Conjunction on Processes: Full–Abstraction via Ready–Tree Semantics

Gerald Lüttgen[1]* and Walter Vogler[2]

[1] Department of Computer Science, University of York, York YO10 5DD, U.K.
luettgen@cs.york.ac.uk
[2] Institut für Informatik, Universität Augsburg, D–86135 Augsburg, Germany
vogler@informatik.uni-augsburg.de

Abstract. A key problem in mixing operational (e.g., process–algebraic) and declarative (e.g., logical) styles of specification is how to deal with inconsistencies arising when composing processes under conjunction. This paper introduces a conjunction operator on labelled transition systems capturing the basic intuition of "*a and b = false*", and considers a naive preorder that demands that an inconsistent specification can only be refined by an inconsistent implementation.
The main body of the paper is concerned with characterising the largest precongruence contained in the naive preorder. This characterisation will be based on a novel semantics called ready–tree semantics, which refines ready traces but is coarser than ready simulation. It is proved that the induced ready–tree preorder is compositional and fully–abstract, and that the conjunction operator indeed reflects conjunction.
The paper's results provide a foundation for, and an important step towards a unified framework that allows one to freely mix operators from process algebras and temporal logics.

1 Introduction

Process algebra [2] and *temporal logic* [14] are two popular approaches to formally specifying and reasoning about reactive systems. The process–algebraic paradigm is founded on notions of *refinement*, where one typically formulates a system specification and its implementation in the same notation and then proves using *compositional reasoning* that the latter refines the former. The underlying semantics is often given operationally, and refinement relations are formalised as precongruences. In contrast, the temporal–logic paradigm is based on the use of temporal logics to formulate specifications abstractly, with implementations being denoted in an operational notation. One then verifies a system by establishing that it is a model of its specification.

Recently, two papers have been published aimed at marrying process algebras and temporal logics [5, 6]. While the first paper introduces a semantic framework based on Büchi automata, the second paper considers labelled transition

* Research support was partially provided by the NSF under grant CCR–9988489.

systems augmented with an "unimplementability predicate". This predicate captures *inconsistencies* arising when composing processes conjunctively; e.g., the composition $a \wedge b$ is contradictory since a run of a process cannot begin with both actions a and b. Moreover, the frameworks in [5, 6] are equipped with a refinement preorder based on De Nicola and Hennessy's must–testing preorder [12]. However, the obtained results are unsatisfactory: the refinement preorder in [5] is not a precongruence, while the $\wedge$–operator in [6] is not *conjunction* with respect to the studied precongruence.

This paper solves the deficiencies of [5, 6] within a simple setting of labelled transition systems in which a state represents either external (non–deterministic) or internal (disjunctive) choice. Moreover, states that are vacuously *true* or *false* are tagged accordingly. The tagging of *false* states, or inconsistent states, is given by an inductive *inconsistency predicate* that is defined very similar but subtly different to the unimplementability predicate of [6]. We then equip our setting with two operators: the conjunction operator $\wedge$ is in essence a synchronous composition on observable actions and an interleaving product on the unobservable action τ, but additionally captures inconsistencies; the disjunction operator $\vee$ simply resembles the process–algebraic operator of internal choice.

Our variant of labelled transition systems gives rise to a naive refinement preorder $\sqsubseteq_F$ requiring that an inconsistent specification cannot be refined except by an inconsistent implementation. We characterise the *consistency preorder*, i.e. the largest precongruence contained in $\sqsubseteq_F$ when conjunctively closing under all contexts. To do so, we define a novel semantics, called *ready–tree semantics* which is — at least when disallowing divergent behaviour — finer than both must–testing semantics [12] and ready–trace semantics [7], but coarser than ready simulation [3]. The resulting *ready–tree preorder* $\precsim$ is not only compositional for $\wedge$ and $\vee$ and fully–abstract with respect to $\sqsubseteq_F$, but also possesses several other desired properties. In particular, we prove that $\wedge$ ($\vee$) is indeed conjunction (disjunction) relative to $\precsim$, and that $\wedge$ and $\vee$ satisfy the expected boolean laws, such as the distributivity laws.

Our results are a significant first step towards the goal of developing a uniform calculus in which one can freely mix process–algebraic and temporal–logic operators. This will give engineers powerful tools to model system components at different levels of abstraction and to impose logical constraints on the execution behaviour of components. The proposed ready–tree preorder will allow engineers to step–wise and component–wise refine systems by trading off logical content for operational content.

Organisation. The next section presents our setting of labelled transition systems augmented with *true* and *false* predicates, together with a conjunction and a disjunction operator. Sec. 3 defines the novel ready–tree semantics, addresses expressiveness issues of several ready–tree variants and introduces the ready–tree preorder. Our compositionality and full–abstraction results are stated in Sec. 4. All proofs can be found in a technical report [11]. Finally, Sec. 5 discusses our results in light of related work, while Sec. 6 presents our conclusions and suggests directions for future research.

2 Labelled Transition Systems & Conjunction

This section first introduces our process–algebraic setting and particularly conjunctive composition informally, discusses semantic choices and their implications, and finally gives a formal account of our framework.

Motivation. Our setting models processes as labelled transition systems, which may be composed conjunctively and disjunctively. As usual in process algebra, transition labels are actions taken from some alphabet $\mathcal{A} = \{a, b, \ldots\}$. When an action a is offered by the environment and the process under consideration is in a state having one or more outgoing a–transitions, the process must choose and perform one of them. If there is no outgoing a–transition, then the process stays in its state, at least in classical process–algebraic frameworks where the composition between a process and its environment is modelled using some parallel operator. However, in a conjunctive setting we wish to mark the composed state between process and environment as inconsistent, if the environment offers an action that the process cannot perform, or vice versa. Hence, taking ordinary synchronous composition as operator for conjunction is insufficient.

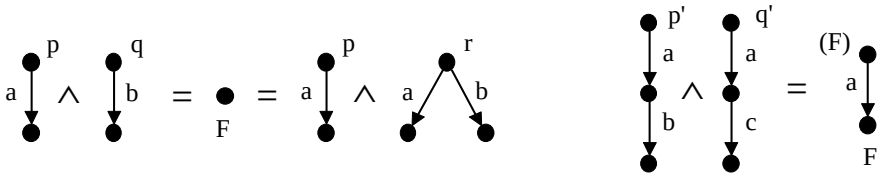

Fig. 1. Basic intuition behind conjunctive composition.

We illustrate this intuition behind our conjunction operator $\wedge$ and its implications by the example labelled transition systems of Fig. 1. First, consider the processes p, q and r. Process p and q specify that exactly action a and respectively action b is offered initially. Similarly, process r specifies that a and b are offered initially. From this perspective, $p \wedge q$ as well as $p \wedge r$ are *inconsistent* and should be tagged as such. Formally, our labelled transition systems will be augmented by an *inconsistency predicate* F, so that $p \wedge q$, $p \wedge r \in F$ in our example. We also refer to inconsistent states as *false*–states.

Now consider the conjunction $p' \wedge q'$ shown on the right in Fig. 1. Since both conjuncts require action a to be performed, $p' \wedge q'$ should have an a–transition. From the preceding discussion, this transition should lead to a *false*–state. No sensible process can meet these requirements of being able to perform a and being inconsistent afterwards. Thus, our inconsistency predicate will propagate backwards to the conjunction itself, as indicated in Fig. 1.

Fig. 2 shows more intricate examples of backward propagation. The inconsistency of the target state of the a–transition of the process on the left propagates backwards to its source state. This is the case although the source state is able to offer a transition leading to a consistent state. However, that transition can only be taken if the environment offers action b. The process is forced into the inconsistency when the environment offers action a.

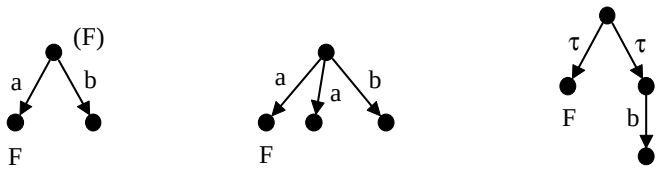

Fig. 2. Backward propagation of inconsistencies.

The situation is different for the process in the middle, which has an additional a–transition leading to a consistent state. Here, the process is consistent, as it can choose to execute this new a–transition and thus avoid to enter a *false*–state. In fact, this choice can be viewed as a disjunction between the two a–branches. As an aside, note that in [6] the design decision was to consider a process already as inconsistent if some a–derivative is. While there might be an intuitive justification for that, it led to a setting where the implied conjunction operator does not reflect conjunction for the studied refinement preorder, i.e., where Thm. 20(1) does not hold.

Disjunction can be made explicit by using the classical *internal–choice* operator. This operator may as usual be expressed by employing the special, unobservable action $\tau \notin \mathcal{A}$ as shown on the right in Fig. 2. Hence, we may identify the internal–choice operator with the disjunction operator $\vee$ desired in our setting. Moreover, a disjunction $p \vee q$ is inconsistent if both p and q are *false*–states. In particular, the process on the right in Fig. 2 will represent *false* $\vee\, q$ in our approach, with q from Fig. 1, which clearly should be consistent.

Formalisation. For notational convenience we denote $\mathcal{A} \cup \{\tau\}$ by $\mathcal{A}_\tau$ and use $\alpha, \beta, \ldots$ as representatives of $\mathcal{A}_\tau$. We start off be defining our notion of *labelled transition systems* (LTS). The LTSs considered here are augmented with a *false*–predicate F on states, as discussed above, and dually with a *true*–predicate T. A state in F represents inconsistent, empty behaviour, while a state in T represents completely underspecified, arbitrary behaviour.

Formally, an LTS is a quadruple $\langle P, \longrightarrow, T, F\rangle$, where P is the set of *processes* (states), $\longrightarrow \subseteq P \times \mathcal{A}_\tau \times P$ is the *transition relation*, and $T \subseteq P$ and $F \subseteq P$ is the *true–predicate* and the *false–predicate*, respectively. We write $p \xrightarrow{\alpha} p'$ instead of $\langle p, \alpha, p'\rangle \in \longrightarrow$, $p \xrightarrow{\alpha}$ instead of $\exists p' \in P.\, p \xrightarrow{\alpha} p'$, and $p \longrightarrow$ instead of $\exists p' \in P,\, \alpha \in \mathcal{A}_\tau.\, p \xrightarrow{\alpha} p'$. When $p \xrightarrow{\alpha} p'$, we say that process p can perform an α–step to p', and we call p' an α–derivative. We also require an LTS to satisfy the following τ–*purity* condition: $p \xrightarrow{\tau}$ implies $\nexists a \in \mathcal{A}.\, p \xrightarrow{a}$, for all $p \in P$. Hence, each process represents either an external or internal (disjunctive) choice between its outgoing transitions. This restriction turns out to be technically convenient, and we leave exploring the consequences of lifting it for future work. The LTSs of interest to us need to satisfy four further properties, as stated in the following formal definition:

Definition 1 (Logical LTS). An LTS $\langle P, \longrightarrow, T, F\rangle$ is a *logical LTS* if it satisfies the following conditions:

1. $T \cap F = \emptyset$

2. $T \subseteq \{p \mid p \not\longrightarrow\}$
3. $F \subseteq P$ such that $p \in F$ whenever $\exists \alpha \in \mathcal{I}(p)\, \forall p' \in P.\ p \xrightarrow{\alpha} p' \implies p' \in F$
4. p cannot stabilise $\implies p \in F$

Naturally, we require that a process cannot be tagged *true* and *false* at the same time. As a *true*–process specifies arbitrary, full behaviour, any behaviour made explicit by outgoing transitions is already included implicitly; hence, any outgoing transitions may simply be cut off. The third condition formalises the backwards propagation of inconsistencies as discussed in the motivation section above; here, $\mathcal{I}(p)$ stands for the set $\{\alpha \in \mathcal{A}_\tau \mid p \xrightarrow{\alpha}\}$ of initial actions of process p, to which we also refer as *ready set*.

The fourth condition relates to *divergence*, i.e., infinite sequences of τ–transitions. In many semantic frameworks, e.g. [12, 7], divergence is considered catastrophic, while in our setting catastrophic behaviour is inconsistent behaviour. We view divergence only as catastrophic if a process cannot *stabilise*, i.e., if it cannot get out of an infinite, internal computation. While this is intuitive, there is also a technical reason to which we will come back shortly.

To formalise our notion of stabilisation, we first introduce a weak transition relation $\Longrightarrow \subseteq P \times (\mathcal{A}_\tau \cup \{\epsilon\}) \times P$ which is defined by (1) $p \overset{\varepsilon}{\Longrightarrow} p'$ if $p \equiv p' \notin F$, where $\equiv$ denotes syntactic equality, or if $p \notin F$ and $p \xrightarrow{\tau} p'' \overset{\varepsilon}{\Longrightarrow} p'$ for some p'', and (2) $p \overset{a}{\Longrightarrow} p'$ if $p \notin F$ and $p \xrightarrow{a} p'' \overset{\varepsilon}{\Longrightarrow} p'$ for some p''. Our definition of a weak transition is slightly unusual: a weak transition cannot pass through *false*–states since these cannot occur in computations, and it does not abstract from τ–transitions preceding a visible transition. However, we only will use weak visible transitions from *stable* states, i.e., states with no outgoing τ–transition. Finally, we can now formalise stabilisation: a process p *can stabilise* if $p \overset{\varepsilon}{\Longrightarrow} p'$ for some stable p'.

Note that both Conds. (3) and (4) are inductively defined conditions. We refer to them as *fixed point conditions of F for LTS* . For convenience, we will often write LTS instead of logical LTS in the sequel. Moreover, whenever we mention a process p without stating a respective LTS explicitly, we assume implicitly that such an LTS $\langle P, \longrightarrow, T, F\rangle$ is given. We let tt (ff) stand for the *true* (*false*) process, which is the only process of an LTS with $tt \in T$ ($ff \in F$).

Operators. Our conjunction operator is essentially a synchronous composition for visible transitions and an asynchronous composition for τ–transitions. However, we need to take care of the T– and F–predicates.

Definition 2 (Conjunction Operator). The conjunction of two logical LTSs $\langle P, \longrightarrow_P, T_P, F_P\rangle$, $\langle Q, \longrightarrow_Q, T_Q, F_Q\rangle$ is the LTS $\langle P \wedge Q, \longrightarrow_{P\wedge Q}, T_{P\wedge Q}, F_{P\wedge Q}\rangle$ defined by:

– $P \wedge Q =_{\mathrm{df}} \{p \wedge q \mid p \in P,\, q \in Q\}$

– $\longrightarrow_{P\wedge Q}$ is determined by the following operational rules:

$$\begin{array}{rcl} p \xrightarrow{\tau}_P p' & \Longrightarrow & p\wedge q \xrightarrow{\tau}_{P\wedge Q} p'\wedge q \\ q \xrightarrow{\tau}_Q q' & \Longrightarrow & p\wedge q \xrightarrow{\tau}_{P\wedge Q} p\wedge q' \\ p \xrightarrow{a}_P p',\ q \xrightarrow{a}_Q q' & \Longrightarrow & p\wedge q \xrightarrow{a}_{P\wedge Q} p'\wedge q' \\ q\in T_Q,\ p \xrightarrow{a}_P p' & \Longrightarrow & p\wedge q \xrightarrow{a}_{P\wedge Q} p'\wedge q \\ p\in T_P,\ q \xrightarrow{a}_Q q' & \Longrightarrow & p\wedge q \xrightarrow{a}_{P\wedge Q} p\wedge q' \end{array}$$

– $p\wedge q\in T_{P\wedge Q}$ if and only if $p\in T_P$ and $q\in T_Q$
– $p\wedge q\in F_{P\wedge Q}$ if at least one of the following conditions holds:
  1. $p\in F_P$ or $q\in F_Q$
  2. $p\notin T_P$ and $q\notin T_Q$ and $p\wedge q \xrightarrow{\tau}\!\!\!\!\!/\ _{P\wedge Q}$ and $\mathcal{I}(P)\neq\mathcal{I}(Q)$
  3. $\exists\alpha\in\mathcal{I}(p\wedge q)\,\forall p'\wedge q'.\, p\wedge q \xrightarrow{\alpha}_{P\wedge Q} p'\wedge q' \implies p'\wedge q'\in F_{P\wedge Q}$
  4. $p\wedge q$ cannot stabilise

Note that the treatment of *true*–processes when defining $\longrightarrow_{P\wedge Q}$ and $T_{P\wedge Q}$ reflects our intuition that these processes allow arbitrary behaviour. We are left with explaining Conds. (1)–(4). Firstly, a conjunction is inconsistent if any conjunct is. Conds. (2) and (3) reflect our intuition of inconsistency and, respectively, backward propagation stated in the motivation section above. Cond. (4) is added to enforce Def. 1(4). We refer to Conds. (3) and (4) as *fixed point conditions of F for $\wedge$*.

It is easy to check that conjunction is well–defined, i.e., that the conjunctive composition of two logical LTSs satisfies the four conditions of Def. 1. For Def. 1(1) in particular, note that $p\wedge q\in T_{P\wedge Q}$ does not satisfy any of the four conditions for $F_{P\wedge Q}$.

We may now demonstrate why we have treated non–escapable divergence as catastrophic in our setting. This is because, otherwise, our conjunction operator would not be associative as demonstrated by the example depicted in Fig. 3. If the conjunction is computed from the left, the result is the first conjunct. Computed from the right, the result is the same but with both processes being in F. Hence, in the first case, the divergence hides the inconsistency. Since this is not really plausible and associativity of conjunction is clearly desirable, we need some restriction for divergence; it turns out that restricting divergence to escapable divergence, i.e., potential stabilisation, is sufficient for our purposes.

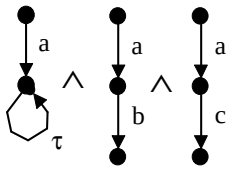

Fig. 3. Counter–example.

Definition 3 (Disjunction Operator). The disjunction of two logical LTSs $\langle P, \longrightarrow_P, T_P, F_P\rangle$ and $\langle Q, \longrightarrow_Q, T_Q, F_Q\rangle$ satisfying (w.l.o.g.) $P\cap Q=\emptyset$, is the logical LTS $\langle P\vee Q, \longrightarrow_{P\vee Q}, T_{P\vee Q}, F_{P\vee Q}\rangle$ defined by:

– $P\vee Q =_{\mathrm{df}} \{p\vee q \,|\, p\in P,\, q\in Q\}\cup P\cup Q$

– $\longrightarrow_{P \vee Q}$ is determined by the following operational rules:

$$\begin{array}{rcl} \text{always} & \Longrightarrow & p \vee q \xrightarrow{\tau}_{P \vee Q} p \\ \text{always} & \Longrightarrow & p \vee q \xrightarrow{\tau}_{P \vee Q} q \\ p \xrightarrow{\alpha}_{P} p' & \Longrightarrow & p \xrightarrow{\alpha}_{P \vee Q} p' \\ q \xrightarrow{\alpha}_{Q} q' & \Longrightarrow & q \xrightarrow{\alpha}_{P \vee Q} q' \end{array}$$

– $p \vee q \notin T_{P \vee Q}$ always
– $p \vee q \in F_{P \vee Q}$ if and only if $p \in F_P$ and $q \in F_Q$

The definition of disjunction, which reflects internal choice, is quite straightforward and well–defined. Only the definition of $T_{P \vee Q}$ for $p \vee q$ is unusual, as one would expect to simply have $p \vee q \in T$ whenever p or q is in T. However, then Cond. (2) of Def. 1 would be violated. Our alternative definition respects this condition and is semantically equivalent. In the sequel we leave out indices of relations and predicates whenever the context is clear.

Refinement preorder. As the basis for our semantical considerations we now define a naive refinement preorder stating that an inconsistent specification cannot be implemented except by an inconsistent implementation.

Definition 4 (Naive Consistency Preorder). The *naive consistency preorder* $\sqsubseteq_F$ on processes is defined by $p \sqsubseteq_F q$ if $p \in F \implies q \in F$.

One main objective of this paper is to identify the corresponding fully–abstract preorder with respect to conjunction and disjunction, which is contained in $\sqsubseteq_F$. Our approach follow the testing idea of De Nicola and Hennessy [12], for which we define a testing relation $\sqsubseteq$ as usual. Note that a process and an observer need to be composed not simply synchronously but conjunctively. This is because we want the observer to be sensitive to inconsistencies, so that $p \sqsubseteq q$ if each "conjunctive observer" that sees an inconsistency in p also sees one in q.

Definition 5 (Consistency Testing Preorder). The *consistency testing preorder* $\sqsubseteq$ on processes is defined as the conjunctive closure of the naive consistency preorder under all processes (observers), i.e., $p \sqsubseteq q$ if $\forall o.\ p \wedge o \sqsubseteq_F q \wedge o$.

To characterise the full–abstract precongruence contained in $\sqsubseteq_F$ we will introduce a new semantics, called *ready–tree semantics*, and an associated preorder, the *ready–tree preorder*, which is compositional for conjunction and disjunction and which coincides with $\sqsubseteq$.

Example. As an illustration for our approach, consider process *spec* in Fig. 4. For $\mathcal{A} = \{a, b, c\}$, *spec* specifies that action c can only occur after action a. In the light of the above discussions, an implementation should offer initially either just a, or a and b, or just b, so that *spec* is an internal choice between three states. Moreover, after an action a, nothing more is specified; after an action b, the same is required as initially. While our specification of this simple behaviour may look quite complex, we may imagine that process *spec* is generated automatically from a temporal–logic formula. Fig. 4 also shows process *impl* which repeats sequence *abc*, and *spec* $\wedge$ *impl*. It will turn out that *spec* $\sqsubseteq$ *impl* (cf. Sec. 4).

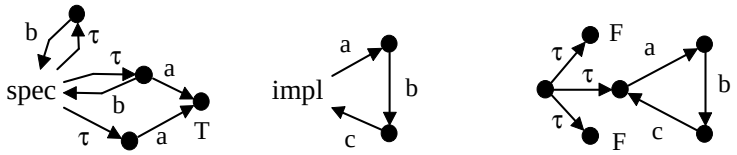

Fig. 4. Example processes.

3 Ready–Tree Semantics

A first guess for achieving a compositional semantics reflecting consistency testing is to use a kind of *ready–trace semantics* [7]. Such a semantics would refine trace semantics by checking the initial action set of every stable state along each trace. However, this is not sufficient when dealing with inconsistencies, since inconsistencies propagate backwards along traces as explained in Sec. 2. It turns out that a set of tree–like observations is needed, which leads to a novel denotational–style semantics which we call *ready–tree semantics*.

Observation trees & ready trees. A tree–like observation can itself be seen as a deterministic LTS with empty F–predicate, reflecting that observers are internally consistent.

Definition 6 (Observation Tree). An *observation tree* is a LTS $\langle V, \rightarrow, T, \emptyset \rangle$ satisfying the following properties:

1. $\langle V, \longrightarrow \rangle$ is a tree whose root is referred to as v_0
2. $\forall v \in V.\ v$ stable
3. $\forall v \in V, a \in \mathcal{I}(v)\ \exists_1 v' \in V.\ v \xrightarrow{a} v'$

We often denote such an observation tree by its root v_0. Next we define the observations of a process p, called *ready trees*. Note that p can only be observed at its stable states.

Definition 7 (Ready Tree). An observation tree v_0 is a *ready tree of* p if there is a labelling $h : V \longrightarrow P$ satisfying the following conditions:

1. $\forall v \in V.\ h(v)$ stable and $h(v) \notin F$
2. $p \stackrel{\varepsilon}{\Longrightarrow} h(v_0)$
3. $\forall v \in V, a \in \mathcal{A}.\ v \xrightarrow{a} v'$ implies (a) $h(v') = h(v) \in T$ or (b) $h(v) \stackrel{a}{\Longrightarrow} h(v')$
4. $\forall v \in V.\ (v \notin T$ and $h(v) \notin T)$ implies $\mathcal{I}(v) = \mathcal{I}(h(v))$

Intuitively, nodes v in a ready tree represent stable states $h(v)$ of p (cf. Cond. (1), first part) and transitions represent computations containing exactly one observable action (cf. Cond. (3)(b)). Since computations do not contain *false*–states, no represented state is in F (cf. Cond. (1), second part). Since p might not be stable, the root v_0 of a ready tree represents a stable state reachable from p by some internal computation (cf. Cond. (2)). If the state $h(v)$ represented by node v is in T, the subtree of v is arbitrary since $h(v)$ is considered to be completely

underspecified (cf. Conds. (3)(a) and (4)). In case $h(v) \notin T$, one distinguishes two cases: (i) if $v \notin T$, then v and $h(v)$ must have the same initial actions, i.e., the same *ready set*; (ii) if $v \in T$, the observation stops at this node and nothing is required in Conds. (3) and (4).

In the following, we write $\mathrm{RT}(p)$ for the set of all ready trees of p, $\mathrm{fRT}(p)$ for the set of all ready trees of p that have finite depth (*finite–depth* ready trees), and $\mathrm{cRT}(p)$ for the set of ready trees $\langle V, \longrightarrow, T, \emptyset \rangle$ where $T = \emptyset$ (*complete* ready trees). Note that a complete ready tree is called *complete* as it never stops its task of observing; hence, complete ready trees are often infinite in practice. Moreover, *false*–states may be characterised as follows:

Lemma 8. *$RT(p) = \emptyset$ if and only if $p \in F$.*

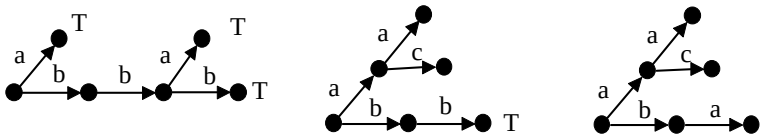

Fig. 5. Some ready trees of *spec*.

We illustrate our concept of ready trees by returning to our example of Fig. 4. Some of the ready trees of process *spec* are shown in Fig. 5. In the first ready tree, the observation stops after the third b. In the second tree, we see that we can observe an arbitrary tree after a, since the respective state of *spec* is in T. An arbitrary tree can also consist of just the root, as shown for the right–most a in the third tree; this tree is also complete. Process *impl* in Fig. 4 has only one complete ready tree which is an infinite path repeating sequence *abc*; this is also a ready tree of *spec*.

Ready–tree preorder & expressiveness. Our ready–tree semantics suggests the following refinement preorder:

Definition 9 (Ready–Tree Preorder). The *ready–tree preorder* $\sqsubseteq\!\!\!\sim$ on processes is defined as reverse ready–tree inclusion, i.e., $p \sqsubseteq\!\!\!\sim q$ if $\mathrm{RT}(q) \subseteq \mathrm{RT}(p)$.

This preorder will turn out to be the desired fully–abstract preorder contained in the naive consistency preorder. We first show that $\sqsubseteq\!\!\!\sim$ could just as well be formulated on the basis of complete ready trees and, for finitely branching LTS, of finite–depth ready trees. A crucial notion for our results is the following:

Definition 10 (Ready–Tree Prefix). Ready tree v_0 is *prefix* of ready tree w_0, in signs $v_0 \leq w_0$, if there exists an injective mapping $\rho : V \hookrightarrow W$ such that:

1. $\rho(v_0) = w_0$
2. $v \xrightarrow{a} v' \implies \rho(v) \xrightarrow{a} \rho(v')$
3. $\rho(v) \xrightarrow{a} w' \implies v \in T$ or $(\exists v'.\ v \xrightarrow{a} v'$ and $\rho(v') = w')$
4. $\rho(v) \in T \implies v \in T$

Intuitively, one observation is a prefix of another if it stops observing earlier. Recall that a *true*–node indicates that observation stops (cf. Cond. (3)). Intuitively, we obtain a prefix of w_0 by cutting all transitions from some nodes (and adding them to T), while cutting just some transitions of a node is not allowed.

Lemma 11. $\{v_0 \mid \exists w_0 \in cRT(p).\ v_0 \leq w_0\} = RT(p)$.

As a consequence, we obtain the following corollary:

Corollary 12.

1. *$RT(p)$ is uniquely determined by $cRT(p)$, and vice versa.*
2. *$RT(p) \subseteq RT(q) \iff cRT(p) \subseteq cRT(q)$*
3. *$fRT(p) = \{v_0$ of finite depth $\mid \exists w_0 \in cRT(p).\, v_0 \leq w_0\}$*

Before stating the next lemma we introduce the following definitions that allow us to approximate ready trees:

Definition 13 (k–Ready Tree). A *k–tree* $\langle V, \longrightarrow, T, \emptyset\rangle$, where $k \in \mathbb{N}_0$, is an observation tree where all nodes have depth at most k, and T is the set of all nodes of depth k. A *k–ready tree* of p is a ready tree of p that is also a k–tree. Moreover, $k\text{–RT}(p) =_{\text{df}} \{v_0 \in \text{RT}(p) \mid v_0$ is a k–tree$\}$.

Intuitively, k–trees represent observations of k steps.

Definition 14 (Limit). Let $\boldsymbol{v}$ be an infinite sequence $(v_k)_{k\in\mathbb{N}}$ where $v_k \in k\text{–RT}(p)$ and $v_k \leq v_{k+1}$, with the identity as injection, for all $k \in \mathbb{N}$. Then, $\lim \boldsymbol{v}$ is the component–wise union of all v_k with T set to empty; $\lim \boldsymbol{v}$ is called a *limit* of p.

Observe that a node of some v_k in such a sequence is not in T_{k+1}, whence nodes in T are successively pushed out. In the limit, we may thus set T to empty. Moreover, if $v_k = v_{k+1} = v_{k+2} = \ldots$ for some k, then the limit is v_k; this happens exactly when v_k is complete. According to the following definition, we base the notion of finite branching on the weak transition relation $\stackrel{\alpha}{\Longrightarrow}$.

Definition 15 (Finite Branching). A process p is finite branching if, for every p' reachable from p, there are only finitely many $\langle \alpha, p''\rangle$ with $p' \stackrel{\alpha}{\Longrightarrow} p''$.

For finite–branching processes p, $\text{cRT}(p)$ is characterised by the limits of p.

Lemma 16. *If p is finite branching, $cRT(p)$ equals the set of all limits of p.*

Note that the premise "p is finite branching" is only needed for direction "$\supseteq$" in the above lemma. We may now obtain the following corollary of Cor. 12(3) and of Lemma 16, which is key to proving compositionality and full abstraction of our ready–tree preorder in the next section.

Corollary 17.

1. *$cRT(p) \subseteq cRT(q) \implies fRT(p) \subseteq fRT(q)$, always.*
2. *$cRT(p) \subseteq cRT(q) \impliedby fRT(p) \subseteq fRT(q)$, if p is finite branching.*

4 Compositionality & Full Abstraction

This section establishes our full–abstraction result of the ready–tree preorder $\sqsubseteq\!\!\!\!\sim$ with respect to the consistency testing preorder $\sqsubseteq$, and proves that $\wedge$ and $\vee$ are indeed conjunction and, respectively, disjunction for $\sqsubseteq\!\!\!\!\sim$. To do so, we first show that $\wedge$ and $\vee$ correspond to intersection and union on the semantic level, respectively. While the correspondence for $\vee$ holds for ready trees in general, the correspondence for $\wedge$ only holds for *complete* ready trees.

Theorem 18 (Set–Theoretic Interpretation of $\wedge$ and $\vee$).

1. $cRT(p \wedge q) = cRT(p) \cap cRT(q)$ *2.* $RT(p \vee q) = RT(p) \cup RT(q)$

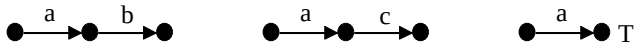

Fig. 6. Necessity of considering complete ready trees for conjunction.

Fig. 6 illustrates that Thm. 18(1) is invalid when considering all ready trees instead of complete ready trees. The two processes displayed on the left and in the middle have the ready tree displayed on the right in common. However, the conjunction of the two processes is false and has no ready trees. Intuitively, the shown common ready tree formalises an observation that finished too early to encounter the inconsistency. Compositionality of our conjunction and disjunction operators for $\sqsubseteq\!\!\!\!\sim$ is now an immediate consequence of Thm. 18:

Theorem 19 (Compositionality).

1. $p \sqsubseteq\!\!\!\!\sim q \implies p \wedge r \sqsubseteq\!\!\!\!\sim q \wedge r$ *2.* $p \sqsubseteq\!\!\!\!\sim q \implies p \vee r \sqsubseteq\!\!\!\!\sim q \vee r$

Thm. 18 also allows us to prove that $\wedge$ and $\vee$ really behave as conjunction and disjunction with respect to our refinement relation.

Theorem 20 ($\wedge$ is And & $\vee$ is Or).

1. $p \wedge q \sqsubseteq\!\!\!\!\sim r \iff p \sqsubseteq\!\!\!\!\sim r$ *and* $q \sqsubseteq\!\!\!\!\sim r$ *2.* $r \sqsubseteq\!\!\!\!\sim p \vee q \iff r \sqsubseteq\!\!\!\!\sim p$ *and* $r \sqsubseteq\!\!\!\!\sim q$

In order to see that ready trees are indeed fully–abstract with respect to our naive consistency preorder, it now suffices to prove that $\sqsubseteq\!\!\!\!\sim$ coincides with our consistency testing preorder. This means that $\sqsubseteq\!\!\!\!\sim$ is *the* adequate preorder in our setting of logical LTSs with conjunction and disjunction.

Theorem 21 (Full Abstraction). $\sqsubseteq = \sqsubseteq\!\!\!\!\sim$

The following proposition states the validity of several boolean properties desired of conjunction and disjunction operators. Here, $=$ denotes the kernel of our consistency testing preorder (ready–tree preorder).

Proposition 22 (Properties of $\wedge$ and $\vee$).

Commutativity:	$p \wedge q = q \wedge p$	$p \vee q = q \vee p$
Associativity:	$(p \wedge q) \wedge r = p \wedge (q \wedge r)$	$(p \vee q) \vee r = p \vee (q \vee r)$
Idempotence:	$p \wedge p = p$	$p \vee p = p$

False:	$p \wedge \mathit{ff} = \mathit{ff}$	$p \vee \mathit{ff} = p$
True:	$p \wedge \mathit{tt} = p$	$p \vee \mathit{tt} = \mathit{tt}$
Distributivity:	$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$	$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$

These properties follow directly from Thm. 18 and Cor. 12(2), as do the following:

Proposition 23 (Relating $\wedge$, $\vee$ to $\sqsubseteq$).

1. $p \wedge q = q \iff p \sqsubseteq q$ *2.* $p \vee q = p \iff p \sqsubseteq q$

We conclude this section by briefly returning to the illustrative processes *spec* and *impl* of Fig. 4. We have already remarked that the only complete ready tree of the latter is also one of the former. Hence, by Thm. 21, *impl* is indeed a refinement of *spec* according to our ready–tree preorder. Considering the conjunction of these processes, also shown in Fig. 4, it might be easier to see this using Prop. 23(1).

5 Related Work

Traditionally, process–algebraic and temporal–logic formalisms are not mixed but co–exist side by side. Indeed, the process–algebra school often uses synchronous composition and internal choice to model conjunction and disjunction, respectively. The compositionality of classic process–algebraic refinement preorders, such as failures semantics [4] and must–testing [12], enables component–based reasoning. However, inconsistencies in specifications are not captured so that, e.g., the conjunctive composition of a and b is identified with deadlock rather than ff. In contrast, the temporal–logic school distinguishes between deadlock and ff, but does not support component–based refinement.

Much research on mixing operational and logical styles of specification avoids dealing with inconsistencies by translating one style into the other. On the one hand, operational content may be translated into logic formulas, as is implicitly done in Lamport's TLA [10] or in the work of Graf and Sifakis [8]. In these approaches, logical implication serves as refinement relation. On the other hand, logical content may be translated into operational content. This is the case in automata–theoretic work, such as Kurshan's work on ω–automata [9], which includes synchronous and asynchronous composition operators and uses maximal trace inclusion as refinement relation. However, both logical implication and trace inclusion are insensitive to deadlock.

A seminal approach to compositional refinement relations in a mixed setting was proposed by Olderog in [13], where process–algebraic constructs are combined with trace formulas expressed in a predicate logic. In this approach, trace formulas can serve as processes, but not vice versa. Thus, freely mixing operational and logical styles is not supported and, in particular, conjunction cannot be applied to processes. For his setting, Olderog develops a denotational semantics that is a slight variation of standard failures semantics. Remarkably, an inconsistent formula is given a semantics that is not an element of the appropriate domain, as is stated on pp. 172–173 of [13].

Recently, a more general approach to combining process–algebraic and temporal–logic approaches was proposed in two papers by Cleaveland and Lüttgen [5, 6], which adopt a variant of De Nicola and Hennessy's must–testing preorder [12] as refinement preorder. However, Cleaveland and Lüttgen have not successfully solved the challenge of defining a semantics that is both deadlock–sensitive and compositional, and in which the conjunction operator and the refinement relation are compatible in the sense of Prop. 23(1). Our work solves this problem in the basic setting of logical LTS. Key for the solution is our new understanding of inconsistency, which is reflected by the fact that we consider processes a and $a + b$ as inconsistent, whereas they were treated as consistent in [6]. Observe that also in failure semantics and must–testing, a and $a + b$ are inconsistent in the sense that they do not have a common implementation.

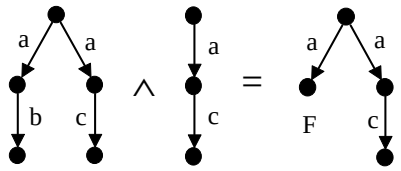

Fig. 7. Backward propagation of inconsistency.

In addition, our backward propagation of inconsistency, as formalised in Def. 1(3), is in line with traditional semantics, as is illustrated in Fig. 7. The first conjunct would be a specification of the second conjunct with respect to failures semantics and must–testing, whence their conjunction should be consistent. In fact, the conjunction equals the second process in our ready–tree semantics.

Comparing Ready–Tree Semantics to Other Semantics

To the best of our knowledge, ready–tree semantics is novel and has not been studied in the literature before. We thus briefly compare it to three popular semantics, namely *ready–trace semantics*, *failures semantics* and *ready simulation* [7]. Since our treatment of divergence is different from the one of failures semantics, we restrict our discussion to τ–free processes.

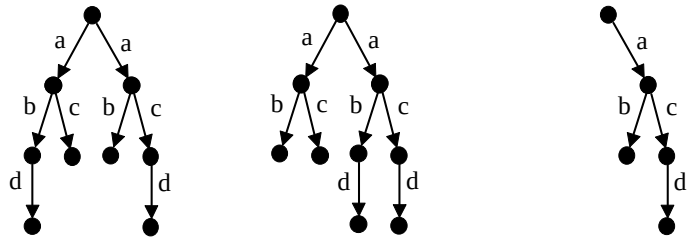

Fig. 8. Ready–tree semantics is strictly finer than ready–trace semantics.

A *ready trace* [1] of a process is a sequence of actions that it can perform and where, at the beginning of the trace, between any two actions and at the end,

the ready set of the process reached at the respective stage is inserted. Such a ready trace can be understood as a particular type of ready tree that consists only of a single path and includes additional transitions representing the ready sets. These additional transitions ensure that each state on the path has, for each action in its ready set, exactly one transition that either belongs to the path or ends in a *true*–state. For example, the first ready tree in Fig. 5 in Sec. 3 represents the ready trace $\{a, b\}b\{b\}b\{a, b\}$. Consequently, the ready traces of a process can be read off from its ready trees, and ready–tree inclusion implies ready–trace inclusion. The reverse implication does not hold: the two left–most processes in Fig. 8 possess the same ready traces; however, the observation tree on the right–hand side is a ready tree of the first, but not of the second process.

The *failures semantics* of a process is the set of its refusal pairs. Such a pair consists of a trace followed by a refusal set, i.e., a set of actions that the process reached by the trace cannot perform. Such a refusal pair can be read off from the respective ready trace by deleting all its ready sets and adding a set of actions having an empty intersection with the last ready set on the trace. Thus, ready–tree semantics is finer than failures semantics.

A process q *ready–simulates* some process p if there exists a simulation relation from p to q such that related states have identical ready sets. When tracing a ready tree of p, it is easy to see that such a simulation translates this ready tree to the same ready tree for q. Thus, the ready–tree preorder is coarser than ready simulation. Fig. 9 shows that it is indeed *strictly* coarser. Both processes displayed have the same ready trees; all of these trees are paths. However, the second process cannot even simulate the first process.

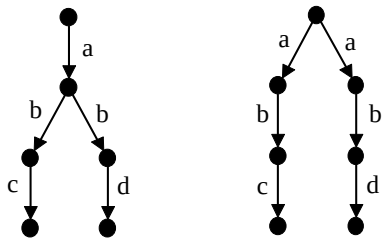

Fig. 9. The ready–tree preorder is strictly coarser than ready simulation.

6 Conclusions & Future Work

This paper introduced a new semantics, the *ready–tree semantics*, that lies between ready–trace semantics and ready simulation. Our framework was one of τ–pure LTSs, with distinguished *true*– and *false*–states, and is equipped with *conjunction* and *disjunction* operators. Key for defining the conjunction operator was the careful, inductive formalisation of an inconsistency predicate. The implied ready–tree preorder proved to be compositional and fully–abstract with respect to a naive preorder that allows inconsistent specifications to be refined only by inconsistent implementations. Standard laws of boolean algebra hold as expected, due to the fact that conjunction and disjunction on LTSs correspond to intersection and union on ready trees, respectively.

Consequently, this paper solves the problems of defining conjunction which are reported in closely related work [5, 6], albeit in a simpler setting that does

not consider process–algebraic operators but only conjunction and disjunction. However, it is the simplicity of our setting that brought the subtleties of defining a fully–abstract semantics in the presence of conjunction to light, and which offered a way forward in addressing the challenge of defining "logical" process calculi, i.e., process calculi that allow one to freely mix process–algebraic and temporal–logic operators [6].

Future work shall extend our results to richer frameworks. Firstly, we plan to lift our requirement of τ–purity on LTS and extend our framework by standard process–algebraic operators such as parallel composition, hiding and recursion. In particular hiding is likely to prove challenging due to its transformation of observable infinite behaviour into divergent behaviour. Secondly, our framework shall be semantically extended from LTS to Büchi LTS [5] so that one may express liveness and fairness properties, and syntactically to linear–time temporal–logic formulas [6]. Last, but not least, we wish to explore tool support.

Acknowledgements. We thank Rance Cleaveland for many fruitful discussions and particularly for suggesting the use of an inconsistency predicate.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Ready-trace semantics for concrete process algebra with the priority operator. *Computer J.*, 30(6):498–506, 1987.
[2] J.A. Bergstra, A. Ponse, and S.A. Smolka. *Handbook of Process Algebra*. Elsevier Science, 2001.
[3] B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.
[4] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.
[5] R. Cleaveland and G. Lüttgen. A semantic theory for heterogeneous system design. In *FSTTCS 2000*, vol. 1974 of *LNCS*, pp. 312–324. Springer-Verlag, 2000.
[6] R. Cleaveland and G. Lüttgen. A logical process calculus. In *EXPRESS 2002*, vol. 68,2 of *ENTCS*. Elsevier Science, 2002.
[7] R. van Glabbeek. The linear time – branching time spectrum II. In *CONCUR '93*, vol. 715 of *LNCS*, pp. 66–81. Springer-Verlag, 1993.
[8] S. Graf and J. Sifakis. A logic for the description of non-deterministic programs and their properties. *Information and Control*, 68(1–3):254–270, 1986.
[9] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton Univ. Press, 1994.
[10] L. Lamport. The temporal logic of actions. *TOPLAS*, 16(3):872–923, 1994.
[11] G. Lüttgen and W. Vogler. Conjunction on processes: Full-abstraction via ready-tree semantics. Tech. Rep. YCS-2005-396, Dept. of Comp. Sci., Univ. of York, UK, 2005.
[12] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *TCS*, 34:83–133, 1983.
[13] E.R. Olderog. *Nets, Terms and Formulas*. Cambridge Tracts in Theoretical Computer Science 23. Cambridge Univ. Press, 1991.
[14] A. Pnueli. The temporal logic of programs. In *FOCS '77*, pp. 46–57. IEEE Computer Society Press, 1977.