the length of the sequence $\tilde{k}$ (ω for infinity). *head* and *tail* (unary prefix operator) are defined as normal. $-$ is a binary infix operator removing from a sequence left to right all the members in another sequence according to multiplicity, e.g. $e_1e_2e_2e_3e_1e_2 - e_3e_2e_2 = e_1e_1e_2$ and $e_1e_2 - e_2e_3 = e_1$.

[4] Formally, given a network of processes, a lvgi action is one that is visible on an individual process but is eventually hidden during process composition and thus invisible at the global network level.

[5] Sometimes, a and e are also used as singleton sequences or traces.

(**Prefix order, Projection and Containment**) $\leq$ is the prefix order on (finite or infinite) sequences. *pref*() calculates the set of (finite) prefixes of a finite or infinite sequence. $\tilde{k} \upharpoonright_\Delta$ removes from $\tilde{k}$ all the events not in Δ. Both can be lifted to operate on sets of sequences. We say $\tilde{l}$ *contains* $\tilde{k}$ iff $\forall\, a \in A^\tau \bullet \tilde{k} \upharpoonright_{\{a\}} \leq \tilde{l} \upharpoonright_{\{a\}}$; $\tilde{l}$ *trace-contains* $\tilde{k}$ iff $\tilde{l} \upharpoonright_A$ contains $\tilde{k} \upharpoonright_A$. Moreover, we define the following notation:

Definition 3.1 [Path and Arrow notation] Given a LTS, (A, S, T, s^0):

- a finite path is a finite sequence of alternating states and events, $s_0\, a_1\, s_1\, a_2 ... a_n\, s_n$ ($\in$ *PATH* $\mathrel{\widehat{=}} S \times (A^\tau \times S)^*$), where $(s_{i-1}, a_i, s_i) \in T$ for all $1 \leq i \leq n$. The labelling sequence of the path is $a_1\, a_2 ... a_n$ (i.e. ϵ when $n = 0$). Similarly, $s_0\, a_1\, s_1\, a_2 ...$ ($\in \overline{PATH} \mathrel{\widehat{=}} S \times (A^\tau \times S)^\omega$) is an infinite path and $a_1\, a_2 ...$ is its labelling sequence.
- $s \xrightarrow{k} s'$ iff there is a k-labelled finite path going from s to s'.
- $s \xrightarrow{\overline{k}}$ iff there is a $\overline{k}$-labelled infinite path starting from s.
- k is *enabled* at s, i.e. $s \xrightarrow{k}$, iff there exists a state s' such that $s \xrightarrow{k} s'$.
- $\xrightarrow{a} s'$ iff there exists a reachable state s in the LTS such that $s \xrightarrow{a} s'$, and we say s' is *caused* by a.
- $s \xRightarrow{t} s'$ iff $s \xrightarrow{k} s'$ and $k \upharpoonright_A = t$; $s \xRightarrow{\overline{t}}$ iff $s \xrightarrow{\overline{k}}$ and $\overline{k} \upharpoonright_A = \overline{t}$; $s \xRightarrow{t}$ iff there exists a state s' such that $s \xRightarrow{t} s'$.

A state s is *deadlocked*, i.e. *deadlock*(s), iff s does not have any outgoing transition. A state s is *divergent*, i.e. *divergent*(s), iff there is an infinite τ-path in the LTS that starts from s.

Definition 3.2 [Traces] Given a LTS, the set of finite traces FT is $\{t \mid s^0 \xRightarrow{t}\}$, the set of infinite traces IT is $\{\overline{t} \mid s^0 \xRightarrow{\overline{t}}\}$, the set of deadlock traces LT is $\{t \mid \exists\, s \bullet deadlock(s) \wedge s^0 \xRightarrow{t} s\}$, and the set of divergence traces DT is $\{t \mid \exists\, s \bullet divergent(s) \wedge s^0 \xRightarrow{t} s\}$.

Definition 3.3 [Normalisation] A LTS, LTS^N, is *normalised* iff all τ-transitions are self-loops, i.e. $s \xrightarrow{\tau} s' \Rightarrow s = s'$, and there is no ambiguous transition, i.e. $s \xrightarrow{e} s' \wedge s \xrightarrow{e} s'' \Rightarrow s' = s''$.

4 Concurrent Systems

Basic processes given as LTSs can be combined using the parallel and the hiding operators to form a concurrent system [15].

$$SCS ::= LTS \mid SCS \parallel SCS' \mid SCS \setminus \Delta \mid SCS[R^{1-1}]$$

Definition 4.1 [Parallel] Given LTS_1 and LTS_2, $LTS_1 \parallel LTS_2$ gives another LTS, (A, S, T, s^0), where $A = A_1 \cup A_2$, $S = S_1 \times S_2$, $s^0 = (s_1^0, s_2^0)$ and T is the least relation satisfying the following rules:

$$\frac{s_1 \xrightarrow{a} s_1' \quad a \notin A_2}{(s_1, s_2) \xrightarrow{a} (s_1', s_2)} \qquad \frac{s_2 \xrightarrow{a} s_2' \quad a \notin A_1}{(s_1, s_2) \xrightarrow{a} (s_1, s_2')} \qquad \frac{s_1 \xrightarrow{e} s_1' \quad s_2 \xrightarrow{e} s_2'}{(s_1, s_2) \xrightarrow{e} (s_1', s_2')}$$

Definition 4.2 [Hiding] Given LTS, $LTS \setminus \Delta$ gives a new LTS, (A', S, T', s^0), where $A' = A \setminus \Delta$, and $T' = T'[\tau/\Delta]$, i.e. substituting τ for every Δ event on every occurrence in T.

Definition 4.3 [Renaming] Given LTS, $LTS[R]$, where $R : A \leftrightarrow A'$ and $\operatorname{dom} R \cap \operatorname{ran} R = \{\}$, gives a new LTS, (A', S, T', s^0), where $A' = (A \setminus \operatorname{dom} R) \cup \operatorname{ran} R$, and $T' = \{(s, a, s') \mid (a \notin \operatorname{dom} R \wedge (s, a, s') \in T) \vee (\exists\, e \bullet (e, a) \in R \wedge (s, e, s') \in T)\}$.

The definition allows m-to-n renaming. Usually only special cases are needed: 1-to-1 (R^{1-1}), 1-to-m (R^{1-m}) and m-to-1 (R^{m-1}).

4.1 Semantics

In classic CSP [15], stable failures and failure/divergences are the major semantic models used in CSP. They are both finite trace models. However, there is a newly developed infinite trace CSP model [16], the $\mathcal{SBD}$ model, which preserves all the divergence traces in CSP processes.

Given a LTS, (A, S, T, s^0), a state s is *stable*, i.e. $\mathit{stable}(s)$, iff $\neg(s \xrightarrow{\tau})$. Sometimes, we also use $\mathit{stable}(s, \Delta)$ to mean $\forall\, a \in \Delta \bullet \neg(s \xrightarrow{a})$. Given a set of finite sequences, $\mathit{lmt}()$ outputs a set of infinite sequences, each being the limit of a chain of increasing (i.e. prefix order) finite sequences belonging to the set. Define $\mathit{IB} \mathrel{\widehat{=}} \mathit{IT} \cup \mathit{lmt}(\mathit{DT})$.

(**Stable failures and Behaviours**) The set of stable traces ST is $\{t \mid \exists\, s \bullet \mathit{stable}(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s\}$. The set of stable failures SF is $\{(t, \Delta) \mid \exists\, s \bullet \mathit{stable}(s) \wedge s^0 \stackrel{t}{\Longrightarrow} s \wedge \forall\, e \in \Delta \bullet \neg\,(s \xrightarrow{e})\}$. The type of *behaviours* is $\mathit{BEHV}(A) \mathrel{\widehat{=}} A^* \cup (A^* \cdot \{\tau^\omega\}) \cup A^\omega$. The set of behaviours BH is $\{\tilde{k} \mid \tilde{k} \in \mathit{FT} \vee \tilde{k} \in \mathit{IT} \vee (\tilde{k} = t\tau^\omega \wedge t \in \mathit{DT})\}$.

Definition 4.4 [SBD equivalences] $\mathit{LTS} \stackrel{\mathit{SBD}}{=} \mathit{LTS}'$ iff $\mathit{SBD}(\mathit{LTS}) = \mathit{SBD}(\mathit{LTS}')$, and $\mathit{LTS} \stackrel{\mathit{SBDF}}{=} \mathit{LTS}'$ iff $\mathit{SBDF}(\mathit{LTS}) = \mathit{SBDF}(\mathit{LTS}')$, where $\mathit{SBD}(\mathit{LTS}) = (\mathit{FT}, \mathit{DT}, \mathit{IB})$ and $\mathit{SBDF}(\mathit{LTS}) = (\mathit{SF}, \mathit{DT}, \mathit{IB})$.

Theorem 4.5 (Weakest congruence [20,13]) *W.r.t. the parallel, hiding and renaming operators defined above*[6]*,* $\stackrel{\mathit{SBDF}}{=}$ *is the weakest congruence preserving* LT *and* DT *information on LTSs.*

Definition 4.6 [U-determinism] Given a LTS, it is *U-deterministic* (i.e. unstably deterministic[7]) iff $\mathit{ST} \cap \mathit{DT} = \{\}$ and $te \in \mathit{FT} \Rightarrow (t, \{e\}) \notin \mathit{SF}$.

Proposition 4.7 *Given U-deterministic* LTS *and* LTS', $\mathit{LTS} \stackrel{\mathit{SBDF}}{=} \mathit{LTS}'$ *iff* $\mathit{LTS} \stackrel{\mathit{SBD}}{=} \mathit{LTS}'$. *Given U-deterministic* LTS*, there exists a normalised* LTS^N *such that* $\mathit{LTS} \stackrel{\mathit{SBDF}}{=} \mathit{LTS}^N$.

[6] The weakest congruence result can be extended to the other CSP operators [15] since SBDF is a congruence on those CSP operators as well [16].

[7] As suggested by Roscoe [19], U-determinism, in contrast to classic determinism of CSP (which rules out divergence), does not quite coincide with the operational intuition of determinism. For instance, it does not possess τ-inertness [8]. In this sense, detachability on an empty A_i set (Section 5) is closer to the operational intuition.

Proof. Follows from the U-determinism definition and $FT = ST \cup DT$. Based on $SBD(LTS)$, a normalised LTS can be constructed due to $IB = lmt(FT)$. □

Proposition 4.8 *U-deterministic LTSs are closed under the parallel composition.*

Proof. Normalise these LTSs and use the transition rules of the parallel operator. It is easy to see that the result is normalised, too. □

5 Untangled action analysis

Given a LTS consisting of τ action, lvgi actions and globally visible actions, the most important ingredient of its state space traversal algorithm probably is, at a state with multiple outgoing transitions, how to choose the next branch to pursue. The decision can be split into two parts. One is what we call *visible choices*, which decide the next visible action in the global behaviour. The other is called *invisible choices*; they decide which specific next branch to follow in order to achieve the objective of the visible choice. τ and lvgi transitions, as well as ambiguous transitions, give rise to invisible choices. Usually, visible choices are intertwined with invisible choices. But under certain conditions, they can be separated or *detached* from each other in the sense that they are "independent" from each other. That is, no matter what invisible choice is taken, it will not affect the achievement of the decided visible choice. For the case when ambiguous transitions are not considered, this is τ*-inertness* [8].

This insight leads to the state space reduction algorithms in many process-algebraic frameworks [14,11,2,6]. The algorithm simply makes arbitrary decisions on invisible choices and ignores the other alternatives completely in the state space traversal. It also forms the basis of our reduction algorithm in Section 6. We call such systems *detachable* systems.

5.1 Detachability

A LTS is regarded as an acceptor of behaviours. We assume A_v and A_i are respectively the set of globally visible actions and the set of lvgi actions. When being fed a global behaviour (i.e. $\tilde{k} \in BEHV(A_v)$), the acceptor will control the invisible choices in the LTS and try to accept or reject the behaviour. The different ways an acceptor decides on invisible choices give rise to different acceptor strategies. Thus, a strategy can be regarded as an unfolding of the original LTS followed by a reduction that resolves all the invisible choices in it, e.g. the reduced LTS in Figure 1 is a strategy of the original one. For the same behaviour, the acceptor may have both a strategy to accept it and one to reject it.

Given LTS, formally a strategy, $stg : PATH \times BEHV(A_v) \nrightarrow (A^\tau \times S) \cup \{stop, reject\}$, is a minimal (subset order) partial function satisfying the rules:

(i) $\{s^0\} \times BEHV(A_v) \subseteq \operatorname{dom} stg$

(ii) $(s_0 a_1 ... s_n, e\tilde{k}) \in \operatorname{dom} stg \Rightarrow stg(s_0 a_1 ... s_n, e\tilde{k}) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau \cup \{e\}\} \cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

(iii) $(s_0 a_1 ... s_n, \epsilon) \in \operatorname{dom} stg \Rightarrow stg(s_0 a_1 ... s_n, \epsilon) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{stop \mid stable(s_n, A_i^\tau)\}$

(iv) $(s_0a_1...s_n, \tau^\omega) \in \operatorname{dom} stg \Rightarrow stg(s_0a_1...s_n, \tau^\omega) \in \{(a,s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{reject \mid stable(s_n, A_i^\tau)\}$

(v) $stg(s_0a_1...s_n, \tilde{k}) = (a,s) \Rightarrow (s_0a_1...s_nas, \tilde{k} - (a \restriction_{A_v})) \in \operatorname{dom} stg$

Initially, the acceptor is ready to be fed with any behaviour (**rule 1**). Once fed, the acceptor starts the execution to consume the sequence step by step (**rule 5**). A state of the execution (i.e. the input to the function) consists of a *history* (a finite path in *LTS* whose labelling sequence, after the projection onto A_v, gives a prefix of the fed behaviour denoting the consumed part) and a *suffix* of the behaviour (denoting the remaining part). Minimality of the function implies that only reachable states are defined on *stg*. Given a reachable state and a *pending* action, i.e. e on top of the current suffix, the acceptor is free to make any invisible choice to transit in *LTS*, e.g. (a,s), so long as the transition is consistent with e (the visible choice), i.e. $a \in A_i^\tau \cup \{e\}$ (**rule 2**). When the execution reaches a state where no more consistent invisible choices can be made, the acceptor will either stop (if the consumption is complete) or reject (if incomplete). Given a behaviour $\tilde{k}$ and a strategy *stg*, the acceptor's execution produces a finite path if it ends with *stop* or *reject*; otherwise the execution produces an infinite path.

Acceptance condition: We say *stg* is an *accepting* strategy for $\tilde{k}$ on *LTS* iff the execution does not end with *reject* and produces a path whose labelling sequence trace-contains $\tilde{k}$. Otherwise, *stg* is a *rejecting* strategy for $\tilde{k}$ on *LTS*.

However, these strategies do not handle divergence correctly. For instance, if the initial state of *LTS* has a τ loop, *LTS* has a simple rejecting strategy (i.e. following the τ loop indefinitely) for any non-trivial A_v behaviours. It is "unfair" for systems like normalised LTSs, where the τ loops are self-loops, causing no state change ("unprogressing loops"). Thus, an extra requirement shall be put on strategies.

Definition 5.1 [Fairness] A strategy *stg* is a *fair* strategy iff its (infinite) execution cannot lead to a state after which, though an action e is pending to be consumed (c.f. **rule 2**), it makes no further A_v transition and an action $a \in A_i^\tau \cup \{e\}$ is always enabled (i.e. on *LTS*) but never taken.

The above definition is a kind of maximality and weak fairness requirement on actions as that in partial order semantics. However, it is not applied on all actions. Only the pending e and the A_i^τ actions will be guaranteed to progress. Progress on e will be able to guide strategies out of unprogressing loops. Progress on the A_i^τ actions can guide strategies out of indefinite delays on any member of A_i^τ. This is necessary for compositionality.

Definition 5.2 [May&Must acceptance] A LTS *may-accept* a behaviour iff there exists an accepting strategy. It *must-accept* a behaviour iff there does not exist a fair rejecting strategy.

It is not difficult to see that the set of may-accepted behaviours is exactly $BH(LTS \setminus A_i)$ and thus implies the *SBD*-equivalence.

Definition 5.3 [Detachability] Given *LTS* and $A_i \cup A_v = A$, A_i is detachable from *LTS* (or *LTS* is detachable on A_i) iff *LTS* may-accept $\tilde{k}$ iff *LTS* must-accept $\tilde{k}$ for all $\tilde{k} \in BEHV(A_v)$.

Detachability has many good properties, some of which will be shown in Section 6, where a reduction algorithm based on detachability will be given. Here we will just mention U-determinism and a restricted form of compositionality.

Proposition 5.4 *Δ is detachable from LTS implies $LTS \setminus \Delta$ is U-deterministic, but not vice versa.*

Proof. The violation of either condition in U-determinism definition implies a fair rejecting strategy for a behaviour in $BH(LTS \setminus \Delta)$. The converse is not true due to the counterexample: $S = (e \rightarrow Div \,\Box\, e' \rightarrow Stop)$ with $\Delta = \{e\}$. $\Box$

It is *crucial*, however, to notice that detachable LTS on Δ does not imply that $LTS \setminus \Delta$ is detachable on $\{\}$. Hiding removes the distinctiveness among the members of Δ; thus less progress requirement is placed on strategies and the fair rejection of behaviours becomes easier. Indeed, hiding shall not be applied on LTSs before the reduction algorithm of Section 6 has used the distinctiveness information.

Detachability is compositional on synchronisation-free lvgi actions (c.f. Theorem 7.1 for its form)[8]. This is in part due to the progress requirement on the A_i^τ actions. For example, $R = e \rightarrow R$ is detachable on $\{e\}$ and $R' = e' \rightarrow e'' \rightarrow Stop$ is detachable on $\{e'\}$ (i.e. even without any progress requirement). But, without the progress requirement on $\{e'\}$, $R \parallel R'$ is not detachable on $\{e, e'\}$.

On the other hand, compositionality does not hold for lvgi actions with synchronisation potential. Informally, it is due to the fact that detachability allows conflicts within A_i actions (an extreme case is "auto-conflict" within one action). It is just that the resolution of these conflicts does not affect the causality dependency with the A_v part which makes these conflicts detachable from those of the A_v part. Once there is synchronisation, conflicts can be propagated amongst processes and create new ones that may not be detachable.

The following two processes give an example:

$$P = e \rightarrow e \rightarrow e' \rightarrow Stop \,\Box\, e \rightarrow e' \rightarrow Stop$$
$$Q = e \rightarrow e' \rightarrow Stop$$

With $A_i = \{e\}$, P and Q are both detachable, although P contains an auto-conflict in the sense that one branch needs two e actions to enable e' while the other needs just one. The parallel composition of the two, however, is not detachable since one branch will lead to the occurrence of e' while the other will not. Therefore, to make compositionality work fully, conflicts must be ruled out completely on A_i actions. This gives us the notion of *untangled actions*.

5.2 Untangledness

With synchronisation on lvgi actions, untangledness shall be sensitive to the type as well as the number of lvgi actions expended to drive causality.

Given LTS^N [9] and $A_i \cup A_v = A$, a strategy, $stg : PATH \times BEHV \nrightarrow (A^\tau \times S) \cup \{stop, reject\}$, is a minimal partial function satisfying:

[8] The theorem and its proof are omitted due to space limitation.

[9] A definition based on unnormalised LTSs is also possible. But it complicates the presentation and the generality is not needed for this paper.

(i) $\{s^0\} \times BEHV \subseteq \operatorname{dom} stg$

(ii) $(s_0 a_1 ... s_n, e\tilde{k}) \in \operatorname{dom} stg \Rightarrow stg(s_0 a_1 ... s_n, e\tilde{k}) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau \cup \{head(e\tilde{k} \restriction_{A_v})\}\} \cup \{reject \mid \neg\, s_n \xrightarrow{e}\}$

(iii) $(s_0 a_1 ... s_n, \epsilon) \in \operatorname{dom} stg \Rightarrow stg(s_0 a_1 ... s_n, \epsilon) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{stop \mid stable(s_n, A_i^\tau)\}$

(iv) $(s_0 a_1 ... s_n, \tau^\omega) \in \operatorname{dom} stg \Rightarrow stg(s_0 a_1 ... s_n, \tau^\omega) \in \{(a, s) \mid s_n \xrightarrow{a} s \wedge a \in A_i^\tau\} \cup \{reject \mid stable(s_n)\}$

(v) $stg(s_0 a_1 ... s_n, \tilde{k}) = (a, s) \Rightarrow (s_0 a_1 ... s_n a s, \tilde{k} - (a \restriction_A)) \in \operatorname{dom} stg$

Like the previous one, the acceptor controls the order and the occurrence of A_i actions. Thus **rule 3** and the parts of **rule 2 and 4** not involving *reject* remain the same. Unlike the previous one, A_i actions become visible in the fed behaviours (**rule 1 and 5**) and the acceptor is more sensitive (the parts of **rule 2 and 4** involving *reject*). For instance, once the right type and number of actions have occurred (i.e. removed from the fed behaviours), a new action will be enabled on top of the current suffix (i.e. the pending e). e cannot be delayed by any other A action; if it is not simultaneously enabled on LTS, it may result in the immediate issue of *reject* (the *reject* part of **rule 2**). It gives the acceptor more freedom in rejecting behaviours (e.g. eee' behaviour of the P process above will incur *reject*). The acceptance conditions remain the same except for the adaptation for A_i visibility.

Acceptance condition: *stg* is an *accepting* strategy for $\tilde{k}$ on LTS iff the execution does not end with *reject* and produces a path whose labelling sequence trace-contains $\tilde{k}$. Otherwise, *stg* is a rejecting strategy for $\tilde{k}$ on LTS.

Similarly, fairness can be simplified since the fed behaviours (with A_i visible) can guide itself now.

Definition 5.5 [Fairness] A strategy *stg* is a *fair* strategy iff its (infinite) execution cannot lead to a state after which the pending action e is always enabled (i.e. on LTS^N) but never taken.

Moreover, if we distinguish infinite rejecting strategies (i.e. infinite executions not trace-containing the fed behaviour) from finite ones (i.e. finite executions ending with *reject*), it is obvious that only finite rejecting strategies are needed.

Proposition 5.6 (Finite rejection) *If there is an infinite fair rejecting strategy for a behaviour, there is also a finite one for it.*

Proof. Issue *reject* at one of the states when the final pending e is not enabled. □

May-acceptance and must-acceptance can be defined like in the previous section. However, $BH(LTS^N)$ is only a subset of the may-accepted behaviours, and the definition of untangledness must change accordingly.

Definition 5.7 [Untangledness] Given LTS^N (and $A_i = \Delta$), Δ is untangled in LTS^N (or LTS^N is untangled on Δ) iff LTS must-accept $\tilde{k}$ for all $\tilde{k} \in BH$.

Given Δ, its untangledness decision problem can be solved by a CSP refinement check using stable failures model in Appendix A. This is due to the finite rejection property. The LHS of the check (i.e. the specification) is a fixed process while the RHS is two copies of LTS^N coordinated by another fixed process. The refinement problem of this form is in NLOGSPACE.

Proposition 5.8 *Untangled action sets are closed under subset inclusion and union.*

Proof. Subset-hood: Any rejecting strategy for a behaviour will remain so with the increase of A_i.

Union: Assume w.l.o.g. A_i, A'_i and A_v partitioning A. If $A_i \cup A'_i$ is tangled, then there is either (t, u) satisfying **c0** or (t', u') satisfying **c0′** (c.f. Proposition A.2 and A.5 in Appendix A). If A_i and A'_i are both untangled, any action in A_i or A'_i can be freely moved forward to or inserted at another position in t or t' where it is also enabled (c.f. Proposition A.3). For instance, if $e = \mathit{head}\ u' \wedge e \in A_i$, $e(t' - e)$ must be in FT (**c1** on A_i); if $e = \mathit{head}\ u' \wedge e \in A_v$, then e is in t' and can be moved step by step to the head (**c2** on A_i and A'_i) resulting in $e(t' - e)$. $e(t' - e) \in FT$ and $e(t' - e) \in DT$ (**c0′**). Thus, step by step t (and t') can be transformed to a form with ue as a prefix (and to $u' \in DT$). Contradiction. □

(**Maximal untangled set**) Given Δ, its maximal subset of untangled actions can be found by doing the CSP check on each singleton subset of Δ and taking the union of the successful ones.

Theorem 5.9 *Untangled Δ in LTS^N is also detachable (not vice versa).*

Proof. Assume detachability is not true and the may-accepted but not must-accepted behaviour is $\tilde{k}$. Then there can be two cases: a finite rejecting strategy or an infinite fair rejecting strategy. $\tilde{k} \in BH(LTS \setminus \Delta)$ implies there is $\tilde{k}' \in BH(LTS^N)$ such that the A_v behaviour implied by $\tilde{k}'$ equals $\tilde{k}$. Both strategies can be used directly as fair rejecting strategies for $\tilde{k}'$ since the untangledness acceptor has more freedom in rejection and the fairness for untangledness is strictly weaker than that for detachability.

The converse is not true due to counterexamples like the process P at the end of Section 5.1. □

Note that untangledness, detachability, U-determinism etc. form a hierarchy of partial determinacy properties (c.f. Figure 2 in Appendix C). An interesting discussion of various determinacy and confluence notions in classic process algebras can be found in [18], where may-testing and must-testing are also used to characterise determinacy. Confluence in our context is the same as the untangledness on A (i.e. the full alphabet).

Untangled actions are compositional; global untangled actions can be calculated from local ones. The compositionality theorem will be given in Section 7, where a new compositional reduction technique enabled by it is also proposed. The new technique feeds the global untangledness information to a specially designed on-the-fly reduction procedure called chase^+, which reduces state spaces by exploiting detachability (c.f. Theorem 5.9).

6 Reduction Algorithm

The new algorithm is an extension of the *chase* function in FDR2 [6], and also shares similarity with the reduction algorithms based on τ-inertness [2,11,14]. The idea is based on the fact that in a detachable system a behaviour is accepted by its LTS iff it is accepted by a fair strategy of the LTS. Thus, the LTS can be reduced by removing all other strategies in it, which results in an equivalent LTS containing just one strategy. The most important ingredient of the reduction algorithm, consequently, is finding a suitable fair strategy.

(**Round robin strategy**) Assume the actions in A_i^τ are arranged in a (directed) cycle with a default starting position, and $next(c, \Delta)$ is a function, which, given the current action c and the set of candidate actions Δ, outputs the candidate following c that is closest in the cycle. (Note that, when $c = \epsilon$, the default starting position is assumed.) A subclass of fair strategies on finite-state LTSs, called *round robin* strategies, use a round robin strategy on the cycle to implement fairness. Formally they are minimal partial functions satisfying the same conditions as in Section 5.1 but with **rule 2** replaced by the following[10]:

$2a'$. $(s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge \neg\, fair\_loop(s_0 a_1 ... a_n s_n) \Rightarrow$
$stg(s_0 a_1 ... a_n s_n, e\tilde{k}) \in$
$\{(a, s) \mid s_n \xrightarrow{a} s \wedge a = next(a_n\restriction_{A_i^\tau}, \{a : A_i^\tau \mid s_n \xrightarrow{a}\})\} \cup$
$\{(e, s) \mid s_n \xrightarrow{e} s \wedge stable(s_n, A_i^\tau)\} \cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

$2b'$. $(s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge fair\_loop(s_0 a_1 ... a_n s_n) \Rightarrow$
$stg(s_0 a_1 ... s_n, e\tilde{k}) \in$
$\{(e, s) \mid s_n \xrightarrow{e} s\} \cup \{(a, s) \mid \neg\, s_n \xrightarrow{e} \wedge\, s_n \xrightarrow{a} s \wedge a \in A_i^\tau\}$
$\cup \{reject \mid \neg\, s_n \xrightarrow{e} \wedge\, stable(s_n, A_i^\tau)\}$

where $fair\_loop(s_0 a_1 ... s_n)$ is true iff the maximal suffix of $s_0 a_1 ... s_n$ that is a A_i^τ-path, say $s_i a_{i+1} ... s_n$, contains a fair A_i^τ-loop but $s_i a_{i+1} ... s_{n-1}$ does not. A fair A_i^τ-loop is a A_i^τ-loop that has gone through at least one round of the cycle.

Intuitively, this means that the strategy will give priority to A_i^τ transitions as long as the A_i^τ transitions on top of the history have not formed a fair A_i^τ loop yet. Once one is formed (and exactly at this moment) the pending e transition will be given priority (to implement the weak fairness on e). Thereafter, A_i^τ transitions continue to have priority.

Proposition 6.1 *Given LTS and A_i, a round robin strategy is a fair strategy.*

Applying a round robin strategy stg on LTS gives a reduced LTS. Similar to [2,3], it can be shown that there exists a "representation mapping", which, for our case, maps an *entry point* to its *exit point*.

Let $\mathcal{MCC} = \{..., S_i, ...\}$ be the equivalence induced by the reflexive, symmetric, and transitive closure of A_i^τ transitions in LTS. Each member S_i is an equivalence class. Define the set of stg entry points on S_i as $ENT(S_i) \mathrel{\widehat{=}} \{s : S_i \mid s = s^0 \vee (stg(s_0 a_1 ... s_n, \tilde{k}) = (e, s) \wedge e \in A_v)\}$, and the set of stg exit points on S_i as $EXT(S_i) \mathrel{\widehat{=}} \{s_n : S_i \mid ((s_0 a_1 ... a_n s_n, e\tilde{k}) \in \mathrm{dom}\, stg \wedge fair\_loop(s_0 a_1 ... a_n s_n)) \vee$

[10] Strictly speaking, this section implicitly assumes normalisation on LTSs. This improves the presentation but is not technically needed.

$stable(s_n, A_i^\tau)\}$. The set of *stg* exit points are exactly those states at which **rule 2b′** is activated or A_i^τ-stability is reached.

Proposition 6.2 *Given detachable A_i and a round robin strategy stg on LTS, if any execution of stg at any time enters $S_i \in \mathcal{MCC}$ with the intention to leave (i.e. having a pending e), then the exit point ($\in EXT(S_i)$) is uniquely determined by its entry point ($\in ENT(S_i)$).*

Proof. *next* is insensitive to how an entry point is entered. So all entries on the same point lead to the same path in *LTS* and thus the same point activated or the same A_i^τ-stable point reached. □

(**Representative function**) Let *ENT* and *EXT* be the union of sets of entry and exit points for all $S_i \in \mathcal{MCC}$. Therefore, there exists a representative function, $exit : ENT \to EXT$, mapping each entry point to its exit point. An exit point can fully represent all its entry points. If the exit point is A_i^τ-stable, then the set of outgoing transitions on the representative is exactly the set of outgoing transitions on the point. Otherwise, the set of outgoing transitions on the representative is exactly the set of A_v outgoing transitions combined with a τ self-loop.

Definition 6.3 [Reduction function] Given detachable A_i from *LTS*, function $chase^+(LTS, A_i)$ outputs another LTS, $(A_v, EXT, T', exit(s^0))$, where $T' = \{(s, e, s') : EXT \times A_v \times EXT \mid \exists\, s_i : S \bullet s \overset{e}{\longrightarrow} s_i \wedge s' = exit(s_i)\} \cup \{(s, \tau, s) \mid s \in EXT \wedge \neg\, stable(s, A_i^\tau)\}$.

Therefore, we can adopt a scheme similar to that in [2] to implement $chase^+$ as an on-the-fly procedure integrated in refinement or model checking. Note also that round robin strategies are *local* strategies. That is, the definition only depends on the pending action and the top elements of the history and the exit points can be calculated by simply following the strategy. Therefore, the *exit* function need not be explicitly constructed. It enables a simpler and more efficient implementation of the $chase^+$ reduction procedure.

Theorem 6.4 (Preservation) *Δ is detachable from finite-state LTS implies $chase^+(LTS, \Delta)$ is normalised and $chase^+(LTS, \Delta) \overset{SBDF}{=} LTS \setminus \Delta$.*

Proof. Normalisation follows from the definition of $chase^+$.

Preservation: $chase^+(LTS, \Delta)$ contains a single strategy (due to normalisation). The strategy is exactly a speed up (i.e. removing intermediate τ-chains) of the original round robin strategy (except for the execution states when there is no pending e, i.e. deadlock or divergence, which can be safely ignored due to detachability). Therefore, $chase^+(LTS, \Delta)$ and *LTS* have the same set of may-accepted behaviours, implying $chase^+(LTS, \Delta) \overset{SBD}{=} LTS \setminus \Delta$. Since both are U-deterministic, they are *SBDF*-equivalent. □

7 Compositional reduction

For the reduction technique of this paper to work effectively, it is preferable to represent all processes (including U-nondeterministic ones) in the form of $LTS^N \setminus \Delta$ rather than directly as unnormalised LTSs. Our philosophy is that, if one is inquisitive enough on details, all the unaccounted-for choices in the LTS, i.e. those due to τ-transitions or ambiguous transitions, can be accounted for by introducing some extra lvgi actions. This will not result in any loss of expressiveness, e.g. w.r.t. *SBDF* models. Moreover, these lvgi choices need to remain so during the verification process, unless they are detachable, in which case they can be hidden and removed after reduction.

A network of processes is often represented as $SC[\overrightarrow{LTS^N}]$, where SC is a "process context". (Details on the context notation and related transformations are in Appendix B.) The compositionality theorem is in the form as follows:

Theorem 7.1 (Compositionality) [11] *LTS_1^N and LTS_2^N have untangled action sets U_1 and U_2 (respectively) implies $U_{\|}$ is untangled in $LTS_1^N \parallel LTS_2^N$, where $U_{\|} = (A_1 \cup A_2) \setminus ((A_1 \setminus U_1) \cup (A_2 \setminus U_2))$.*

Proof. Parallel composition of normalised LTSs gives a new normalised LTS. Given a (global) behaviour $\tilde{k} \in BH(LTS_1^N \parallel LTS_2^N)$, $\tilde{k} \upharpoonright A_j$ is a (local) behaviour of LTS_j for all $j \in \{1,2\}$. The definition of $U_{\|}$ ensures that any untangled action at the global level is also untangled in all local LTSs taking part in the action. Thus, if a (global) strategy for $\tilde{k}$ can result in an execution with a behaviour $\tilde{k}'$, following $\tilde{k}' \upharpoonright A_j$ will give a (local) strategy for $\tilde{k} \upharpoonright A_j$ on LTS_j.

Assume there is a finite rejecting strategy for a global behaviour $\tilde{k}$ which results in an execution with the behaviour t (before *reject*). If the rejection is due to **rule 2** on e, following $t \upharpoonright A_j$, where $e \in A_j$ and e is not enabled on LTS_j after trace $t \upharpoonright A_j$, gives a finite rejecting strategy for $\tilde{k} \upharpoonright A_j$.

If $\tilde{k}$ is divergent and the rejection is due to **rule 4**, then at least one of its projected local behaviour is a divergent trace, say $\tilde{k}_j \upharpoonright A_j$. Thus $(\tilde{k}_j \upharpoonright A_j)\tau^{\omega}$ is a divergence behaviour of LTS_j. Following $t \upharpoonright A_j$ gives a finite rejecting strategy for $(\tilde{k}_j \upharpoonright A_j)\tau^{\omega}$. Thus, untangledness on both local LTSs is untrue. □

Thus, our reduction works as follows:

(i) $SC[\overrightarrow{LTS^N}]$ can be transformed to $(\|[\overrightarrow{LTS'^N}]) \setminus \Delta$.

(ii) On each LTS'^N, find the maximal untangled subset of Δ, say U.

(iii) Use Theorem 7.1 to calculate the global untangled action set $U_{\|}$ from $\overrightarrow{U}$.

(iv) Apply $chase^+$ on the global system and we have the final reduced system: $chase^+(\|[\overrightarrow{LTS'^N}], U_{\|}) \setminus (\Delta \setminus U_{\|})$ [12].

Preliminary experiment. The CSP check in Appendix A was tested on the arbiter cell. It took a fraction of a second to correctly identify that the set of maximal untangled subset is $A \setminus \{a1+, a2+\}$. Theorem 7.1 then showed that all the actions in the tree arbiter, except those of $a1+$ and $a2+$, are untangled.

Since $chase^+$ is not available in FDR2 yet, *chase* is used instead to reduce the state space. Fortunately, this is correct due to the fact that a tree arbiter remains a divergence-free system after the untangled actions are hidden.

[11] Note that the maximality of untangled action sets is not necessarily preserved in this theorem.

[12] Another, potentially more efficient, approach is to push the hiding of $U_{\|}$ downwards along the parallel composition hierarchy as much as possible, and then apply nested $chase^+$ layer by layer.

We have checked the system using FDR2. The results are very encouraging compared to previous works [1,10]. The checking time is nearly linear in the size of the tree arbiter. More intriguingly the memory used is negligible (below 100MByte) and is sub-linear relative to the tree size. Thus, it is fair to say that the state explosion has been avoided.

8 Conclusion

We have proposed a truly compositional reduction technique for concurrent systems with a large number of small processes. Relative to previous works, the merits of the current work are summarised as follows:

- Our reduction technique is compositional and places minimal restrictions on the synchronisation potential of processes.
- It gives an accurate treatment of divergence despite the interference between divergence and compositionality. That is, a divergent process can delay other parallel processes indefinitely. Our solution is to keep lvgi actions visible and use fairness to guide the state space traversal out of unprogressing loops.
- It uses a weakest possible failure equivalence and thus has advantages in reduction.
- A hierarchy of partial determinacy properties are identified, e.g. untangledness, detachability and U-determinism (c.f. Figure 2 in Appendix C). They can be of independent interest. For instance, it seems possible that any CSP process equals a (possibly infinite) nondeterministic choice on a set of U-deterministic processes [19,15].
- Preliminary experiments show that state explosion can be avoided using our technique in the case of tree arbiters.

This paper presents mostly the theory part of our work. The priority in future work is to implement $chase^+$ in tools like FDR2 and to evaluate its performance on a larger class of systems.

Acknowledgements We are grateful to A.W. Roscoe for reading an earlier draft of this paper and giving valuable suggestions. The research is funded by EPSRC projects GR/S11091/01 & GR/S11084/01.

References

[1] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer and S. K. Rajamani. Partial-Order Reduction in Symbolic State Space Exploration. CAV 1997: 340-351.

[2] S. Blom. Partial τ-confluence for Efficient State Space Generation. Technical Report SEN-R0123, CWI, Amsterdam, 2001.

[3] S. Blom and J. van de Pol. State Space Reduction by Proving Confluence. CAV 2002: 596-609.

[4] A. Davis and S. M. Norwick. *An Introduction to Asynchronous Circuit Design.* The Encyclopedia of Computer Science and Technology (vol 38), Marcel Dekker, New York, 1998.

[5] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1993.

[6] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement: FDR2 User Manual*, 1999.

[7] J.F. Groote and J.C. van de Pol. State space reduction using partial tau-confluence. MFCS 2000, LNCS 1893.

[8] J. F. Groote and M. P. Sellink. Confluence for Process Verification. CONCUR 1995, LNCS 962.

[9] N. Lynch and M. Tuttle. An introduction to Input/Output automata. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, MIT, 1988.

[10] K. L. McMillan. Trace Theoretic Verification of Asynchronous Circuits Using Unfoldings. CAV 1995: 180-195.

[11] G. J. Pace, F. Lang and R. Mateescu. Calculating-Confluence Compositionally. CAV 2003: 446-459.

[12] D. Peled. Partial Order Reduction: Linear and Branching Temporal Logics and Process Algebras. Proceedings of POMIV'96, DIMACS Series Vol. 29, AMS, 1997.

[13] A. Puhakka and A. Valmari. Weakest-Congruence Results for Livelock-Preserving Equivalences. Proceedings of CONCUR '99, LNCS 1664.

[14] Y. S. Ramakrishna and S. A. Smolka. Partial-Order Reduction in the Weak Modal Mu-Calculus. CONCUR 1997, LNCS 1243.

[15] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1998.

[16] A. W. Roscoe. Seeing beyond divergence. Proceedings of "Symposium on the Occasion of 25 years of CSP", London, July 2004, LNCS 3525.

[17] A. W. Roscoe. The pursuit of buffer tolerance. Unpublished manuscript, 2005.

[18] A. W. Roscoe. Confluence thanks to extensional determinism. In Proceedings of Bertinoro meeting on Concurrency, BRICS 2005.

[19] A. W. Roscoe. Personal communication, March 2006.

[20] A. Valmari. The Weakest Deadlock-Preserving Congruence. Information Processing Letters 53 (1995) 341-346.

[21] A. Valmari. Stubborn Set Methods for Process Algebras. Proceedings of POMIV'96, DIMACS Series Vol. 29, AMS, 1997.

[22] X. Wang, M. Kwiatkowska. On process-algebraic verification of asynchronous circuits. Proceedings of ACSD 2006, IEEE Press, Turku, Finland.

[23] G. Winskel. Event structures. In Advances in Petri Nets 1986, Part II; 1987.

A CSP check for untangled actions

Our idea of formulating the untangledness decision problem as a CSP check is inspired by Roscoe's work on buffer tolerance [17]. It works as follows.

Two copies of the same LTS^N are put in parallel. One acts as the generator of behaviours, while the other acts as the acceptor of behaviours. The former's behaviours are forced onto the latter by *Agent*. If no rejection ever occurs in this system (i.e. refinement of *reject_free* process below), then every generated behaviour has been must-accepted

Assume there is a bijection $^+$ from A to another set disjoint to A. $^+$ can be used as a unary suffix operator so that we have e^+ and A^+. Similarly, we assume bijection * (with A^* and A^+ disjoint). $^{\mathcal{I}}$ is the identity function on A. Then we can define the CSP check. (Note that $[^+ \cup ^*]$ below is the renaming operator).

Definition A.1 Given divergence-free LTS^N and $A_i \cup A_v \subseteq A$,

$$Check(A_i, A_v, LTS^N) \mathrel{\widehat{=}} reject\_free \sqsubseteq_F LTS^N[^+ \cup ^*] \parallel Agent(\epsilon, \epsilon) \parallel LTS^N[^{\mathcal{I}} \cup ^*]$$

where, $reject\_free = Stop \sqcap (?x : A \cup A^+ \cup A^* \to reject\_free) \Box (f \to \bigsqcap x : A \bullet x^+ \to reject\_free)$ and

$$\begin{aligned}
&Agent(ext, buf) = \\
&buf = \epsilon \& ?x : A \setminus \{ext\} \to Agent(ext, x) \\
&\Box\ ext = \epsilon \& \Box\, x : (A_i \setminus \{buf\}) \bullet x^+ \to Agent(x, buf) \\
&\Box\ buf \in A_i \& \Box\, x : A_v \bullet x^* \to Agent(ext, buf) \\
&\Box\ ext \neq \epsilon \& ext \to Agent(\epsilon, buf) \\
&\Box\ buf \neq \epsilon \& f \to buf^+ \to Agent(ext, \epsilon)
\end{aligned}$$

Note that the alphabet of $Agent(ext, buf)$ is $A \cup A^+ \cup A^* \cup \{f\}$.

However, one complication is that, unlike the previous definition of strategy, in CSP implementation the acceptor cannot know a priori the whole sequence of the generated behaviour (much less to operate on it). This can be solved if we observe that the re-ordering power of the acceptor can be restricted without affecting the set of must-accepted behaviours. That is, at any time, the acceptor is allowed only to see and operate on the top two elements of the current suffix (implemented by using *buf* to buffer the top one) and it is only allowed to freely make one A_i action at a time (stored in *ext*), i.e. no further freely-made A_i action (i.e. those that do not match the top two elements of the current suffix) can happen before the last one is eventually expended in consuming a pending action. The correctness of this reduction is based on the following proposition. It is an extension of the solution suggested in Section 8 of [17].

Proposition A.2 *Given divergence-free LTS^N and $A_i \cup A_v = A$, A_i is untangled iff the following condition is not true:*

***c0**: LTS^N has a pair of finite traces (t, u) such that $u \upharpoonright_{A_v} \leq t \upharpoonright_{A_v}$ and $ue \notin FT$, where $e = head(t - u)$.*

Proof. From **c0** to tangledness: t has a finite rejecting strategy that simply follows u before issuing *reject* using **rule 2**. The other direction: with divergence-freedom and Proposition 5.6, there is a finite rejecting strategy for some $\tilde{t} \in BH$ that results in an execution $u' \in FT$ before issuing *reject*. The pair $(\tilde{t}, u')$ satisfies **c0**. □

Proposition A.3 *Given LTS^N and $A_i \cup A_v = A$, **c0** is true iff one of the following conditions is true:*

***c1**: LTS^N has both trace teu and te' but not $te'e(u - e')$, for some t, u, e and e' such that $e' \in A_i \wedge e \neq e'$.*

***c2**: LTS^N has both trace $tee'u$ and te' but not $te'eu$, for some t, u, e and e' such that $e \in A_i$ and $e' \in A_v$.*

Proof. From **c1** or **c2** to **c0** is straightforward.

For the other direction, assume both **c1** and **c2** are not true but **c0** is true. **c1** is not true implies $e' \in A_i$ can be inserted or moved forward (towards the head) if it is also enabled at that position. **c2** is not true implies $e' \in A_v$ can also be moved forward one step if it is also enabled at that position.

c0 is true implies there exists the pair (t, u). Let t' be the minimal prefix of t such that $u \upharpoonright_{A_v} = t' \upharpoonright_{A_v}$ and t' contains just one more e element than u. Then (t', u) is also a pair satisfying **c0**.

Compare *head* u and *head* t'. If matching, then output t'. Otherwise, consider *head* u. If it is a member of A_i, then output $(head\ u)(t' - head\ u)$. If it is a member of A_v, then it must be an element of t'. Assume the first such element in t' is prefixed by $t''e'$. Then $t''e'$ only contains A_i elements. Inserting each element of t'' before *head* u in the same order gives us $t''(head\ u) \in FT$. Thus $(head\ u)$ can be moved forward one step in t'. Similarly, it can continue to be moved forward until reaching the head position. Output the result trace.

The output of the above procedure, say u', is in FT and its first element is the same as u. Continue to use the procedure on the second element and so on... It eventually leads to the output $u' = ue$. Therefore, contradiction. □

Proposition A.4 *Given divergence-free LTS^N and A_i, $Check(A_i, A \setminus A_i, LTS^N)$ is true iff A_i is untangled.*

Proof. Note that the RHS of the refinement is divergence-free and deterministic and its finite traces are a subset of the LHS. Thus, the refinement fails iff deadlock happens at the state that is after the occurrence of f but before the occurrence of buf^+, since it is the only place where the LHS does not have the maximal refusals, i.e. deadlock.

It is easy to see that if either **c1** or **c2** happens, it will lead to the special deadlock state.

Assume, if deadlock indeed happens at the special state, the trace on the generator is te and the trace on the acceptor is u (deadlock on forcing e). Then $t - u = \epsilon \wedge u - t \in A_i \cup \{\epsilon\} \wedge t \upharpoonright_{A_v} = u \upharpoonright_{A_v}$ and thus (te, u) satisfies **c0**. □

To extend the above technique to normalised LTS with divergences, some transformation on the τ self-loops in the LTS is in order. Given LTS^N, function $DCT(LTS^N)$ outputs another normalised divergence-free LTS (with alphabet $A \cup \{d\}$) exactly like LTS^N but with all τ self-loops replaced by a special $d \notin A$ transition to a deadlock state.

Proposition A.5 *Given LTS^N and $A_i \cup A_v = A$, A_i is untangled iff both* ***c0*** *and* ***c0′*** *are not true, where*

c0′*: LTS^N has a pair of finite traces (t, u) such that u contains t, $u \upharpoonright_{A_v} = t \upharpoonright_{A_v}$, $t \in DT$ and $u \notin DT$.*

Proof. **c0** to tangledness is in Proposition A.2. **c0′** to tangledness: $t\tau^\omega$ is in BH and has a finite rejecting strategy that simply follows u and issues *reject* using **rule 4**.

Tangledness to **c0** or **c0′**: with Proposition 5.6, there is a finite rejecting strategy for some $\tilde{k} \in BH$ that results in an execution $u' \in FT$ before issuing *reject* using **rule 2** or **rule 4**. Let $\tilde{t} = \tilde{k} \upharpoonright_A$. If **rule 2** is used, $\tilde{t}$ has a prefix t' such that all the removed elements in the operation $\tilde{t} - u'e$ are from t'. The pair (t', u') satisfies **c0**. Otherwise, $\tilde{t}$ is finite and $u' \notin DT$. The pair $(\tilde{t}, u')$ satisfies **c0′**. □

Theorem A.6 *Given LTS^N and $A_i \cup A_v = A$, $Check(A_i, A_v, DCT(LTS^N))$ is successful iff A_i is untangled.*

Proof. Note that the special action $d \notin A_i \cup A_v$. Thus, d cannot exchange order with any other action and can only be the last element in a trace. If $Check(A_i, A_v, DCT(LTS^N))$ is unsuccessful and deadlocking on e after a trace te on the generator and a trace u on the acceptor (deadlock on forcing e). Since d is not in t or u, (t, u) satisfies **c0′** when $e = d$, and (te, u) satisfies **c0** when $e \neq d$.

The other direction: when **c0** is true and the pair is (t, u), (t, u) also satisfies **c0** on $DCT(LTS^N)$; when **c0′** is true and the pair is (t, u), (td, u) satisfies **c0** on $DCT(LTS^N)$. Both of them are reducible to **c1** or **c2** on $DCT(LTS^N)$. Thus, the special deadlock is reachable. □

B Context and super-combinator

Assume an infinite set of SCS variables (typed by their alphabets) X ranged over by x. A *Generalised Context* GC is defined as:

$$GC ::= LTS \mid x \mid GC \parallel GC' \mid GC \setminus \Delta \mid GC[R^{1-1}]$$

where any variable x in GC must have a unique occurrence.

Often we use $\overrightarrow{x}$ to represent the vector of variables in GC, and, to make them explicit, GC can be equivalently written as $GC[\overrightarrow{x}]$. If there is no LTS occurring in GC (i.e. GC is made up of only variables and operators), we say it is a *super-combinator*, written as $SC[\overrightarrow{x}]$. If GC has only one variable, we say it is a *context* and the variable is its *hole*, written as $C[\cdot]$. Given $GC[\overrightarrow{x}]$ and a vector $\overrightarrow{SCS}$ of *compatible* SCSs, substituting $\overrightarrow{SCS}$ for $\overrightarrow{x}$ in GC gives us a new SCS, written as $GC[\overrightarrow{SCS}]$. It is easy to see that any SCS can be written in the form of $SC[\overrightarrow{LTS}]$.

Definition B.1 $\overrightarrow{SCS}$ is *compatible* with $GC[\overrightarrow{x}]$ iff each SCS in $\overrightarrow{SCS}$ has the same alphabet as that of the corresponding x in $\overrightarrow{x}$.

Theorem B.2 ([15]) *Given $SC[\overrightarrow{LTS}]$, it has a normal form, $(\|[\overrightarrow{LTS'}]) \setminus \Delta$, where $\overrightarrow{LTS}$ and $\overrightarrow{LTS'}$ are of the same dimension and $\|$ is a super-combinator consisting* *of only parallel operators, such that*

(i) *the corresponding members of $\overrightarrow{LTS}$ and $\overrightarrow{LTS'}$ are renaming-isomorphic.*

(ii) *$SC[\overrightarrow{LTS}]$ and $(\|[\overrightarrow{LTS'}]) \setminus \Delta$ are isomorphic.*

Note that LTS and LTS' are renaming-isomorphic iff there is a 1-to-1 renaming on one of them and the resulting LTS is isomorphic to the other.

C Partial Determinacy Hierarchy

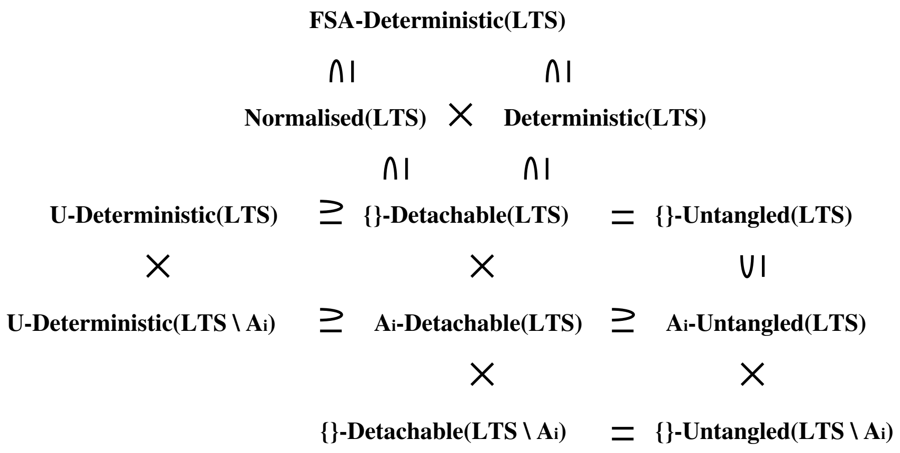

Figure 2. A Hierarchy of Partial Determinacy Properties

In Figure 2, the LHS property of $\supseteq$ is weaker than the RHS property. **X** means that the two properties are incomparable. Determinism is used to denote CSP determinism [15]. FSA-determinism denotes the determinism in classical automata theory.