Congruent Weak Conformance

Ronald W. Brower, *Member, IEEE* and Kenneth S. Stevens, *Senior Member, IEEE*

***Abstract*—Congruent weak conformance is a property between formal models capturing the desired relationship between a specification and its implementation by allowing unused and redundant circuitry and tolerating unspecified behavior in the unreachable state space. By providing greater flexibility in design than previous properties, it becomes a useful tool to validate transformational systems, such as logic synthesis and hardware description language translation systems.**

***Index terms*—formal methods, process algebras, congruence, conformance, hardware equivalence.**

I. Introduction

Engineers are continually challenged to produce electronic designs that meet specification; and logisticians are forever seeking replacements for obsolete, non-procurable microcircuits. Thus there is a general need to find circuits and circuit models that are "equivalent" either to a specification model or to some obsolete part that needs to be replaced. However a moment's reflection reveals that *equivalence* is a stronger notion than what is really needed or desired.

First of all, equivalent *speed* is not necessary. One can often replace an obsolete circuit with a faster circuit of equivalent function. This approach springs from the rationalization that the faster part can certainly keep pace with system demands, while timing constraints simply become less stringent. However, introducing a speedier component can uncover race conditions and hazards that were safeguarded by the delays inherent in the original component. In fact, practitioners often deliberately introduce delays to recover timing safeguards when faster parts are used.

Secondly, excess or redundant circuitry in the implementation can often be tolerated. The extra circuitry can simply sit idle, with pins either unconnected or grounded. Also, unneeded behaviors at connected pins can often be ignored during certain phases of the execution. For example, testability circuits constitute redundant logic when a circuit is under normal operation.

Thirdly, options allowed by output concurrency can be exploited. If the specification calls for the production of two concurrent outputs *x* and *y*, then both output interleavings: *x* followed by *y*, and *y* followed by *x*, are admissible. The original implementing device may consistently produce one interleaving and the replacement device the other. One would never consider the two devices "equivalent," yet each may serve equally well within a specific application.

Examples of hardware equivalences abound [1-12, 15, 16]. Any equivalence relation enjoys the symmetric property which requires that $A = B$ implies $B = A$. As noted before, however, designers and logisticians may settle for devices that "exceed" the specification, rather than merely "equaling" it. Symmetry is not necessary. *A* might "comply with" *B*, yet *B* could never "comply with" *A*. To truly model the notion of device compliance a hardware partial order is more useful than an equivalence.

The new property of *congruent weak conformance* captures the desired relationship between a specification and a conforming implementation. Congruent weak conformance will be useful in supporting future research which seeks to link simulation-based hardware description languages such as VHDL to process algebras such as CCS [8]. Once established, this link will allow stricter verifications of VHDL models based on the bisimulation semantics of CCS.

II. Example

Consider a circuit specified to convert binary-coded-decimal (BCD) to pure decimal. Four input bits are needed to encode a decimal digit. The converter will need four inputs corresponding to each of the encoding bits. Call them *a*, *b*, *c* and *d*. The ten outputs will be labeled $\overline{o}_1, \overline{o}_2, ..., \overline{o}_9$ corresponding to the decimal digit detected. One can think of the outputs as ten lights. Each time there is change on an input bit, one of the lights turns on while another is extinguished. According to the specification, one will not care if *momentarily* two are lit, or none are lit. The

---

Mr. Brower is a Ph.D. candidate at the Air Force Institute of Technology, Wright-Patterson AFB, OH. Dr. Stevens is with Intel Corporation, Hillsboro, OR.

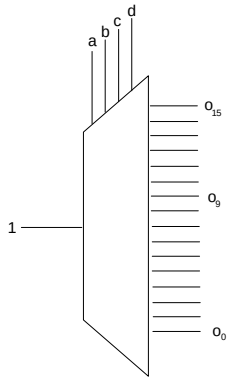
This work is supported by the Air Force Research Laboratory, Wright-Patterson AFB, OH.

CCS specification model will have ten named[1] states corresponding to each decimal digit detected. In the specification model given below, the shorthand notation $(\bar{o}_0|\bar{o}_1)$ is used to express the concurrency of output signals.[2]

$$S \stackrel{def}{=} a.(\bar{o}_0|\bar{o}_1).S1 + b.(\bar{o}_0|\bar{o}_2).S2 + c.(\bar{o}_0|\bar{o}_4).S4 + d.(\bar{o}_0|\bar{o}_8).S8$$
$$S1 \stackrel{def}{=} a.(\bar{o}_0|\bar{o}_1).S + b.(\bar{o}_1|\bar{o}_3).S3 + c.(\bar{o}_1|\bar{o}_5).S5 + d.(\bar{o}_1|\bar{o}_9).S9$$
$$S2 \stackrel{def}{=} a.(\bar{o}_2|\bar{o}_3).S3 + b.(\bar{o}_2|\bar{o}_0).S + c.(\bar{o}_2|\bar{o}_6).S6$$
$$S3 \stackrel{def}{=} a.(\bar{o}_2|\bar{o}_3).S2 + b.(\bar{o}_3|\bar{o}_1).S1 + c.(\bar{o}_3|\bar{o}_7).S7$$
$$S4 \stackrel{def}{=} a.(\bar{o}_4|\bar{o}_5).S5 + b.(\bar{o}_4|\bar{o}_6).S6 + c.(\bar{o}_4|\bar{o}_0).S$$
$$S5 \stackrel{def}{=} a.(\bar{o}_5|\bar{o}_4).S4 + b.(\bar{o}_5|\bar{o}_7).S7 + c.(\bar{o}_5|\bar{o}_1).S1$$
$$S6 \stackrel{def}{=} a.(\bar{o}_6|\bar{o}_7).S7 + b.(\bar{o}_6|\bar{o}_4).S4 + c.(\bar{o}_2|\bar{o}_6).S2$$
$$S7 \stackrel{def}{=} a.(\bar{o}_7|\bar{o}_6).S6 + b.(\bar{o}_7|\bar{o}_5).S5 + c.(\bar{o}_7|\bar{o}_3).S3$$
$$S8 \stackrel{def}{=} a.(\bar{o}_8|\bar{o}_9).S9 + b.(\bar{o}_8|\bar{o}_0).S$$
$$S9 \stackrel{def}{=} a.(\bar{o}_8|\bar{o}_9).S8 + d.(\bar{o}_9|\bar{o}_1).S1$$

Only the states *S* and *S1* respond to all four inputs because combinations above **1001** are illegal BCD codes. Omitting the input transitions that would result in illegal codes in the equations for *S2* to *S9* constitutes the specification's guarantee that the illegal input combinations will not be received.

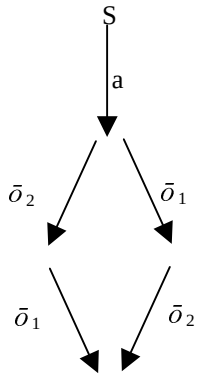
Given the above specification *S*, what constitutes a valid implementation? A 4:16 demultiplexer, or "demux," as shown below, is an obvious choice. The inputs *a*, *b*, *c*, and *d* form the four select lines of the demux. Of the sixteen outputs, only ten are used. A fifth input pin, here hardwired to **1**, represents the multiplexed input. Note therefore that a conforming implementation must have a pin for every input and output called out by the specification, though it may have more.



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.

$$I \stackrel{def}{=} a.(\bar{o}_0|\bar{o}_1).I1 + b.(\bar{o}_0|\bar{o}_2).I2 + c.(\bar{o}_0|\bar{o}_4).I4 + d.(\bar{o}_0|\bar{o}_8).I8$$
$$I1 \stackrel{def}{=} a.(\bar{o}_0|\bar{o}_1).I + b.(\bar{o}_1|\bar{o}_3).I3 + c.(\bar{o}_1|\bar{o}_5).I5 + d.(\bar{o}_1|\bar{o}_9).I9$$
…
$$I9 \stackrel{def}{=} a.(\bar{o}_9|\bar{o}_8).I8 + b.(\bar{o}_9|\bar{o}_{11}).I11 + c.(\bar{o}_9|\bar{o}_{13}).I13 + d.(\bar{o}_1|\bar{o}_9).I1$$

This implementation has more states than the specification since it can execute illegal sequences. The illegal transitions are allowed because the specification guarantees that they are unreachable—the illegal input combinations will never be forthcoming. One might hastily conclude that implementations must duplicate all the states of the specification, with additional states allowed. Yet this is not the case. Although implementation *I* gratuitously generates all the possible output interleavings allowed by the specification, 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, especially when the delays are due solely to passive components. Take, for example, the transitions from *S* to *S1*. The concurrency of the outputs is represented by a diamond in the transition diagram below. Clearly, the implementation need only navigate one path through this diamond, or through any such output "burst." The same is not true for inputs. When an input concurrency is present, as in the case of the C-element [13], the implementation must be poised to accept any possible interleaving that may come and therefore must be able navigate all paths through a specified input burst.



A "second cut" implementation, *J*, chooses specific output interleavings where possible. This implementation might look something like this:

$$J \stackrel{def}{=} a.\bar{o}_1.\bar{o}_0.J1 + b.\bar{o}_0.\bar{o}_2.J2 + c.\bar{o}_4.\bar{o}_0.J4 + d.\bar{o}_0.\bar{o}_8.J8$$
$$J1 \stackrel{def}{=} a.\bar{o}_0.\bar{o}_1.J + b.\bar{o}_3.\bar{o}_1.J3 + c.\bar{o}_1,\bar{o}_5.J5 + d.\bar{o}_1.\bar{o}_9.J9$$
…

and so forth where one specific interleaving is chosen at each output concurrency. Thus, when presented with an output concurrency, the implementation can implement any or all the paths, as long as at least one path is implemented.

Implementations *I* and *J* do indeed accept more input behaviors than the specification calls out and both are able to

[1] Here ten of the states bear explicit names, but the model has many more intermediate states. There is a state after the occurrence of each atomic action.

[2] This shorthand, which one can think of as a "parallelism of actions," is not part of the CCS formal syntax.

generate the unused outputs should an illegal input code be forthcoming. However, the parent system does not care if the illegal inputs are properly decoded or not. In fact, designers will usually want to exploit this "don't care" region of behavior to produce more efficient designs.

The BCD decoder example shows how a compliant implementation can exceed the specification in the number of I/O pins, and can generate illegal behavior in the unreachable state space. In general it can possess more behaviors than the specification, thought it can get by with fewer output behaviors.

III. Congruent Weak Conformance

The authors have devised a new property called *congruent weak conformance* to capture the intuitive notion of conformance presented above. This property is symbolized by '$\trianglerighteq_w$'. By definition, whenever $I \trianglerighteq_w S$ holds between implementation *I* and specification *S* then the following four laws govern what must transpire when either agent requests an input, or issues an output or hidden action:

Law of Specified Input or Tau (LSIT)

$\forall \alpha \in \mathcal{A}(S) \cup \{\tau\}$, whenever $S \stackrel{\alpha}{\Rightarrow} S'$ then

$\exists t \in (\mathcal{A}(S) \cup \overline{\mathcal{EX}tr})^*$ such that

(1) $I \stackrel{t}{\Rightarrow} I'$

(2) $t \uparrow \mathcal{A}(S) = \hat{\alpha}$

(3) $I' \trianglerighteq_w S'$

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) $t \uparrow \mathcal{A}(S) = s$

(4) $I' \trianglerighteq_w S'$

Law of Implemented Input (LII)

$\forall \gamma \in \mathcal{A}(S)$, whenever $I \stackrel{\gamma}{\rightarrow} I'$ and $S \stackrel{\gamma}{\Rightarrow}$ then

(1) $S \stackrel{\gamma}{\Rightarrow} S'$

(2) $I' \trianglerighteq_w S'$

Law of Implemented Output or Tau (LIOT)

$\forall \beta \in \overline{\mathcal{A}}(I) \cup \{\tau\}$, whenever $I \stackrel{\beta}{\rightarrow} I'$ and $\delta \equiv \beta \uparrow \overline{\mathcal{A}}(S)$ then

(1) $S \stackrel{\delta}{\Rightarrow} S'$

(2) $I' \trianglerighteq_w S'$

A technical description of congruent weak conformance and proofs of its important properties are outside the scope of this paper, but will be published shortly.

Congruent weak conformance is called "weak" in the same sense as weak bisimulation [8:108], *i.e.*, it abstracts away internal actions that are irrelevant to the observable behavior of devices. Congruent weak conformance nevertheless respects internal actions that lead to instability [8:112]. In that regard it is similar to observational congruence [8:153]. Like its predecessor, *logic conformance* [14:136-145], congruent weak conformance does *not* require the symmetric property and thereby imparts greater freedom to implementation designs than do hardware equivalences. Furthermore, both logic conformance and congruent weak conformance allow unspecified behavior as long as such behavior occurs within the unreachable state space.

Congruent weak conformance is an improvement over all previous properties in several respects:

1. Congruence weak conformance allows extra input and output ports or pins, called *extraneous* pins, in the implementation. The role of extraneous inputs is restricted somewhat so that they do not block specified behavior. Extraneous outputs, however, can freely interleave all behavior, subject only to the *relative stability* requirement given below.
2. The implementation can chose a single path through an output concurrency burst, instead of having to implement all such paths.
3. Congruent weak conformance uses a new kind of stability called *relative stability*. Relative stability recognizes the ability of extraneous outputs to play the same role as internal action to in yielding unstable models.
4. Congruent weak conformance employs several rules of construction for building compound models. Though they seem restrictive at first glance, these rules are indeed reasonable as well as consistent with good design intent. Violating these rules is tantamount to changing the specification after the implementation is begun. When these rules are employed congruent weak conformance can indeed be shown to be a *congruence* (hence the name). Congruent properties are preserved by all the operators of the underlying algebra. In practical terms, congruence allows for the safe substitution of conforming parts within a system.

IV. Conclusion

In work yet to be published, we have formally proven that congruent weak conformance is indeed a congruence and thus correctly models "safe substitution." We foresee the useful application of this property to tools that transform models between design languages. For example, a tool to transform models from a design languages such as VHDL to CCS would allow the greater verification powers of CCS to accrue to VHDL models. Of course, the event-based simulation semantics of VHDL do not match the bisimulation semantics of CCS. Thus, such a transformed model can not be called "equivalent" to its VHDL original. However, one

does not want the principle of safe substitution to be lost, so the property of congruent weak conformance between models ought to be preserved in the course of the translation. Since we have strived to develop as unrestrictive a property as possible, it will be easier to devise tools that preserve congruent weak conformance than any of the other properties or equivalences. Thus we recommend, when developing such tools, that each transformation be validated by formal proof that it preserves congruent weak conformance.

V. REFERENCES

[1] Bloom, B., S. Istrail and A. R. Meyer. "Bisimulation Can't Be Traced: Preliminary Report," *15th ACM Symposium on Principles of Programming Languages* (POPL), pp. 229-239, San Diego, CA. 1988.

[2] Brookes, S. D., C. A. R. Hoare and A. W. Roscoe. "A Theory of Communicating Sequential Processes," *JACM* **31**(3), pp. 560-599. 1984.

[3] De Nicola, R. and M. Hennessy. "Testing Equivalences for Processes," *Theoretical Computer Science* **34**, pp. 83-133. 1984.

[4] Groote, J. F. and F. W. Vaandrager. *Structured Operational Semantics and Bisimulation as a Congruence*. Report CS-R8845, Centrum voor Wiskunde En Informatica, Amsterdam. 1988.

[5] Hennessy, M. and R. Milner. "Algebraic Laws for Nondeterminism and Concurrency," *JACM* **32**(1), pp. 137-161. 1985.

[6] Hoare, C. A. R. "Communicating Sequential Processes," *On the Construction of Programs—an Advanced Course* (R. M. McKeag and A. M. Macnaghten, eds.), pp. 229-254. Cambridge University Press. 1980.

[7] Milner, R. "Calculi for Synchrony and Asynchrony," *Theoretical Computer Science* **25**, pp. 267-310. 1983.

[8] Milner, R. *Communication and Concurrency*, Prentice Hall. New York. 1989.

[9] Olderog, E. R. and C. A. R. Hoare. "Specification-oriented Semantics for Communicating Processes," *Acta Informatica* **23**, pp. 9-66. 1986.

[10] Park, D. M. R. "Concurrency and Automata on Infinite Processes," *Proceedings 5th GI Conference* (P. Deussen, ed.) LNCS 104, pp. 167-183. Springer-Verlag, 1981.

[11] Phillips, I. C. C. "Refusal Testing," *Theoretical Computer Science* **50**, pp. 241-284. 1987.

[12] Pnueli, A. "Linear and Branching Structures in the Semantics and Logics of Reactive Systems," *Proceedings ICALP 85*, Nafplion (W. Brauer, ed.), LNCS 194, pp. 15-32. Springer-Verlag. 1985.

[13] Shams, M., J. C. Ebergen and M. I. Elmasry. "Modeling and Comparing CMOS Implementations of the C-Element," *IEEE Transactions on VLSI Systems* **6**(4), pp. 563-567. December 1998.

[14] Stevens, K. S. *Practical Verification and Synthesis of Low Latency Asynchronous Systems*. Doctoral Dissertation. University of Calgary. Calgary, Alberta, Canada. 1994.

[15] Rounds, W. C. and S. D. Brookes. "Possible Futures, Acceptances, Refusals and Communicating Processes," *Proceedings 22nd Annual Symposium on Foundations of Computer Science*, pp. 140-149. IEEE. New York. 1981.

[16] van Glabbeek, R. J. *The Linear Time - Branching Time Spectrum*. Technical Report CS-R9029, Centre for Mathematical and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands, 1990.