Congruent Weak Conformance, a Partial Order among Processes

Ronald W. Brower [1] and Kenneth S. Stevens [2]

[1] Air Force Research Laboratory, Embedded Information Systems Engineering Branch
Wright-Patterson Air Force Base, Ohio 45433-7334, USA
r.brower@ieee.org
[2] Intel Corporation, Strategic CAD Labs
Portland, Oregon, USA
k.stevens@ieee.org

Abstract. This paper presents a new property between processes arising from a set of relations called weak conformations. The largest, called weak conformance, is analogous to Milner's observational equivalence. Unlike observational equivalence, however, weak conformance is not an equivalence but rather a preorder between processes. Like the previous property of logic conformance, weak conformance allows behaviors in the implementation that are unreachable in the specification. Unlike logic conformance, however, weak conformance exploits output concurrencies and allows interleaving of extraneous output actions in the implementation. Finally, reasonable restrictions in CCS syntax strengthen weak conformance to a precongruence. The resulting property, congruent weak conformance, forms a partial ordering among processes. As a precongruence, it models safe substitution of hardware.

1 Introduction

When modeling the compliance of an implementation to a specification, the symmetry inherent in an equivalence is excessive—a fact well known to logisticians who attempt to replace discontinued parts. First of all, I/O pins in excess of those specified are certainly tolerable in the implementation. Secondly, unspecified behaviors in the implementation are also tolerable when they reside in the unreachable state space.

Furthermore, the specification may provide an *output concurrency* where multiple output events are produced without interleaving input, and the order of events is unimportant. Such concurrency is characterized by branching behavior that converges when all outputs have occurred. The implementation is free to realize just one path through an output concurrency. Exploiting such output options leads to more efficient designs. The same is not true for input concurrencies, where the implementation must remain poised to accept all input interleavings defined by the specification.

Examples of hardware equivalences abound [4, 5, 8, 10, 12, 13, 19-23, 26]. However, rather than a symmetric equivalence relation, an asymmetric preordering among processes is suggested for modeling compliance.

D.A. Peled and M.Y. Vardi (Eds.): FORTE 2002, LNCS 2529, pp. 34-49, 2002.

Examples of preorder relations also appear in the literature. The *efficiency* and *divergence* preorders create an ordering based on the amount of internal computation that processes perform [2, 3, 15]. The *faster-than* preorder associates worse-case execution times with actions, and then orders processes based on speed [17]. All three preorders create an ordering among observationally equivalent processes, but they are undefined defined between $\approx$ classes.

Other researchers note that a process's transition graph creates an ordering among its derivatives [1, 6, 7, 9]. If a transition $P \stackrel{s}{\Rightarrow} Q$ exists then $P \geq Q$. These *causal* or *derivational* preorders yield an order-of-occurrence, but do not capture the desired implementation-to-specification relationship, that should apply at instants of time.

The *quiescent* preorder compares only quiescent states—those that only accept inputs [24]. This preorder is undefined over the many intermediate states capable of output or internal action. All required outputs must be "yielded" by the implementation, so the quiescent preorder does not exploit output concurrency.

The *may* and *must* preorders are based on *testing* semantics (11:Chapter 2), and are defined by set containment. $P \leq_{\text{may}} Q$ if all the tests P may pass are contained in those Q may pass (and similarly for $\leq_{\text{must}}$). However, the testing semantics discipline is weaker than bisimulation [26] and therefore undesirable as a conformance preorder.

This paper introduces a new preorder, *congruent weak conformance*, that captures intuitive ideas governing the implementation-to-specification relationship. Congruent weak conformance is being used in research to link simulation-based hardware description languages such as VHDL [16] to process algebras such as the Calculus of Communication Systems (CCS) [19]. Such a link will allow stricter verifications of VHDL models based on the bisimulation semantics of CCS.

The development of congruent weak conformance resembles Milner's development of *observational congruence* [19:Chapters 5 and 7]. In this analogy, *weak bisimulations* are replaced with a set of *weak conformations*. Whereas the largest weak bisimulation is *observational equivalence* $\approx$, the largest weak conformation is called *weak conformance* and given the symbol $\succeq_{\text{w}}$. Observational equivalence is strengthened to a congruence by disallowing initial instability. Similarly, weak conformance is strengthened by the enforcement of five reasonable design constraints. The resulting property, *congruent weak conformance*, is symbolized as a double-underlined $\succeq_{\text{w}}$.

2 Notation

Process algebras such as *Communicating Sequential Processes* (CSP) [13, 14] and the *Calculus of Communicating Systems* (CCS) [19] are used to model concurrency and thus are useful for modeling digital electronic hardware. CCS is a very concise, but highly expressive process algebra. Especially important are its mechanisms for expressing non-determinism, choice and hidden internal action. CCS will serve as the basis for the discussion that follows.

Consider the *C-element*. The C-element awaits the receipt of two inputs. Once both have arrived, an output $\overline{c}$ is produced. The CCS model for the C-element is:

$$C \stackrel{def}{=} a.b.\bar{c}.C + b.a.\bar{c}.C \quad \textbf{(1)}$$

Inputs *a* and *b* are concurrent and may arrive in either order. This allowance is indicated by the use of two terms separated by '+'.

Processes (or agents) appear in upper case. Lower case names serve as transition labels. The overbar distinguishes inputs from outputs and is generally associated with output, though it can denote either as long as the usage is consistent.

CCS employs six combinators. Consistent with Milner's usage, combinator names are capitalized to distinguish them from their common English meanings:

- The *Constant* operator $\stackrel{def}{=}$ assigns an agent name to a behavior.
- *Prefix*, denoted by the period, indicates one action following another.
- *Choice* or *Summation*, denoted by +, indicates a fork in the execution path.
- *Parallel Composition*, denoted by ($A \mid B$), indicates agents A and B operating concurrently with a possibility of communication.
- *Relabeling,* denoted by $A[x/y]$, indicates that action x has been renamed to y in the agent A.
- *Restriction*, denoted by $A\backslash L$, represents agent A with all the actions in set L removed.

The Choice operator *can* express nondeterministic behavior. Consider the agent:

$$COIN \stackrel{def}{=} flip.\overline{heads}.COIN + flip.\overline{tails}.COIN \quad \textbf{(2)}$$

Here the environment can exert no control over the outcome, since the input *flip* occurs in both branches. Once a *flip* arrives, the *COIN* agent nondeterministically selects one branch, and produces an output accordingly. However the Choice expressed in the C-element specification is perfectly predictable due to the environment's ability to control the input sequence and select which branch is executed.

Now consider the behavior of a simple one-place buffer or FIFO:

$$FIFO \stackrel{def}{=} in.\overline{out}.FIFO \quad \textbf{(3)}$$

One can build a two-place FIFO by connecting two one-place FIFOs in series:

$$FIFO2 \stackrel{def}{=} (FIFO[mid/out] \| FIFO[mid/in]) \backslash \{mid\} \quad \textbf{(4)}$$

The Parallel Composition (*FIFO* | *FIFO*) builds a composite model from two (in this case identical) submodels. The Relabeling functions [*mid*/*out*] and [*mid*/*in*] create signals *mid* and $\overline{mid}$ which then form an implicit internal connection. Communication between submodels is accomplished implicitly when actions and co-actions share the same label. $\backslash\{mid\}$ hides this internal signal from the external environment. The hiding of internal signals, implicit in many languages, must be made explicit in CCS.

The compound agent *FIFO2* conducts a silent action by transferring a hidden datum from the first to the second *FIFO*. Such hidden or silent actions are denoted by the symbol τ. *All* silent actions are abstracted into this single symbol.

$\mathcal{A}$ is the set of all input symbols, or *names*. $\overline{\mathcal{A}}$ contains all output symbols, or *co-names*. $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ contains all visible labels, and $\mathcal{A}ct = \mathcal{L} \cup \{\tau\}$ adds τ. $\mathcal{A}(P)$,

$\overline{\mathcal{A}}(P)$, $\mathcal{L}(P)$ and $\mathcal{A}ct(P)$ are the corresponding *sorts* for agent P.

$\hat{s}$ is an *observable* trace created from $s \in \mathcal{A}ct^*$ by removing all embedded τ actions. $\hat{s}$ is the projection of s onto $\mathcal{L}$. The empty sequence or trace is denoted as ε.

$\xrightarrow{\alpha}$ denotes an agent's ability to perform some $\alpha \in \mathcal{A}ct$. If $P \xrightarrow{a} Q$ then P can receive input a and thereby evolve into Q. The symbol $\Rightarrow$ denotes the ability to perform sequences wherein any number of intermediate τ actions can be inserted to realize the transition. With the hat embellishment, $P \stackrel{\hat{s}}{\Rightarrow} Q$ denotes the complete abstracting away (ignoring) of τ actions. An unlabelled $\Rightarrow$ denotes a transition of zero or more τ actions. Arrows "pointing nowhere," such as $P \stackrel{\hat{s}}{\Rightarrow}$, merely indicate the ability to perform a given transition without the necessity of naming the target agent.

$\restriction$ denotes *projection* and normally applies to the projection of an action string onto a set. Thus $t \restriction \mathcal{A}(S)$ is the string t with all actions removed *except* those in $\mathcal{A}(S)$.

A new notation now denotes the additional input and output symbols or pins that an implementation has in excess of the specification. These are called *extraneous* pins. The expression $\overline{\mathcal{E}xtr}(I,S) = \overline{\mathcal{A}}(I) - \overline{\mathcal{A}}(S)$ denotes the *extraneous output sort* of I with respect to S. Similarly, $\mathcal{E}xtr(I,S) = \mathcal{A}(I) - \mathcal{A}(S)$ is the *extraneous input sort*.

3 Conformance Example

Consider a circuit that converts binary-coded-decimal (BCD) to pure decimal. It takes four bits to encode a decimal digit, so the converter will have four inputs, call them *a, b, c* and *d*. The ten outputs will be labeled $\bar{o}_0, \bar{o}_1, \ldots, \bar{o}_9$, one for each decimal digit detected. One can think of the outputs as ten indicator lights. In a CCS model of the specification, each time there is change on an input bit, one output light turns on and another is extinguished. Since CCS models transitions and not level signals, there will be two output transitions concurrently, but it will not be readily apparent which is turning on and which is turning off. Assume the parent system does not care if *momentarily* two are lit, or none. The specification will look like this:

$$\begin{aligned} S \stackrel{def}{=} S0 \stackrel{def}{=}\ & a.(\bar{o}_0.\bar{o}_1.S1 + \bar{o}_1.\bar{o}_0.S1) + b.(\bar{o}_0.\bar{o}_2.S2 + \bar{o}_2.\bar{o}_0.S2) \\ & + c.(\bar{o}_0.\bar{o}_4.S4 + \bar{o}_4.\bar{o}_0.S4) + d.(\bar{o}_0.\bar{o}_8.S8 + \bar{o}_8.\bar{o}_0.S8) \end{aligned} \quad \textbf{(5)}$$

with similar definitions for states *S1* to *S9*. A shorthand notation $(\bar{o}_0 \mid \bar{o}_1)$ can now be used to express the concurrency of output signals while economizing on the code.

$$\begin{aligned} S0 &\stackrel{def}{=} a.(\bar{o}_0 \mid \bar{o}_1).S1 + b.(\bar{o}_0 \mid \bar{o}_2).S2 + c.(\bar{o}_0 \mid \bar{o}_4).S4 + d.(\bar{o}_0 \mid \bar{o}_8).S8 \\ S1 &\stackrel{def}{=} a.(\bar{o}_1 \mid \bar{o}_0).S0 + b.(\bar{o}_1 \mid \bar{o}_3).S3 + c.(\bar{o}_1 \mid \bar{o}_5).S5 + d.(\bar{o}_1 \mid \bar{o}_9).S9 \\ &\ldots \\ S9 &\stackrel{def}{=} a.(\bar{o}_9 \mid \bar{o}_8).S8 + d.(\bar{o}_9 \mid \bar{o}_1).S1 \end{aligned} \quad (6)$$

Only states *S0* and *S1* respond to all four inputs because combinations above '1001' are illegal. Omitting these transitions in *S2* to *S9* constitutes the specification's guarantee that the illegal input combinations will not be received.

Given specification *S*, what sort of circuit would make a conforming implementation? A 4:16 demultiplexor, or "demux," is an obvious choice. The inputs *a, b, c*, and *d* control the four select lines. Only ten of the outputs are used, with six unconnected. A fifth input pin represents the multiplexed input. In this application that pin is tied to '1'. Note therefore that a compliant implementation must have a pin for every I/O pin called out by the specification, though it may have more.

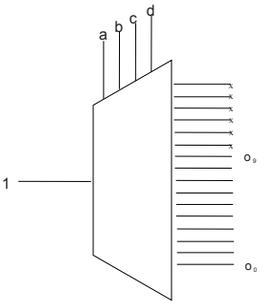

Fig 1. 4:16 Demultiplexor

A "first cut" CCS model for this demux could read just like the specification model but with the missing input transitions added and the extra outputs generated.

$$I \stackrel{def}{=} I0 \stackrel{def}{=} a.(\bar{o}_0|\bar{o}_1).I1 + b.(\bar{o}_0\,\bar{o}_2).I2 + c.(\bar{o}_0|\bar{o}_4).I4 + d.(\bar{o}_0|\bar{o}_8).I8$$

…

$$I15 \stackrel{def}{=} \ldots \tag{7}$$

Unlike the specification, the implementation has sixteen named states. Each responds to all four inputs. Thus, *I* can execute the illegal sequences. Indeed, this is allowable since the specification guarantees that these additional states are unreachable. One might hastily conclude that implementations must duplicate *all* the states of the specification, with additional states allowed. Yet this is not the case. Though the implementation *I* gratuitously generates *all* the possible output interleavings allowed by *S*, in reality it would be both difficult and counterproductive to create such a device. A real, physical layout results in finite delays along various paths. Most likely, the same interleaving appears every time in a physical implementation. Hence, a more accurate and realistic rendering of the implementation *I* would have output concurrencies such as ($\bar{o}_0 \mid \bar{o}_1$) replaced with specific interleavings such as $\bar{o}_1.\bar{o}_0$.

4 Previous Conformances

Conformances are asymmetric relations with the *specification* agent appearing on the right, and the *implementation* on the left. Conformances treat inputs and outputs distinctly. Hence, for conformance relations, the overbar is identified specifically with output. This departs from previous usage, where the association is arbitrary.

A number of conformances have been developed, including *logic conformance*.

Definition 1 [25: Definition 30]. Implementation *I logically conforms* to specification *S*, written $I \succeq_l S$, iff $\forall \alpha \in \mathcal{A}ct, \forall \beta \in \overline{\mathcal{A}} \cup \{\tau\}$ and $\forall \gamma \in \mathcal{A}$:

(1) Whenever $S \xrightarrow{\alpha} S'$ then $\exists I' : I \stackrel{\hat{\alpha}}{\Rightarrow} I'$ and $I' \succeq_l S'$.

(2) Whenever $I \xrightarrow{\beta} I'$ then $\exists S' : S \stackrel{\hat{\beta}}{\Rightarrow} S'$ and $I' \succeq_l S'$.

(3) Whenever $I \xrightarrow{\gamma} I'$ and $S \stackrel{\gamma}{\Rightarrow}$ then $\exists S'$ such that $S \stackrel{\gamma}{\Rightarrow} S'$ and $I' \succeq_l S'$.

Part (1) demands that all specified behaviors be implemented. Part (2) assures that every implemented output corresponds to a specified output event. In part (3), the premise $S \stackrel{\gamma}{\Rightarrow}$ allows the implementation to accept unspecified inputs.

Logic conformance respects preemptive tau actions and abstracts away all others. It is sensitive to the branching structure of agents, detects deadlock, and requires that deadlocks in the implementation must match deadlocks in the specification.

Logic conformance has shortcomings that need remedy. First of all, part (1) is overly restrictive with respect to outputs. *I* must implement *every* specified output action, even when there is output concurrency. Secondly, part (2) makes no allowance for the implementation to generate output signals outside of the specification.

5 Local Confluence and Maxoctsets

The weak conformance property uses the notion of *confluence*, a restricted form of determinism. There are both *strong* and *weak* versions of confluence, with weak being the more interesting. One of Milner's results serves as a working definition.

Definition 2 [19:Proposition 11.11]. *If P is weakly confluent then whenever* $P \stackrel{r}{\Rightarrow} P'$ *and* $P \stackrel{s}{\Rightarrow} P''$ *then* $P' \stackrel{s/r}{\Rightarrow} Q'$ *and* $P'' \stackrel{s/r}{\Rightarrow} Q''$ *with* $Q' \approx Q''$.

r/*s* denotes "excess of *r* over *s*" [19: Definition 11.6], being the string *r* with the symbols it shares with *s* removed. This removal occurs from left to right and takes note of the multiplicity of symbols within *s*. If the symbol *a* appears twice within *s* then no more than two occurrences of *a* are removed from *r*. As examples, observe that *a.b.c/a.c = b, a.b.a.b/a = b.a.b,* and *a.b.c/a.a = b.c.*

For the confluent agent *P* of Definition 2, all traces arrive at the same (up to ≈) destination. Whatever the path, the same visible actions are encountered the same number of times, albeit the *order* of the actions may be different. No visible action is preempted from occurring its appointed number of times. Strings *r.s/r* and *s.r/s* have the same net effect. Such strings, which are equivalent up to permutation, are *con-*

fluence equivalent, written $x =_{\text{conf}} y$. This leads to the new concept of *local confluence*.

Definition 3. Let $s \in \mathcal{L}^*$. Agent P is *locally confluent* with respect to s if $P \overset{s}{\Rightarrow}$ and $\forall r =_{\text{conf}} s$, whenever $P \overset{s}{\Rightarrow} P'$ and $P \overset{r}{\Rightarrow} P''$ then $P' \approx P''$.

All the $=_{\text{conf}}$ sequences terminate in the same state up to $\approx$. One may write $P \overset{r}{\Rightarrow} \approx P'$ for all such r. Note that P is locally confluent with respect to *any* r. Whereas Milner's confluence is a global property that insists that all exiting sequences preserve the confluence, local confluence is content if a portion of the transition graph resembles the confluence diagram. Transitions that destroy global confluence are ignored. Note also that not *all* permutations of s need be present for local confluence.

The area of local confluence can be embodied as a set of traces:

Definition 4. If P is locally confluent with respect to s then $X = \{r =_{\text{conf}} s : P \overset{r}{\Rightarrow}\}$ is the *confluent trace set* (*CT set*) of P with respect to s. Each member sequence in X is said to be a *defining sequence,* a *defining trace* or a *defining string* for X.

A CT set composed strictly of outputs is the more interesting, because only output concurrencies can be exploited. Hence:

Definition 5. Let X be a CT set of P with respect to s. X is an *output confluent transition set* (*octset*) of P with respect to s if $s \in \overline{\mathcal{A}}^+$.

Octset sequences are of non-zero length. A trivial octset $\{\varepsilon\}$ is disallowed. In fact, since only one of the sequences needs to be implemented, the desire is to have large, lengthy octsets to give the greatest design flexibility. Hence the *maxoctset*:

Definition 6. Let X be an octset of P with respect to s. X is a *maxoctset* of P if $\nexists t \in \overline{\mathcal{A}}(P)^+$ such that P has an octset with respect to st.

Every state with at least one exiting output transition has *some* maxoctset. In the extreme case, when no flexibility in design is offered, a maxoctset is a singleton set containing a lone output string that *must* be implemented.

6 Weak Conformations

A family of coinductively defined properties called *weak conformations* is now introduced that embody previously intuitive notions of compliance. Of these, *weak conformance* will be the largest. Being precursors to a conformance, weak conformations are also asymmetric relations with the specification agent on the right and the implementation on the left.

Definition 7. Binary process relation $\mathcal{W}$ is a *weak conformation* if $\forall \alpha \in \mathcal{A}(S) \cup \{\tau\}$, $\forall \beta \in \overline{\mathcal{A}}(I) \cup \{\tau\}$, $\forall \gamma \in \mathcal{A}(S)$: $I \mathcal{W} S$ *implies* the following four laws:

Law of Specified Input or Tau (LSIT). If $S \xrightarrow{\alpha} S'$ then $\exists t \in (\mathcal{A}(S) \cup \overline{\mathcal{E}xtr}(I,S))^*$ such that

(1) $I \stackrel{t}{\Rightarrow} I'$ (2) $t \upharpoonright \mathcal{A}(S) = \hat{\alpha}$ (3) $I' \mathcal{W} S'$

Law of Specified Output (LSO). Let X be a maxoctset of S. $\exists s \in X$ and $\exists t \in \overline{\mathcal{A}}(I)^+$ such that

(1) $S \stackrel{s}{\Rightarrow} S'$ (2) $I \stackrel{t}{\Rightarrow} I'$ (3) $t \upharpoonright \overline{\mathcal{A}}(S) = s$ (4) $I' \mathcal{W} S'$

Law of Implemented Input (LII). Whenever $I \xrightarrow{\gamma} I'$ and $S \stackrel{\gamma}{\Rightarrow}$ then

(1) $S \stackrel{\gamma}{\Rightarrow} S'$ (2) $I' \mathcal{W} S'$

Law of Implemented Output or Tau (LIOT). If $I \xrightarrow{\beta} I'$ and $\delta \equiv \beta \upharpoonright \overline{\mathcal{A}}(S)$ then

(1) $S \stackrel{\delta}{\Rightarrow} S'$ (2) $I' \mathcal{W} S'$

LSIT describes the obligation of the implementation when the specification requires an input or τ. The implementation answers by performing a string t. Both agents then evolve to derivatives I' and S' that also share the relation $\mathcal{W}$. t contains *one* occurrence of an input when α is visible and *none* when $\alpha = \tau$. The remainder of t contains extraneous outputs that can occur without harm because they are unknown to the specification. Other than a lone $\hat{\alpha}$, t can contain no other inputs. Even those inputs in $\mathcal{E}xtr(I,S)$ are prohibited. If t did contain such unspecified input actions, the implementation might wait forever on those inputs, and would thus be blocked.

LSO describes how an implementation answers specified output activity. Considering that a maxoctset may be minimal, containing a single output, then clearly *all* specified outputs fall within the scope of *some* maxoctset. When a maxoctset contains many sequences, the implementation need only "match" one of them (some $s \in X$). The implementation answers with t. String t contains all the actions of s, and in the same sequence. As before, t can further incorporate any number of extraneous outputs without harm. One will often say that *t implements s,* or alternately, that *t implements X*, since s is the representative of the entire maxoctset X.

Typically, a specified maxoctset consists of all possible interleavings of the output actions it governs. However, it is important to note that LSO makes no such assumption of "completeness." "Sparse" maxoctsets are allowed where specific interleavings are conspicuously absent, *i.e.*, not allowed by the specification. LSO directs the implementation of explicit member strings only—*not* just any permutation.

LII is a reuse of Definition 1(2). It addresses the care that must be taken when the implementation performs an input action within the specification sort. If the specification is not *immediately* capable of such action, there is no harm done because that action will not be forthcoming anyway. If the specification *is* immediately capable ($S \stackrel{\gamma}{\Rightarrow}$) then of course the implementation must match that action in accordance with LSIT. However, LII goes beyond that to state that I is prohibited from any other use of specified input symbols, except those arising by LSIT. Otherwise, the implementation could stray into illegal behavior.

LIOT addresses the occurrence of taus and specified outputs in the implementation. Although the implementation can freely engage in *extraneous* outputs, LIOT requires that any use of *specified* output symbols in the implementation be limited to those that arise by legal application of LSO. Again, this prevents the implementation from straying into illegal behavior.

The following results can be readily demonstrated:

Proposition 1. *The process identity relation* Idp *is a weak conformation.*
Proposition 2. *The union of weak conformations is a weak conformation.*

Now Milner [19] developed *observational equivalence* $\approx$ as the largest of the *weak bisimulations*, and then strengthened $\approx$ to a congruence by requiring initially stable agents. Since weak conformations are based on bisimulation semantics, we seek to do the same, *i.e.*, to identify the largest weak conformation and strengthen it to a congruence over CCS. The necessary proofs make frequent use of the composition $\circ$ of weak conformations, relying on such compositions to also be weak conformations.

The proof that $\circ$ preserves weak conformation requires that each of the weak conformation laws be preserved. In fact, the proof fails for unrestricted weak conformations. Consider LII, for example. LII cannot be proven to hold across the composition $P \mathcal{V} Q \mathcal{W} R$ when $R \overset{\gamma}{\Rightarrow}$ because Q has no obligation to perform an immediate $\overset{\gamma}{\Rightarrow}$. This happens because LSO permits Q to perform extraneous outputs prior to its first γ. From the perspective of specification R, such outputs γ are just as spontaneous and uncontrollable as τ actions, and can yield instabilities. For example, an output $\overline{x}$ extraneous to both A and B creates an instability in agents of the form $\overline{x}.A + B$. There can be *no* weak conformation between $\overline{x}.A + B$ and $A + B$. Hence an otherwise stable implementation can become *relatively unstable* with respect to its specification when an extraneous output plays the role of τ. The symbol τ_S now denotes such actions where the subscript S is the specification agent's name. Thus the *relative tau*, τ_S, admits both literal τ actions as well as output actions beyond the sort of S. One says that P is *relatively stable* with respect to Q if P has no τ_Q derivatives.

Definition 8. Weak conformation $I \mathcal{W} S$ meets the *conformational stability* (*CS*) *assumption* if S is stable and I is relatively stable with respect to S.

Proposition 4. *Relational composition* $\circ$ *preserves weak conformation under the CS assumption.*

The proof is to show that each law holds between P and R in the composition $P \mathcal{V} Q \mathcal{W} R$. The proof for LII is the most challenging, and is given here:

Proof: Consider the composition $P \mathcal{V} Q \mathcal{W} R$. One must establish $\forall \gamma \in \mathcal{A}(R)$ that whenever $P \overset{\gamma}{\rightarrow} P'$ and $R \overset{\gamma}{\Rightarrow}$ then $R \overset{\gamma}{\Rightarrow} R'$ with $P' \mathcal{V} \mathcal{W} R'$.

- Since Q is relatively stable with respect to R, LSI requires that $Q \overset{\gamma}{\Rightarrow}$ immediately.
- $\therefore Q \overset{\gamma}{\Rightarrow} Q'$ with $P' \mathcal{V} Q'$ by LII.
- In turn, $R \overset{\gamma}{\Rightarrow} R'$ with $Q' \mathcal{W} R'$ by LII on γ (with LIOT applied on any taus.
- Hence $P' \mathcal{V} \mathcal{W} R'$. □

7 Weak Conformance

Just as $\approx$ is the largest *weak bisimulation*, *weak conformance* is the largest weak conformation.

Definition 9. The *weak conformance* relation, written $\succeq_w$, is the union of all weak conformations.

The following propositions are easily demonstrated. The proof of the last is given.

Proposition 5. $\succeq_w$ *is a weak conformation.*
Proposition 6. $\succeq_w$ *is the largest weak conformation.*
Proposition 7. $\succeq_w$ *is reflexive.*
Proposition 8. $\succeq_w$ *is transitive under the CS assumption.*

Proof of Proposition 8: Given that $P \succeq_w Q \succeq_w R$, $P \succeq_w R$ follows since the composition $\succeq_w \succeq_w$ is a weak conformation and $\therefore \succeq_w \succeq_w \subseteq \succeq_w$. □

8 Congruent Weak Conformance

Weak conformance $\succ\!\!\_\!\!_w$, like $\approx$, is not a congruence over the unmodified CCS. A slightly finer conformation is needed. Thus a *congruent* weak conformance, to be symbolized as $\succeqq_w$, is desired.

One difficulty in achieving congruence has already been discussed: that of initial instability. This issue is addressed by the CS assumption, which has already been invoked to preserve weak conformation under $\circ$. An additional difficulty comes from the possibility of extraneous actions being "promoted" to specified actions during the construction of compound agents. This is not an issue for equivalences, where there are no extraneous actions. To achieve a congruent weak conformance over CCS constructions, one must prevent extraneous actions from being promoted during the course of the construction. Extraneous actions *prior to* the construction must remain extraneous *after* the construction, unless they disappear entirely from the sort of the composite implementation.

The reason for this becomes clear when one considers these agents to represent hardware. Extraneous actions represent pins on the implementation device that remain unconnected. Except for possible instability, the designer implicitly states that he does not care what activity occurs at these pins. If, when inserting this device into a higher design, he then proceeds to wire these pins to another device, then he must *indeed* care what happens at these device pins. To say that behavior is "don't care" at a lower level, and "do care" at a higher level, is a case of dubious design intent!

All the CCS constructors (except Restriction) can create problems by unwittingly promoting extraneous actions. Therefore, to obtain a congruent weak conformance relation, CCS constructions need to be constrained such that promotion of extraneous actions does not occur.

Definition 10. Let $\widetilde{I}\ \mathcal{W}\ \widetilde{S}$ be a system of weak conformations under $\mathcal{W}$, and let $E\{\widetilde{X}\}$ be a CCS expression on multiple agents denoted by indexed variable $\widetilde{X}$. $E\{\widetilde{X}\}$ can employ *all* constructors *except* Restriction and Relabeling. $E\{\widetilde{X}\}$ meets the *Preservation of Extraneous Action* (*PEA*) condition with respect to $\widetilde{I}\ \mathcal{W}\ \widetilde{S}$ if:

(1) $\mathcal{E}xtr(I_i, S_i) \subseteq \mathcal{E}xtr(E\{\widetilde{I}\}, E\{\widetilde{S}\})$ for all indices *i*.

(2) $\overline{\mathcal{E}xtr}\ (I_i, S_i) \subseteq \overline{\mathcal{E}xtr}\ (E\{\widetilde{I}\}, E\{\widetilde{S}\})$ for all *i*.

PEA guarantees that all extraneous actions remain extraneous after the construction $E\{\widetilde{X}\}$ so that unreachable paths are not inadvertently activated.

For the same reasons, *co-actions* should not be introduced that attempt to synchronize with extraneous actions, for such synchronization can activate unreachable paths via the silent action τ. This requirement applies to Parallel Composition, which introduces such synchronizations.

Definition 11. Under the same assumptions as Definition 10, if for all parallel compositions $(E_1\{\widetilde{X}\}|E_2\{\widetilde{X}\})$ within $E\{\widetilde{X}\}$:

(1) $\forall a \in \mathcal{L}(E_1\{\widetilde{S}\})$:
$\overline{a} \notin \overline{\mathcal{E}xtr}\ (E_2\{\widetilde{I}\}, E_2\{\widetilde{S}\}) \cup \mathcal{E}xtr(E_2\{\widetilde{I}\}, E_2\{\widetilde{S}\})$

(2) $\forall a \in \mathcal{L}(E_2\{\widetilde{S}\})$:
$\overline{a} \notin \overline{\mathcal{E}xtr}\ (E_1\{\widetilde{I}\}, E_1\{\widetilde{S}\}) \cup \mathcal{E}xtr(E_1\{\widetilde{I}\}, E_1\{\widetilde{S}\}))$

then $E\{\widetilde{X}\}$ meets the *Extraneous Synchronization Prohibition* (*ESP*) with respect to $\widetilde{I}\ \mathcal{W}\ \widetilde{S}$.

One can invoke these conditions to demonstrate the following for all weak conformations $\mathcal{W}$ in general:

Proposition 9. *$\mathcal{W}$ is preserved by Prefix under PEA.*
Proposition 10. *$\mathcal{W}$ is preserved by Choice under PEA.*
Proposition 11. *$\mathcal{W}$ is preserved by Parallel Composition under PEA and ESP.*

The proof of Proposition 9 is relatively easy. Propositions 10 and 11 have similar proofs. The proof of Proposition 10 relies on the following lemma:

Lemma. *Let M be a maxoctset of R = S + T. Let $M = M_S \cup M_T$ where $M_S = \{s \in M : S \overset{s}{\Rightarrow}\}$* and *$M_T = \{s \in M : T \overset{s}{\Rightarrow}\}$. M_S and M_T* are *maxoctsets of S and T, respectively.*

The proof for the lemma is a straightforward proof by contradiction If either M_S or M_T are not maxoctsets, then *M* can be shown not to be a maxoctset. Using the lemma, one now proves Proposition 10:

Proof of Proposition 10: Given $I\ \mathcal{W}$S and $J\ \mathcal{W}$T, show that $(I + J)\ \mathcal{W}$(S + *T*). Assume w.l.o.g. that any transition out of $S + T$ has *S* as its source.

- LSIT. Let $S + T \overset{\alpha}{\Rightarrow} S'$ for $\alpha \in \mathcal{A} \cup \{\tau\}$.
  By LSIT, $I \overset{t}{\Rightarrow} I'\ \mathcal{W}\ S'$.

Yet if $I \stackrel{t}{\Rightarrow} I'$ then it follows that $I + J \stackrel{t}{\Rightarrow} I'$.

To show that $t \upharpoonright \mathcal{A}(S + T) = \hat{\alpha}$, assume otherwise:

$\exists\, \overline{a} \in t$ such that $\overline{a} \in \overline{\mathcal{A}}(T)$ while $\overline{a} \notin \overline{\mathcal{A}}(S)$.

This violates PEA.

$\Rightarrow\Leftarrow$

- *LII* and *LIOT*. Similar to LSIT.
- *LSO*. Let M be a maxoctset of $S + T$. By the above lemma, $M = M_S \cup M_T$ where M_S and M_T are maxoctsets of S and T, respectively. As maxoctsets, M_S and M_T each contain at least one string implemented by I and J, respectively.

  If $s \in M_S$ is implemented by I, then $I \stackrel{t}{\Rightarrow} I' \mathcal{W} S'$.

  It follows that $I + J \stackrel{t}{\Rightarrow} I'$ for $S + T \stackrel{s}{\Rightarrow} S'$.

  Thus $I + J$ implements $s \in M_S \subseteq M$.

  If $s \in M_T$ the argument is similar. □

Now consider Restriction. For inputs, one can safely restrict *specified inputs*, since their removal does not affect the ability of the remaining inputs to conform to LSIT and LII. *Extraneous* inputs can also be safely restricted. Naturally, one can also restrict input symbols *external* to *both* the specification and implementation, since such Restriction does not modify the base agents at all. In conclusion: there are *no* limitations on Restriction of *inputs*.

For *outputs*, note that *specified* outputs can be restricted. In the case of LSO, restricting a single specified output action $\overline{a}$ will remove from consideration every maxoctset in which $\overline{a}$ participates, since $\overline{a}$ must appear in every string of that maxoctset. As with inputs, the Restriction of outputs *external* to *both* sorts is moot. However, one must be careful when restricting *extraneous* outputs, for they can appear within implementing strings, and their Restriction will block these strings. The only extraneous output actions that can safely be restricted are those whose *only* occurrences lie along unreachable paths. This special type of extraneous output is called an *idle* output action, and we define:

Definition 12. $\overline{Idle}(I \mathcal{W} S) \equiv \{ \overline{a} : \overline{a}$ is an idle output action of $I \mathcal{W} S. \}$

One can now define the condition required to achieve congruence for weak conformations under Restriction.

Definition 13. The label set $L \subseteq \mathcal{L}$ meets the *Congruent Output Restriction* (*COR*) condition with respect to some weak conformation $I \mathcal{W} S$ if $\forall a \in L : a, \overline{a} \notin \overline{Extr}(I,S) - \overline{Idle}(I \mathcal{W} S)$.

COR forbids the Restriction of extraneous outputs, unless they are idle. Thus three kinds of outputs *may* be safely restricted: (1) *specified* outputs, (2) *idle* outputs and (3) outputs *external* to both $\overline{\mathcal{A}}(I)$ and $\overline{\mathcal{A}}(S)$, whose Restriction is moot.

Proposition 12. *Restriction preserves a weak conformation when COR applies.*

The proof is accomplished by showing that the relation $S_C \equiv \{ (P\backslash\{c\}, Q\backslash\{c\}) : P \mathcal{W} Q \}$ is a weak conformation.

Now consider Relabeling. Relabeling can cause congruence failure if the Relabeling function assigns the same name to previously distinct symbols. Therefore, the Relabeling function must be an *injection.* Also, those signals not explicitly renamed are implicitly renamed to their "old" names, which of course must not collide with any "new" names, so the Relabeling function must in fact be a *bijection.*

Definition 14. A Relabeling operation meets the *Bijective Relabeling* (*BR*) condition if its function is explicitly or implicitly bijective.

Proposition 13. *Relabeling preserves weak conformation under the BR condition*.

The proof is to show that the process relation $\mathcal{S} \equiv \{ (P[f], Q[f] : P \mathcal{W} Q \}$ is a weak conformation.

It remains to show that weak conformance $\succeq_w$ is preserved under recursive definition. This cannot be done for general weak conformations, or even for $\succeq_w$ itself. It *can* however be demonstrated for $\succeq_w$ when CS, PEA, ESP, COR and BR apply.

Proposition 14. *Let $E \succeq_w F$ where expressions E and F contain at most the single variable X. Under the CS, PEA, ESP, COR and BR conditions, whenever $I \stackrel{def}{=} E\{I/X\}$ and $S \stackrel{def}{=} F\{S/X\}$ then $I \succeq_w S$.*

Proposition 14 is the statement that $\succeq_w$ is preserved by recursive definition under five constructional conditions. Its proof is analogous to Milner's proof for strong congruence [19:99-101]. One first defines $\mathcal{R} \equiv \{ (G\{I/X\}, G\{S/X\}) : I \stackrel{def}{=} E\{I/X\}, S \stackrel{def}{=} F\{S/X\} \}$ and then shows that $\mathcal{R}$ is a "weak conformation up to $\succeq_w$" (analogous to "bisimulation up to $\approx$"). The proof then proceeds as a transition induction by cases on the form of $G\{I/X\}$. The conditions CS, PEA, ESP, COR and BR must be invoked at critical points to complete the induction.

The discussion has arrived at the point where congruent weak conformance can be defined. Briefly summarizing: a set of properties called *weak conformations* were established under four transitional laws that embody intuitive notions of hardware compliance. Of the *weak conformations*, the largest was designated *weak conformance* and given the symbol $\succeq_w$. To model safe substitution of hardware, $\succeq_w$ had to be shown to be a *congruence*, *i.e.*, that it be preserved by all CCS operators.

To achieve the congruence goal, it was necessary to limit CCS constructions by the five conditions: CS, PEA, ESP, COR and BR. Happily, these limitations are not unworkable. Rather, they are consistent with good design sense, and have a simple interpretation, *to wit* that a "don't care" for the component must be a "don't care" for the system," or, even more simply:

"Don't cares" must be preserved by hierarchy.

The next definition fully integrates this precept into the $\succeq_w$ property. Future tools that verify for $\succeq_w$ will provide an automatic check that this design guideline is obeyed.

Definition 15. Let $I \succeq_w S$ under the CS, PEA, ESP, COR and BR conditions. Then *I* and *S* enjoy the *congruent weak conformance* relation, written $I \succeq_w S$.

These results follow:

Proposition 15. *If* $P \succeq_w Q$ *and* $Q \succeq_w P$ *then* $P = Q$.

Proof: Since $\succeq_w$ applies bidirectionally, no extraneous actions are possible and it is easy to show from the conformation laws that P and Q simulate each other, in other words $P \approx Q$. Observing that both P and Q are stable under the CS assumption, one concludes that $P = Q$.

Proposition 16. $\succeq_w$ *is a partial order.*

Proof: $\succeq_w$ inherits reflexivity and transitivity from $\succ_w$. Proposition 15 demonstrates antisymmetry.

Proposition 17. $\succeq_w$ *is a congruence.*

Proof: Propositions 9 through 14.

Congruence of $\succeq_w$ is a major result, establishing it as a correct model for safe substitution. Since $\succeq_w$ is in fact a preorder, it is appropriate and consistent with prior usage to call it a *precongruence*.

9 Conclusion

We have developed and proven a formal relationship between agents and specifications to be a congruent partial order. We feel this is the best formal relation so far for verifying implementations against specifications. This congruence derives maximal flexibility and embodies all weaknesses in input, output, and no-connect signals while retaining a fully replaceable conformance to the specification. This desirable relation is described in four transitional laws with five constructional restrictions.

10 Recommendations

The development of congruent weak conformance presumes *semantic* sorts. Whereas the *syntactic* sort of an agent easily derives from its lexical representation, the *semantic* sort must discard any symbols that are unreachable. Deriving a semantic sort is, in fact, undecidable, being akin to the halting problem. Therefore, automated tools to calculate congruent weak conformance will be limited to syntactic sorts. For the "syntactic" $\succeq_w$, unreachable specification actions will constrain the implementation to the four transitional rules in regions that are semantically "don't care." Semantically-extraneous actions detected as specified will lose their freedom to interleave or to trigger unsafe behavior. Unreachable implementation actions that are reachable in the specification will fail LSO and LSIT, as they would anyway. Falsely non-extraneous actions will *not* have an impact on the five construction restrictions, because these actions are also falsely adhere to the transitional laws. Therefore, the syntactic tools

will detect a slightly stronger $\succeq_w$. Such tools will be analogous to equivalence checkers, *i.e.*, *conformance* checkers. Creating such tools is a longer-term goal for us.

We also foresee tools that transform models from design languages such as VHDL to process algebras such as CCS—such that the greater verification powers of CCS will accrue to VHDL. Of course, the event-based simulation semantics of VHDL do not match the bisimulation semantics of CCS, and such a transformed model can never be "equivalent" to its VHDL original. However, the property of congruent weak conformance *between* models ought to be preserved in the course of the translation. Thus we recommend, when developing such tools, that each transformation be formally validated by formal proof that it preserves congruent weak conformance. Our current research is using $\succeq_w$ to establish such a translation from VHDL to CCS.

Finally, we claim that $\succeq_w$ "is the best formal relation *so far*…" Possibly, $\succeq_w$ is stronger than it need be. Continuing the analogy with Milner's work, note that $\approx$ with stability *implies* =, but does not *define* it [19:Proposition 7.10]. In fact, Milner's Definition 7.2 *allows* initial taus as long as the other agent matches them. Thus, had Milner allowed no tau derivatives at all, = would be stronger than necessary.

In the present development, the five constructional restrictions are analogous to stability, and $\succeq_w$ plus these restrictions *defines* (not *implies*) $\succeq_w$. A definition of $\succeq_w$ analogous to Milner's definition of =, where the derivatives share $\succeq_w$ and not $\succeq_w$, is suggested. However, such a formulation will be more complex. First of all, conformance treats outputs distinctly from inputs, whereas bisimulation does not. Secondly, there are four conformational laws to respect, and they have distinct form. There are just two bisimulation laws, and they are symmetric. Achieving such a formulation for $\succeq_w$ will be the subject of future research.

References

1. Alur, R., Brayton, R.K., Henzinger, T.A., Qadeer, S, Rajamani, S.K.: Partial Order Reduction in Symbolic State Space Exploration. In: Proceedings of the Ninth International Conference on Computer-aided Verification (CAV 1997), Lecture Notes in Computer Science, Vol. 1254, Berlin Heidelberg New York (1997) 340-351
2. Arun-Kumar, S., Hennessy. M.: An Efficiency Preorder for Processes. Acta Informatica, Vol. 29 (1992) 737-76
3. Arun-Kumar, S., Natajaran, V.: Conformance: A Precongruence Close to Bisimilarity. In: Desel, J. (ed.): International Workshop on Structures in Concurrency Theory (STRICT '95). Workshops in Computing, Springer-Verlag, Berlin Heidelberg New York (1995) 55-68.
4. Bloom, B., Istrail, S. Meyer, A.R.: Bisimulation Can't Be Traced. In: Journal of the ACM. Vol. 42. No.1. (January 1995) 232-268
5. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A Theory of Communicating Sequential Processes. In: Journal of the ACM, Vol. 31. No. 3. (1984) 560-599
6. Corradini, F., Gorrieri, R., Roccetti, M.: Performance Preorder and Competitive Equivalence. In: Acta Informatica, Vol. 34. (1997)805-835
7. Degano, P., Priami, C.: Non-interleaving Semantics for Mobile Processes. In: Theoretical Computer Science, Vol. 216. (1999) 237-270
8. De Nicola, R., Hennessy, M.: Testing Equivalences for Processes. In: Theoretical Computer Science, Vol. 34. (1984) 83-133

9. Godefroid, P. Partial Order Methods for the Verification of Concurrent Systems. Doctoral Thesis. University of Liege. (1995)
10. Groote, J. F., Vaandrager, F.: Structured Operational Semantics and Bisimulation as a Congruence. In: Information and Computation, Vol. 100. (1992) 202-260
11. Hennessy, M.: Algebraic Theory of Processes. MIT Press, Cambridge, Mass. (1988)
12. Hennessy, M., Milner, R.: Algebraic Laws for Nondeterminism and Concurrency. In: Journal of the Association of Computing Machinery, Vol. 32, No. 1. (1985) 137-161
13. Hoare, C.A.R.: Communicating Sequential Processes. In: (McKeag, R.M., Macnaghten, A.M. (eds.): On the Construction of Programs—an Advanced Course. Cambridge University Press. (1980) 229-254
14. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall International, London (1985)
15. Ingólfsdóttir, A., Schalk, A.: A Fully Abstract Denotational Model for Observational Congruence. Basic Research in Computer Science Report BRICS-RS-95-40 (August 1995)
16. Institute of Electrical and Electronics Engineers, IEEE Standard VHDL Language Reference Manual. IEEE Press, New York (1993)
17. Lüttgen, G., Vogler, W.: A Faster-than Relation for Asynchronous Processes. ICASE Report No. 2001-2. NASA Langley Research Center, Hampton, Virginia (January 2001)
18. Milner, R.: Calculi for Synchrony and Asynchrony. In: Theoretical Computer Science, Vol. 25 (1983) 267-310
19 Milner, R.: Communication and Concurrency. Prentice Hall, New York, London (1989)
20. Olderog, E.R., Hoare, C.A.R.: Specification-oriented Semantics for Communicating Processes. In: Acta Informatica, Vol. 23. (1986) 9-66
21. Park, D.M.R.: Concurrency and Automata on Infinite Processes. In: Deussen P. (ed.): Proceedings 5th GI Conference. Lecture Notes in Computer Science, Vol. 104. Springer-Verlag, Berlin Heidelberg New York (1981) 167-183
22. Phillips, I.C.C.: Refusal Testing. In: Theoretical Computer Science, Vol. 50 (1987) 241-284
23. Rounds, W.C., Brookes, S.D.: Possible Futures, Acceptances, Refusals and Communicating Processes. In: Proceedings 22nd Annual Symposium on Foundations of Computer Science. IEEE, New York (1981) 140-149
24. Segala, Roberto.: A Process Algebraic View of I/O Automata. MS Thesis. Massachusetts Institute of Technology, Cambridge, Mass. (1994)
25. Stevens, K.S.: Practical Verification and Synthesis of Low Latency Asynchronous Systems. Doctoral Dissertation. University of Calgary. Calgary, Alberta, Canada (1994)
26. van Glabbeek, R.J.: The Linear Time—Branching Time Spectrum II. In: CONCUR'93. Lecture Notes in Computer Science, Vol. 715. Springer-Verlag, Berlin Heidelberg New York (1993)