Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 265 (2010) 301–324

www.elsevier.com/locate/entcs

Geometry of Synthesis II: From Games to Delay-Insensitive Circuits

Dan R. Ghica and Alex Smith

University of Birmingham, U.K.

Abstract

This paper extends previous work on the compilation of higher-order imperative languages into digital circuits [4]. We introduce concurrency, an essential feature in the context of hardware compilation and we re-use an existing game model to simplify correctness proofs. The target designs we compile to are asynchronous *event-logic* circuits, which naturally match the asynchronous game model of the language.

Keywords: Game semantics, asynchronous digital circuits.

1 Introduction

In previous work [4] the first author showed a circuit model for a higher-order imperative and procedural language. The input-output behaviour of the circuit model is closely related to a game-like semantics [2,11] of the programming language. The type system of the language, *Syntactic Control of Interference* (SCI) [18], is designed so that it only allows contraction, i.e. sharing of identifiers, in product formation, prohibiting it in function application. This restriction greatly simplifies the game model, and consequently its circuit realization, and ensures that any term in the language has a finite-state model, thus allowing a "static" circuit realization which does not need to rely on dynamic components (random-access memory).

The circuit model in *loc. cit.* is used as a basis for a "hardware compiler" from SCI to gates [6]. However, the technique has several technical shortcomings. First, the object language has no parallelism, although the type system has elegant support for it. This is a significant restriction especially considering the practical application to hardware compilation; a main advantage of having a direct circuit instantiation of a program, as opposed to running it on a CPU, is the possibility to create parallel threads with minimal run-time overhead. The second problem is the use of a "game-like" semantics for the object language instead of the already established game semantic formalism, which leads to a certain level of informality in

doi:10.1016/j.entcs.2010.08.018

the proof of correctness. Finally, the third problem is that the semantic model used is essentially asynchronous while, for pragmatic reasons, the circuits which are the target of the compiler are synchronous. This leads to further problems with proofs of correctness, problems which were addressed in a rather ad hoc way. A principled method for the synthesis of synchronous circuits from asynchronous specifications is now studied separately [14].

In this paper we correct all these problems by adding parallel composition and logical operators, by using a standard game-semantic model for the language, and by representing it using a well-known class of asynchronous circuits, Sutherland's event logic [20].

2 Syntactic Control of Interference

The primitive types of the language are commands, memory cells and (boolean) expressions: $\sigma ::= \mathsf{com} \mid \mathsf{var} \mid \mathsf{exp}$. The static nature of hardware requires a bounded data type, so for simplicity we only deal with booleans. Bounded integers can be added in a conceptually straightforward way. The language contains function types and products $\theta ::= \sigma \mid \theta \times \theta' \mid \theta \to \theta$. The special feature of this affine type system is that pairs *may* share identifiers but functions *may not* share identifiers with their arguments. Term types are described by typing judgments of the form $\Gamma \vdash M : \theta$, where $\Gamma = x_1 : \theta_1, \ldots x_n : \theta_n$ is a variable type assignment, M is a term and θ the type of the term.

$$\frac{}{x : \theta \vdash x : \theta}\ \text{Identity} \qquad \frac{\Gamma \vdash M : \theta}{\Gamma, x : \theta' \vdash M : \theta}\ \text{Weakening}$$

$$\frac{\Gamma, x : \theta' \vdash M : \theta}{\Gamma \vdash \lambda x.M : \theta' \to \theta} \to \text{Intro} \qquad \frac{\Gamma \vdash M : \theta' \qquad \Gamma \vdash N : \theta}{\Gamma \vdash \langle M, N\rangle : \theta' \times \theta} \times \text{Intro}$$

$$\frac{\Gamma \vdash F : \theta' \to \theta \qquad \Delta \vdash M : \theta'}{\Gamma, \Delta \vdash FM : \theta} \to \text{Elim}\ .$$

The functional constants of the language are:

$0, 1 : \mathsf{exp}$	constant
$\mathsf{skip} : \mathsf{com}$	no-op
$\mathsf{asg} : \mathsf{var} \times \mathsf{exp} \to \mathsf{com}$	assignment
$\mathsf{der} : \mathsf{var} \to \mathsf{exp}$	dereferencing
$\mathsf{seq} : \mathsf{com} \times \mathsf{com} \to \mathsf{com}$	sequencing
$\mathsf{seq} : \mathsf{com} \times \mathsf{exp} \to \mathsf{exp}$	sequencing with boolean
$\mathsf{par} : \mathsf{com} \to \mathsf{com} \to \mathsf{com}$	parallel execution
$\mathsf{neg} : \mathsf{exp} \to \mathsf{exp}$	negation
$\mathsf{op} : \mathsf{exp} \times \mathsf{exp} \to \mathsf{exp}$	logical operations
$\mathsf{if} : \mathsf{exp} \times \mathsf{com} \times \mathsf{com} \to \mathsf{com}$	branching
$\mathsf{while} : \mathsf{exp} \times \mathsf{com} \to \mathsf{com}$	iteration
$\mathsf{newvar} : (\mathsf{var} \to \mathsf{com}) \to \mathsf{com}$	local variable

$\mathsf{newvar} : (\mathsf{var} \to \mathsf{exp}) \to \mathsf{exp}$ local variable.

Product has syntactic precedence over arrow, which associates to the right.

This "functionalised" syntax can be represented in a more conventional way. For example, a program such as `bool x; if (y) x=z and t else x=z or t` can be written using the functionalised syntax as:

$$\mathsf{newvar}(\lambda x.\mathsf{if}\, y\, (\mathsf{asg}((\mathsf{and}(\mathsf{der}\, z)(\mathsf{der}\, t)), x))(\mathsf{asg}((\mathsf{or}((\mathsf{der}\, z), (\mathsf{der}\, t))), x))).$$

Although the former is more readable the latter is more convenient for presenting the semantics.

Note the difference between the type of sequential composition $\mathsf{seq} : \mathsf{com} \times \mathsf{com} \to \mathsf{com}$ and that of parallel composition $\mathsf{par} : \mathsf{com} \to \mathsf{com} \to \mathsf{com}$. The uncurried type allows contraction whereas the curried type prevents it. Therefore, $c : \mathsf{com} \vdash c; c : \mathsf{com}$ is derivable but $c : \mathsf{com} \vdash c \mid\mid c : \mathsf{com}$ is not.

SCI here is the basic form of a well-studied type system [18,17,13]; it is also an instance of another more general type system, *Syntactic Control of Concurrency* (SCC) [9], which allows contraction in non-sequential contexts but only when there is a static bound on the number of times each identifier is used in concurrent contexts. SCC types are generated by the grammar $\theta ::= \sigma \mid \gamma \times \gamma \mid \gamma \to \theta$, where $\gamma ::= \theta^n$. Typing judgements have form $\Gamma \vdash_r M : \theta$ where $\Gamma = x_1 : \theta_1^{n_1}, \ldots x_k : \theta_k^{n_k}$ and we write $m \cdot \Gamma = x_1 : \theta_1^{mn_1}, \ldots x_k : \theta_k^{mn_k}$. The type system SCC is similar to that of SCI except for allowing contraction while incorporating bookkeeping rules for "concurrency bounds":

$$\frac{\Gamma, x : \theta^m, y : \theta^n \vdash_r M : \theta'}{\Gamma, x : \theta^{m+n} \vdash_r M[x/y] : \theta'} \qquad \frac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n \cdot \Delta \vdash_r MN : \theta'}.$$

The functional constants of SCC are the same as those of SCI, except that the local-variable binder can accommodate variables shared in non-sequential contexts, $\mathsf{newvar} : (\mathsf{var}^n \to \mathsf{com}) \to \mathsf{com}$. The difference between SCC and SCI is that $c \mid\mid c$ can be typed, and the type system tracks the fact that c is used in (at most) two concurrent threads, $c : \mathsf{com}^2 \vdash_r c \mid\mid c : \mathsf{com}$. The following property is immediate:

Proposition 2.1 *If* $\Gamma \vdash M : \theta$ *then* $\Gamma_r \vdash_r M : \theta$ *where* $\Gamma_r(x) = \Gamma(x)^1$.

Any SCI term is an SCC term in which all free identifiers have bound 1. Note that the reverse is not true. The fact that SCI is an instance of SCC means that we can use its fully abstract game semantic model [9] as a basis for our circuit semantics. The SCC game model is constructed from the game model of *Idealized Concurrent Algol* (ICA) [8], which we present below.

Definition 2.2 An *arena* A is a triple $\langle M, \lambda, \vdash \rangle$ where:

- M is a set of "moves";
- $\lambda : M \to \{O, P\} \times \{Q, A\}$ is a "labelling" function;
- $\vdash \subseteq M \times M$ is an "enabling" relation satisfying:
  · if $\forall m \in M.m \not\vdash n$ then $\lambda n = OQ$;

· if $m \vdash n$ then $(\pi_1 \circ \lambda)(m) \neq (\pi_1 \circ \lambda)(n)$ and $(\pi_2 \circ \lambda)(m) = Q$.

Moves are basic observable actions for some type. The labelling function distinguishes between *Opponent* and *Proponent* moves, as well as *Question* and *Answer* moves. *Enabling* establishes a fundamental causation relation between moves, subject to some conditions: only Opponent Questions can be without an enabler, an enabling move and an enabled move have distinct Opponent/Proponent labels and only Question moves can be enabling moves. For more detailed intuitive explanations of game semantic concepts the reader is referred to the literature [3,5].

We denote the set of moves in an arena A without an enabler the set of *initial* moves I_A. Composite arenas can be constructed as follows:

$$\langle M, \lambda, \vdash\rangle \times \langle M', \lambda', \vdash'\rangle = \langle M + M', [\lambda, \lambda'], \vdash + \vdash'\rangle$$
$$\langle M, \lambda, \vdash\rangle \Rightarrow \langle M', \lambda', \vdash'\rangle = \langle M + M', [\lambda^*, \lambda'], \vdash + \vdash' + I' \times I\rangle.$$

Above, by $M + M'$ we mean the disjoint union of the two sets, by $[\lambda, \lambda']$ the co-pairing of the two labelling functions, λ^* is like lambda except the O and P labels are swapped, i.e. $\lambda^* = \langle (P \mapsto O, O \mapsto P), \mathrm{id}_{Q,A}\rangle \circ \lambda$.

The arenas used to interpret base types are as follows:

$$\begin{aligned}
[\![\mathsf{com}]\!] &= \big\langle \{q, a\}, (q \mapsto OQ, a \mapsto PA), \{(q, a)\}\big\rangle \\
[\![\mathsf{exp}]\!] &= \big\langle \{q, t, f\}, (q \mapsto OQ, t \mapsto PA, f \mapsto PA), \{(q, t), (q, f)\}\big\rangle \\
[\![\mathsf{var}]\!] &= \big\langle \{q, t, f, w_t, w_f, a\}, (q \mapsto OQ, t \mapsto PA, f \mapsto PA, w_t \mapsto OQ, w_f \mapsto OQ, a \mapsto PA), \\
&\qquad \{(q, t), (q, f), (w_t, a), (w_f, a)\}\big\rangle.
\end{aligned}$$

The basic observable actions for commands are q, running the command, and a, finishing execution. For boolean expression the actions are evaluating the expression q then producing either true t or false f. For variables, we can either query the variable q or write true w_t or false w_f; the answers to the query are t true or f false, and for w_t, w_f an acknowledgement a, respectively.

A *justified sequence* in an arena A is a finite sequence of moves of A equipped with pointers. Occurrences of initial moves have no pointers, but any other move must have a unique pointer to an earlier occurrence of a move that enables it. If a question does not justify an answer in a justified sequence we say that it is *pending*. Not all justified sequences are valid. In order to constitute a *legal play* a justified sequence must satisfy the following condition:

Definition 2.3 The set P_A of *plays* over arena A consists of the justified sequences s over A such that

- in any prefix $s' = \cdots q \cdots m$ of s such that q justifies m the question q must be pending before m;
- in any prefix $s' = \cdots q \cdots a$ of s such that q is answered by a, no questions justified by q are pending.

For two justified sequences s, s' we denote by $s \coprod s'$ the set of all their interleavings. This is applied point-wise to sets of justified sequences $S \coprod S'$. We denote by

$S^{\textcircled{n}} = S \coprod \cdots \coprod S$, n times, and by $S^{\circledast}$ the smallest set such that $S^{\circledast} = S^{\circledast} \coprod S$, the iterated shuffle of S. We say that a set is *O-complete* if for any element s of the set, if so is a legal play with o an O-move then so is also in that set.

Definition 2.4 A *strategy* on arena A is a prefix-closed, O-complete subset of P_A.

Strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ can be composed in a standard way, by considering all possible interactions of plays from τ with shuffled plays from $\sigma^{\circledast}$ in the shared arena, followed by hiding all B moves.

$$\sigma \bullet \tau = \{u \mid u \upharpoonright A, B \in \sigma^{\circledast}, u \upharpoonright B, C \in \tau\} \qquad \sigma ; \tau = \{u \upharpoonright A, C \mid u \in \sigma \bullet \tau\}.$$

A key notion from concurrent game semantics is that of *saturation* [8], reflecting the fact that in an asynchronous setting the program only has a limited amount of control over the ordering of events.

Definition 2.5 Let $\preceq \in P_A \times P_A$ be the least transitive and reflexive relation such that $sos's'' \preceq ss'os''$ and $ss'ps'' \preceq sps's''$, where o is any O-move, p any P-move, and the justification pointers are the same. A strategy σ is *saturated* if and only if for any $s \in \sigma$, if $s' \preceq s$ then $s' \in \sigma$.

Given a set of plays P we denote the least strategy that contains it (i.e. its closure under prefix, O-completion and saturation) $\mathsf{strat}(P)$.

Arenas and saturated strategies form a Cartesian Closed Category in which the objects are arenas and morphisms $A \to B$ are saturated strategies $\sigma : A \Rightarrow B$. The identity strategy is defined by saturating the "copy-cat" strategy common in game semantics: $id_A = \mathsf{strat}\{s \in P_{A \Rightarrow A} \mid t \upharpoonright inl(A) = t \upharpoonright inr(A)$, for all even prefixes t of $s\}$.

The constant functions of the language are interpreted by:

$$\begin{aligned}
[\![\mathsf{skip} : \mathsf{com}]\!] &= \mathsf{strat}(qa), [\![1 : \mathsf{exp}]\!] = \mathsf{strat}(qt), [\![0 : \mathsf{exp}]\!] = \mathsf{strat}(qf)\\
[\![\mathsf{neg} : \mathsf{exp} \to \mathsf{exp}']\!] &= \mathsf{strat}\big(q'q(tf' + ft')\big)\\
[\![\mathsf{seq} : \mathsf{com} \times \mathsf{exp}' \to \mathsf{exp}'']\!] &= \mathsf{strat}\big(q''qaq'(t't'' + f'f'')\big)\\
[\![\mathsf{seq} : \mathsf{com} \times \mathsf{com}' \to \mathsf{com}'']\!] &= \mathsf{strat}\big(q''qaq'a'a''\big)\\
[\![\mathsf{par} : \mathsf{com} \to \mathsf{com}' \to \mathsf{com}'']\!] &= \mathsf{strat}\big(q''qq'aa'a''\big)\\
[\![\mathsf{if} : \mathsf{exp} \times \mathsf{com}' \times \mathsf{com}'' \to \mathsf{com}''']\!] &= \mathsf{strat}\big(q'''qtq'a'a''' + q'''qfq''a''a'''\big)\\
[\![\mathsf{while} : \mathsf{exp} \times \mathsf{com}' \to \mathsf{com}'']\!] &= \mathsf{strat}\big(q''(qtq'a')^*qfa''\big)\\
[\![\mathsf{asg} : \mathsf{var} \times \mathsf{exp}' \to \mathsf{com}'']\!] &= \mathsf{strat}\big(q''q'(f'\,w_f + t'\,w_t)aa''\big)\\
[\![\mathsf{der} : \mathsf{var} \to \mathsf{exp}']\!] &= \mathsf{strat}\big(q'qtt', q'qff'\big)\\
[\![\mathsf{newvar} : (\mathsf{var} \to \mathsf{com}') \to \mathsf{com}'']\!] &= \mathsf{strat}\big(q''q'(w_t\,a(qt)^* + w_f a(qf)^*)^* a'a''\big)\\
[\![\mathsf{newvar} : (\mathsf{var} \to \mathsf{exp}') \to \mathsf{exp}'']\!] &= \mathsf{strat}\big(q''q'(w_t\,a(qt)^* + w_f a(qf)^*)^*(t't'' + f'f'')\big)
\end{aligned}$$

Note that the saturation condition expressed in Def. 2.5 allows the compact definition of par given above, as it generates all the possible shuffles of qa and $q'a'$. On the other hand, saturation applied to the definition of seq does not lead to any new traces.

Binary arithmetical-logical operators can be defined in several ways (sequential, lazy sequential or parallel). Consider this three versions of the OR operator:

$$\begin{aligned}
[\![\mathsf{or_s} : \mathsf{exp} \times \mathsf{exp}' \to \mathsf{exp}'']\!] &= \mathsf{strat}\big(q''qtq'(t'+f')t'' + q''qfq't't'' + q''qfq'f'f''\big)\\
[\![\mathsf{or_l} : \mathsf{exp} \times \mathsf{exp}' \to \mathsf{exp}'']\!] &= \mathsf{strat}\big(q''qtt'' + q''qf(q't't'' + q'f'f'')\big)\\
[\![\mathsf{or_p} : \mathsf{exp} \to \mathsf{exp}' \to \mathsf{exp}'']\!] &= \mathsf{strat}\big(q''q'q(t't + t'f + f't)t'' + q''q'qf'ff''\big).
\end{aligned}$$

The game model for ICA is fully abstract if the language has semaphores and so-called *bad variable constructors*. In the absence of these constructs we can state:

Theorem 2.6 **([8])** *The game model of ICA is sound and adequate, relative to a standard operational semantics.*

Concrete representations of the game model of ICA are complicated by the iterated shuffle operation which is not implementable using finite state automata. In fact, in the presence of semaphores the model of ICA is undecidable [9, Thm. 6]. Bounding the amount of concurrency and interleaving in game models, as SCC does, leads to a finite state model. In fact, for SCI, which is SCC with all bounds set to the unit, the model is particularly simple. For expedience, we will present the model for SCC when it coincides with SCI, i.e. when the bounds are all set to one.

Definition 2.7 A *unit-bounded* set of plays $\underline{P}_A \subseteq P_A$ is the set of all plays such that if $\cdots q \cdots q' \in \underline{P}_A$ and $q, q' \in I_B$ from some sub-arena B of A then q is not pending before q' is played.

This definition is a simplified instance of Def. 12 [9] when the bound is fixed.

For example, in arena $\mathsf{com} \times \mathsf{com}' \to \mathsf{com}''$ the play $q''qaq'$, which occurs in sequential composition is legal, whereas the play $q''qq'$ which occurs in parallel composition is illegal because $\mathsf{com} \times \mathsf{com}$ is a sub-arena of $\mathsf{com} \times \mathsf{com}' \to \mathsf{com}''$ and q, q' are both initial questions in it. However, note that the same play $q''qq'$ is legal in arena $\mathsf{com} \to \mathsf{com}' \to \mathsf{com}''$; this is another explanation for the curried type of parallel composition.

Proposition 2.8 *If $s, s' \in \underline{P}_A$ have the same move occurrences then they have the same justification pointers, i.e. $s = s'$.*

The fact that the justification pointers can be uniquely reconstructed for a given play greatly simplifies the representation of the model, cf. [7].

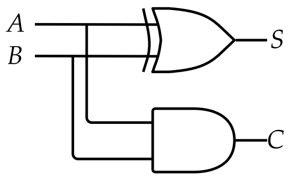
The bounded-play model is in general fully abstract for SCC [9] but it requires the use of semaphores, bad-variable constructors and, in the proof of definability, the use of local identifiers with arbitrary concurrency bounds. For SCI as an instance of SCC it follows immediately from Thm. 2.6 and Prop. 2.8 that

Theorem 2.9 *In the game model of SCI if Opponent plays are unit bounded then all strategies have Proponent unit-bounded plays.*

This is simply a special case of the resource-bound game model of SCC, and it means that the unit-bound game model can be used as a basis for a compiler.

3 Event logic

Consider the typical implementation of a digital adder circuit:



The inputs are A and B and the output are the sum S and the carry C:

$$S = A \oplus B, \qquad C = A \wedge B.$$

Suppose that the circuit is in an initial state, where $A = B = C = S = 0$ and we want to change the input values to $A = B = 1$. In a synchronous (clocked) circuit, the system clock has a period longer than the propagation delay of signals through wires and gates, and values are only considered meaningful on the falling (or raising) edge of the clock, giving them time to stabilize at the correct values of $S = 0, C = 1$. However, in an asynchronous (clock-less) circuit the new input signals will propagate along the wires and reach the four gate inputs at different times. Depending on the relative wire delays, there are 8 different orders in which this can happen. The two gates will see a sequence of four distinct inputs, and produce the corresponding outputs, before settling on the correct values. As inputs change from 0 to 1 on its inputs, the outputs of the AND gate are the sequence 001, which corresponds to a "clean" transition from 0 to 1. However, on the XOR gate, as the inputs change from 0 to 1 the outputs will see the sequence 010. Before settling on the correct value of 0, the circuit shows a spurious value of 1, a so-called *hazard*. If this adder is connected to other circuits then these circuits will consider the hazard value as a genuine value and propagate it, leading to more spurious values and ultimately a rather chaotic circuit behaviour.

In a nut-shell, this is the main problem of asynchronous circuit design, and there exist a variety of theoretical and practical approaches to mitigating it [10]. A particularly interesting and clean solution was proposed by Sutherland in his seminal Turing Award lecture [20]. At the foundation of his approach lies the observation that boolean logic is not particularly well suited to implementing asynchronous circuits, suggesting instead an *event logic*: a logic of pure control, dealing not with "true" and "false" but with the more fundamental notions that "something happened" or "nothing happened". The basic logical functions on events can be (efficiently) implemented as special gates or modules. At the level of physical implementation, an event is either a high-to-low or a low-to-high transition on a wire (the so-called *two-phase event encoding*).

XOR provides an OR-like function for events, producing an output when an event arrives on any of the input ports.

C is the so-called Muller C-element [15, Chap. 10], a fundamental gate in asyn-

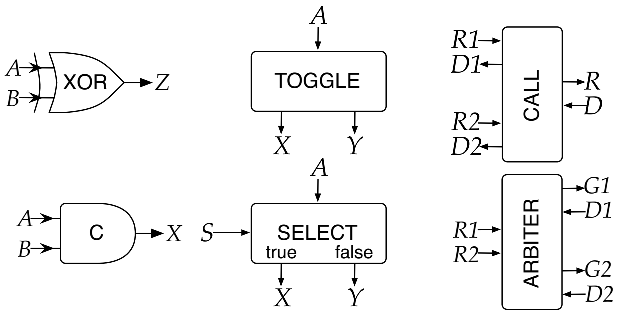


Fig. 1. Logic modules for events

chronous design. It has an AND-like functionality on events, producing an output when events arrive on both input ports.

TOGGLE steers events to its outputs alternately, starting with the dot.

SELECT steers its input event to the the output according to the value of the diamond input S.

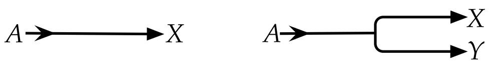
CALL remembers which "client", R_1 or R_2, called more recently and it steers the matching D back to D_1 or D_2 as is the case. CALL can be generalised to the case where R, D, R_i, D_i represent sets of ports.

ARBITER grants service G_1 or G_2 to only one input request R_1 or R_2 at a time, delaying subsequent grants until the matching done event D_1 or D_2.

The *signature* of a circuit is a set of ports (labels) with input or output labels. Let *prefix*(L) be the set of prefixes of a set of traces L. A more formal description of event logic modules can be given in terms of traces of events on their ports. The polarity of the ports is as given in Fig. 1, i.e. A, B inputs, X, Y outputs etc. We only define the behaviour of the following circuits which we shall need, as traces of input and output events:

$$\begin{aligned} [\![XOR]\!] &= \mathit{prefix}(AX + BX)^* \\ [\![C]\!] &= \mathit{prefix}(ABX + BAX)^* \\ [\![CALL]\!] &= \mathit{prefix}(R_1R(DD_1)^* + R_2R(DD_2)^*)^* \end{aligned}$$

To this we must add descriptions of behaviours for plain and forking wire:



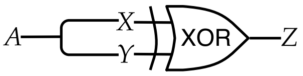
$$[\![WIRE]\!] = \mathit{prefix}(AX)^* \qquad [\![FORK]\!] = \mathit{prefix}(AXY + AYX)^*.$$

We introduce two notions of composition. For two circuits K_1, K_2 with ports A_1, A_2 we defined the "vertical" composition $K_1 \otimes K_2$ with ports $A_1 + A_2$ as their "tensor", obtained simply by placing the two circuits side by side:

$$K_1 \otimes K_2 = \{u \mid u \upharpoonright A_i \in K_i, i = 1, 2\}.$$

Let $\overline{A}$ be a port signature like A but with input-output polarities reversed. For K, L with ports $A + B$, and $\overline{B} + C$ respectively, we can also define a "horizontal" composition $K; L$, by connecting the ports in B with the ones in $\overline{B}$ carrying the same label, resulting in a circuit with ports $A + C$. This notion of composition is more subtle because the interaction between two circuits can lead to "unsafe" traces. We illustrate this with the following example.

Example 3.1



Given the definition of the two circuits, the composition $FORK; XOR$ might be expected to produce input-output traces of the form $(AZZ)^*$. However, if we consider traces including the internal channels X and Y, we can see that these observable traces might correspond to interactions $AXYZZ$, which are from a physical point of view unsafe: if events X and Y arrive very close to each other temporally, then it is possible that the sequence ZZ consists of two events that happen faster than the inertial delay of the wire or the gate and may be suppressed [19, Sec. 6.1.3].

Our notion of composition needs to disallow such unsafe traces. The set of traces of a composite system should only contain those traces that can only be produced safely.

Definition 3.2 We define the *interaction* of two circuits with sets of traces $K : A + B$ and $L : \overline{B} + C$ as the set $K \bullet L = \{u \mid u \upharpoonright (A, B) \in K \land u \upharpoonright (B, C) \in L\}$.

Definition 3.3 Given a trace $u \in U$ over signature $A + B$ we define its *next-action* set $\text{next}_U(u) = \{m \in A + B \mid u \cdot m \in U\}$.

Definition 3.4 We also define the next-action set for an interaction u, where U is a set of traces, as $\text{next}_U(u) = \text{next}_U(u \upharpoonright U)$.

We define $\text{next}^i_U(u)$, $\text{next}^A_U(u)$ or $\text{next}^{A,i}_U(u)$ as the obvious restrictions of the next-action set to inputs (or outputs) or a sub-set of ports or both. A safe interaction between two circuits is one in which the outputs of one of the circuits can be handled by the other as an input and vice versa.

Definition 3.5 An interaction $u \in K \bullet L$, where $K : A + B$ and $L : \overline{B} + C$ is said to be *safe* if and only if $\text{next}^{o,B}_K(u) \subseteq \text{next}^{i,B}_L(u)$ and $\text{next}^{o,B}_L(u) \subseteq \text{next}^{i,B}_K(u)$.

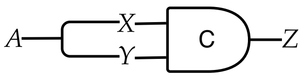
Definition 3.6 If $K : A + B$ and $L : \overline{B} + C$, we define their (safe) *composition* as

$$K; L = \{u \mid u \in (K \bullet L) \upharpoonright (A, C) \land \forall u' \in K \bullet L, \text{ if } u' \upharpoonright (A, C) = u \text{ then } u' \text{ is safe.}\}$$

In other words, the (safe) composition of two circuits will contain only those traces that can be arrived at from safe interactions only.

Example 3.7 $FORK; XOR = \emptyset$. This can be shown analysing all possible interactions. AX is unsafe because $\text{next}^{o,\{X,Y\}}_{FORK}(AX) = \{Y\} \not\subseteq \text{next}^{i,\{X,Y\}}_{XOR}(AX) = \emptyset$ and the safety condition fails. AY is unsafe for a similar reason. Since all interactions are prefixed by AX or AY there are no safe interactions.

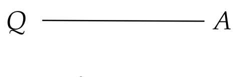
Example 3.8



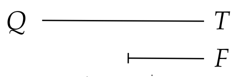
We can show that $FORK; C = (AZ)^*$. The use of safe composition is essential here. For example, the interaction $AXYA \in FORK \bullet C$, but $\text{next}^{o,XY}_{FORK}(AXYA) = \{X, Y\} \not\subseteq \text{next}^{i,XY}_{C}(AXYA) = \emptyset$. This is because we can use the $FORK$ circuit correctly, applying the next input after the outputs have been produced, but this could still be too fast for the C circuit, which may not have produced its output yet and is unable to process more input. Without the safe composition requirement the composition of $FORK$ and C would contain interactions such as $AXYAZXYZ$ which correspond to input-output trace $AAZZ$. In a physical circuit traces $AAZZ$ are possible, if the delay in the input wire is longer than the inertial delay of the output wire, but they are not *safely* possible unless we start taking explicitly into account timing considerations. Such a requirement of a "slow enough" environment is often required in asynchronous design, cf. *burst mode* circuits [21].

4 A circuit model for SCI

The game model for SCI can be represented using only the XOR, C, CALL, WIRE and FORK fragment of event logic. We represent arenas as sets of ports, with a distinct port corresponding to each move. An O move is an input and a P move is an output. Constants are

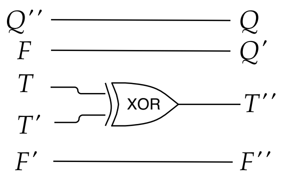


skip : com



1 : exp

The representations of games for the imperative language constants are given in Fig. 2. For now we only consider the representation of lazy sequential operators, such as $\mathsf{or}_l : \mathsf{exp} \times \mathsf{exp}' \to \mathsf{exp}''$:



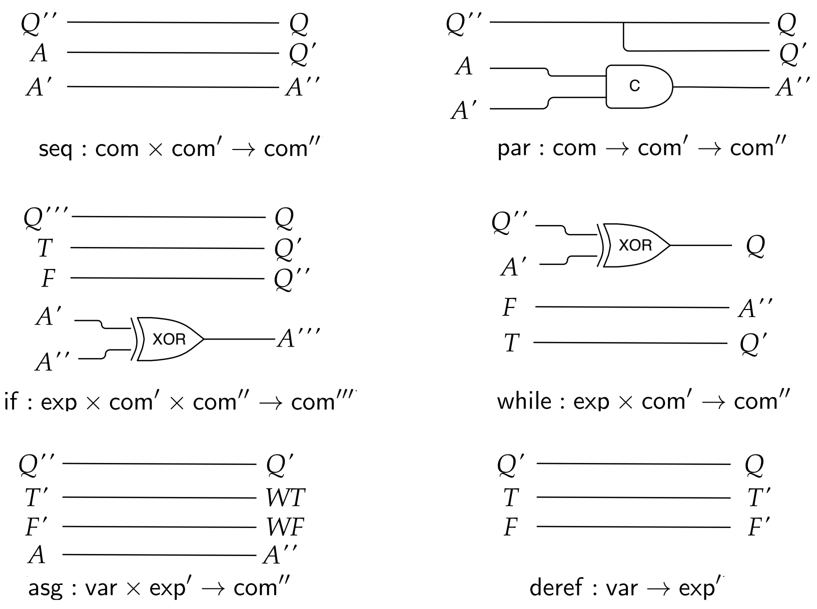
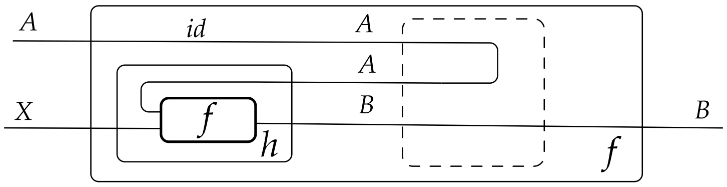


Fig. 2. Event-logic circuits for SCI imperative constants

The other versions (eager sequential, parallel) can be implemented using more complex circuitry which we will discuss at the end of this section. Operators can be extended to finite integers using the dual-rail binary encoding, which means that each digit of the number is represented by two wires, one for the value 0 of the digit and one for the value 1, a standard approach in asynchronous design.

Since there is a reasonable level of abstraction for circuits where circuits with graph-equivalent diagrams are always behaviourally equal (this ignores wire delays), it is obvious that circuit diagrams can be naturally structured into monoidal categories [12], where objects are ports, circuits are morphisms and identity and evaluation are expressed using wires. Such categories play an important role in the analysis of asynchronous concurrency [1]. Without describing the categorical framework (see [4] for a more thorough treatment of this aspect) we will just give the circuits that correspond to the structural aspects of the language. Morphism composition is horizontal composition and tensor product in the category is, on objects, disjoint sum and, on morphisms, vertical circuit composition. The identity is formed of wires. The axioms of the category have intuitive diagrammatic representations. For example, the universal property of the evaluation morphism corresponds to the diagram in Fig. 3. The relabelling of ports on f which gives h is in fact the currying operation.

We will not show that asynchronous circuits form a category, as this requires a precise analysis of what constitutes asynchronous behaviour, which entails the use of notions of causality and saturation quite similar to those we already use in game semantics. Instead, we will show that asynchronous circuits that represent SCI terms have the correct behaviour.



$$\forall f : A \otimes X \to B.\exists h : X \to (A \Rightarrow B).f = eval_{A,B} \circ (id_A \otimes h)$$

Fig. 3. Universal property

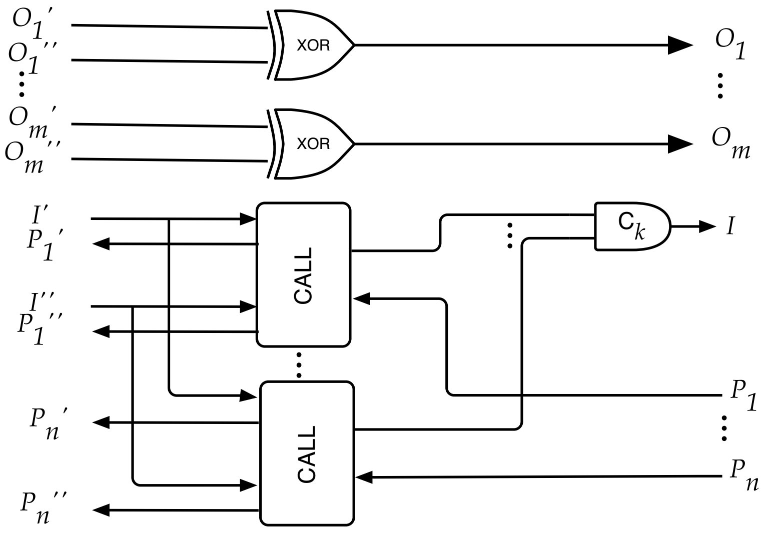
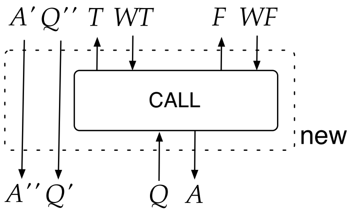


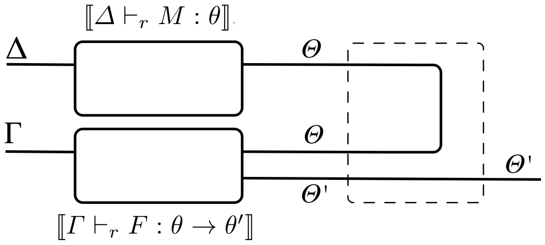
Fig. 4. Generalised CALL module

The CALL module is used to implement the diagonal strategy $\delta_\theta : \theta \to \theta' \times \theta''$ which is needed to implement contraction, e.g. for $\delta_{\mathsf{com}} : \mathsf{com} \to (\mathsf{com}_1 \times \mathsf{com}_2)$ as below. Higher-order contraction is implemented using a generalised CALL module as shown in Fig. 4. The initial moves of δ_θ, which are the initial moves of θ' or θ'' will set the direction in which the CALL module will de-multiplex the P-moves of θ. The non-initial O-moves of θ will just be multiplexed through XOR gates. For simplicity we only show the implementation when there is a unique initial move, for more than one initial move the generalisation is the obvious one.

Perhaps surprisingly, the local-variable binder $\mathsf{newvar} : (\mathsf{var} \to \mathsf{com}') \to \mathsf{com}''$ can be also be implemented by taking advantage of the stateful nature of the CALL module.



SCI terms can be interpreted inductively on the syntax, where terms are formed from constants, contraction (described above), function application, function declaration and free identifiers. Given circuits for terms $[\![\Gamma \vdash_r F : \theta \to \theta']\!]$ and $[\![\Delta \vdash_r M : \theta]\!]$, the circuit for $[\![\Gamma, \Delta \vdash_r F(M) : \theta']\!]$ is


$[\![\Delta \vdash_r M : \theta]\!]$
Δ
Θ
Γ
Θ
Θ'
Θ'
$[\![\Gamma \vdash_r F : \theta \to \theta']\!]$

The sub-circuit identified with a dashed contour, having as ports two instances of the set of ports θ and two instances of the set of ports θ' is the "evaluation morphism" in the category, $ev_{\theta,\theta'} : (\theta \Rightarrow \theta') \otimes \theta \to \theta'$. Function declaration is the currying relabeling of ports discussed earlier, and free identifiers are the identity (wires).

Example 4.1 A simple but useful program which illustrates the compilation of open higher-order programs is *in-place map,* which applies a function f to all elements of a data structure, modifying them in place. Consider an iterator over some data structure, provided with the following interface:

$\mathsf{init} : \mathsf{com}$	initialize an iterator over the data structure;
$\mathsf{curr} : \mathsf{var}$	get the current element in the data structure;
$\mathsf{next} : \mathsf{com}$	advance the iterator to the next element;
$\mathsf{more} : \mathsf{exp}$	return false iff the end of the data structure has been reached.

Note that SCI being a call-by-name language these identifiers represent thunks, i.e. parameter-less procedures. The program for in-place map is:

$$\begin{aligned}&\mathsf{init} : \mathsf{com}, \mathsf{curr} : \mathsf{var}, \mathsf{next} : \mathsf{com}, \mathsf{more} : \mathsf{exp} \vdash_r \\ &\qquad\qquad \lambda f : \mathsf{exp} \to \mathsf{exp}.\mathsf{init}; \mathsf{while}\ (\mathsf{more})(\mathsf{curr} := f(!\mathsf{curr}); \mathsf{next}) : \mathsf{com}.\end{aligned}$$

The structure of the resulting circuit is shown in Fig. 5. The concrete circuit, Fig. 6, is strikingly simple. Ports are annotated with the variable name for readability; top-level ports are *top.q* and *top.a*. For function $f : \mathsf{exp}' \to \mathsf{exp}$ the ports corresponding to the argument are primed. Technically, variable *curr* should go through a contraction circuit $\delta_{var} : \mathsf{var} \to \mathsf{var} \times \mathsf{var}$; however, because the first occurrence uses only the "write" ports and the second only the "read" ports, no connectors need to be actually reused and contraction can be omitted.

It is worth emphasising that conventional hardware compilers, which usually rely on inlining to handle procedure calls, cannot compile open or higher-order programs.

Remark 4.2 The implementation of sequential (eager) and parallel operators re-

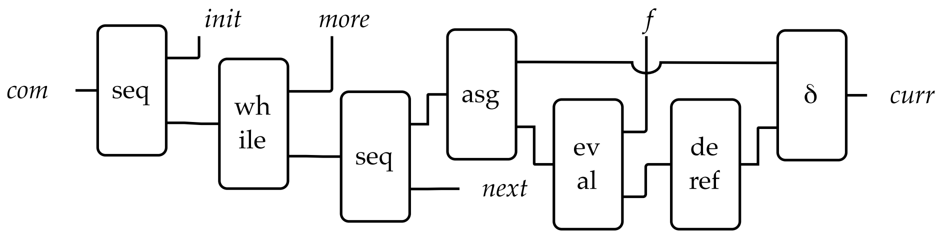


Fig. 5. In-place map, overall structure

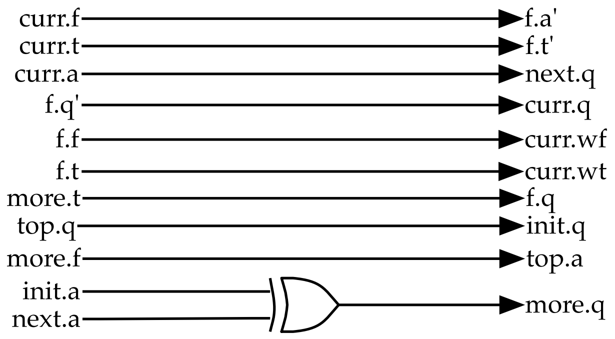
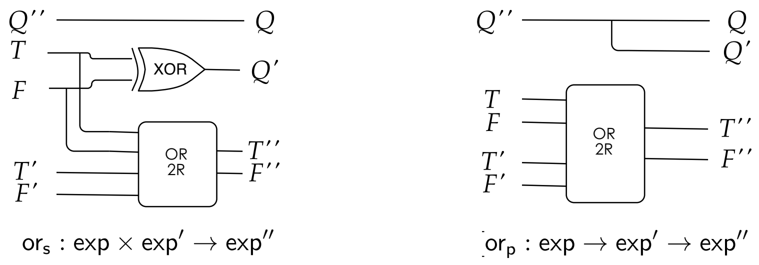


Fig. 6. In-place map, event-logic implementation

quires circuits of the following shape:



Above, OR2R is a *dual-rail two-phase gate*, which can be implemented as in Fig. 7 [16]. We did not include this circuit before because it requires a rather different specification of the C gate, in which consecutive inputs on the same port cancel each other out, $[\![C]\!] = (((AA)^*(BB)^*)^*(ABX + BAX))^*$. This specification complicates the correctness proofs as does the presence of local feed-backs in the OR2R circuit. It is also inconsistent with our physical interpretation of safe composition, since it must record consecutive events on the same port.

5 Correctness

We show two correctness results for the translation of SCI into asynchronous circuits. We show that if the environment sends inputs to a circuit which are consistent with an SCI Opponent behaviour in the strategy represented by the circuit, then the circuit will respond with outputs which are consistent with Player behaviour in the same strategy. This is a statement both of liveness (the circuit *will* respond) and

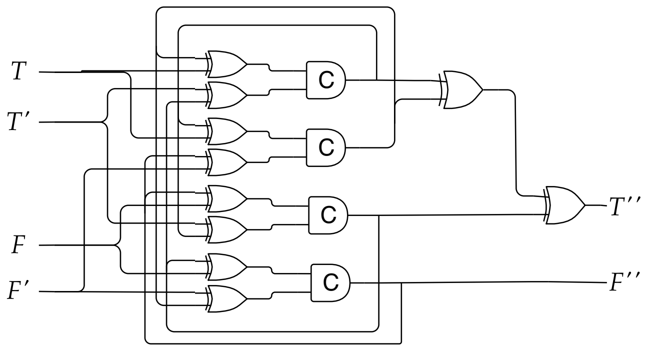


Fig. 7. Dual-rail two-phase OR gate

of safety (the response is correct).

Theorem 5.1 (Correctness) *Let $\Gamma \vdash M : \theta$ be an SCI term, K its event-logic representation and u a trace on K which is also a play in the strategy $[\![M]\!]$. Then*

Safety: $u \cdot \mathrm{next}^o_K(u) \subseteq [\![M]\!]$

Liveness: $\mathrm{next}^o_K(u) = \emptyset$ *if and only if there is no P-move such that $u \cdot n \in [\![M]\!]$.*

Additionally, we show that taking into account the same assumptions about the behaviour of the environment, the circuit will have no failed traces. This is important because the notion of composition of Def. 3.6 only tells us what correct traces are produced, not that no bad interactions occur.

Theorem 5.2 (Program safety) *Let $\Gamma \vdash M : \theta$ be a SCI term and K its event-logic representations, and $\vdash \mathcal{C}[-] :$ com a SCI program context with a θ-typed hole with L its event-logic representation. The interaction $K \bullet L$ has no unsafe traces.*

Let us define an analogous notion of the next-play set for games, where M is an SCI term, as $\mathrm{next}_M(u) = \{n \mid u \cdot n \in [\![M]\!]^{\circledast}\}$, and $\mathrm{next}^P_M(u)$ the obvious restriction to P-moves, and $\mathrm{next}^{P,\theta}_M(u)$ the obvious restriction to P-moves in sub-arena θ (and likewise for O-moves, and for other sub-arenas). Then both correctness and program safety follow as corollaries of this more general theorem:

Theorem 5.3 *Let $\Gamma \vdash M : \theta$ be an SCI term with event-logic representation K, and let u be a finitely long trace on K which is also a unit-bounded justified sequence in $[\![M]\!]^{\circledast}$. Then* $\mathrm{next}^o_K(u) = \mathrm{next}^P_M(u)$, *and* $\mathrm{next}^i_K(u) \supseteq \mathrm{next}^O_M(u)$.

The proof of this theorem is given in the Appendix.

References

[1] Abramsky, S., *Retracing some paths in process algebra*, in: *CONCUR*, 1996, pp. 1–17.

[2] Abramsky, S., R. Jagadeesan and P. Malacaria, *Full abstraction for PCF.*, Inf. Comput. **163** (2000), pp. 409–470.

[3] Abramsky, S. and G. McCusker, *Game semantics*, in: *Proceedings of the 1997 Marktoberdorf Summer School*, 1997.

[4] Ghica, D. R., *Geometry of Synthesis: a structured approach to VLSI design*, in: *POPL*, 2007, pp. 363–375.

[5] Ghica, D. R., *Applications of game semantics: From software analysis to hardware synthesis*, in: *LICS*, 2009, pp. 17–26.

[6] Ghica, D. R., *Function interface models for hardware compilation: Types, signatures, protocols*, CoRR **abs/0907.0749** (2009).

[7] Ghica, D. R. and G. McCusker, *The regular-language semantics of second-order Idealized Algol.*, Theor. Comput. Sci. **309** (2003), pp. 469–502.

[8] Ghica, D. R. and A. Murawski, *Angelic semantics of fine-grained concurrency*, Annals of Pure and Applied Logic **151** (2008), pp. 89–114.

[9] Ghica, D. R., A. S. Murawski and C.-H. L. Ong, *Syntactic control of concurrency*, Theor. Comput. Sci. **350** (2006), pp. 234–251.

[10] Hauck, S., *Asynchronous design methodologies: an overview*, Proceedings of the IEEE **83** (1995), pp. 69–93.

[11] Hyland, J. M. E. and C.-H. L. Ong, *On full abstraction for PCF: I, II, and III.*, Inf. Comput. **163** (2000), pp. 285–408.

[12] Joyal, A., R. Street and D. Verity, *Traced monoidal categories*, Mathematical Proceedings of Cambridge Philosophical Society **119** (1996), pp. 447–468.

[13] McCusker, G., *A fully abstract relational model of Syntactic Control of Interference*, in: *CSL*, 2002, pp. 247–261.

[14] Menaa, M. N., *Towards a synchronous game semantics*, workshop on Games for Logic and Programming Languages IV, York, 2009.

[15] Miller, R. E., "Sequential Circuits," Wiley, NY, 1965.

[16] Moradi, A., M. T. M. Shalmani and M. Salmasizadeh, *Dual-rail transition logic: A logic style for counteracting power analysis attacks*, Computers & Electrical Engineering **35** (2009), pp. 359 – 369.

[17] O'Hearn, P. W., J. Power, M. Takeyama and R. D. Tennent, *Syntactic control of interference revisited.*, Theor. Comput. Sci. **228** (1999), pp. 211–252.

[18] Reynolds, J. C., *Syntactic control of interference.*, in: *POPL*, 1978, pp. 39–46.

[19] Sparsø, J. and S. Furber, editors, "Principles of Asynchronous Circuit Design: A Systems Perspective," European Low-Power Initiative for Electronic System Design, Kluwer Academic Publishers, 2001.

[20] Sutherland, I. E., *Micropipelines*, Commun. ACM **32** (1989), pp. 720–738, turing Award Paper.

[21] Yun, K. Y. and D. L. Dill, *Automatic synthesis of extended burst-mode circuits I*, IEEE Trans. on CAD of Integrated Circuits and Systems **18** (1999), pp. 101–117.

A Proof of Theorem 5.3

By structural induction on syntax.

First, note that a unit-bounded justified sequence in $[\![M]\!]^{\circledast}$ must consist of any number of unit-bounded plays (including ϵ) in $[\![M]\!]$ with their initial questions answered, followed by one more unit-bounded play in $[\![M]\!]$; this is because unit-boundedness implies that the initial question must be answered before it can be asked again, and a play cannot contain any moves past the answer of the initial question, so the iterated shuffle cannot lead to any overlapping plays. This fact is used to calculate most of the definitions for $[\![M]\!]^{\circledast}$ given in the base cases below from the definitions for $[\![M]\!]$ given above.

Base cases:

- $[\![\mathsf{skip}]\!]^{\circledast} = \mathit{prefix}\,((qa)^*) = [\![\mathit{WIRE}]\!]$; because the SCI term and its event-logic representation have identical definitions in this case, the next-action and next-play sets must be identical, and thus certainly obey the (weaker) requirements $\mathrm{next}^o_K(u) = \mathrm{next}^P_M(u)$ and $\mathrm{next}^i_K(u) \supseteq \mathrm{next}^O_M(u)$.
- $\mathsf{0}$ and $\mathsf{1}$ are identical to skip except for the addition of one output/P-move that can never happen (t and f respectively); because that move can never happen, it will never be in a next-action/next-play set, so the proof for skip works here too.
- $[\![\mathsf{seq}]\!]^{\circledast} = \mathit{prefix}\left((q''qaq'a'a'')^*\right)$, and $[\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] = \mathit{prefix}\left((q''q)^* \otimes (aq')^* \otimes (a'a'')^*\right)$. Preceding a play in $[\![\mathsf{seq}]\!]^{\circledast}$ by any number of repetitions of $q''qaq'a'a''$ (the only play in seq with its initial question answered) will have no effect on any of the next-action sets (because $q''qaq'a'a'' \cdot \left((q''q)^* \otimes (aq')^* \otimes (a'a'')^*\right) \subseteq (q''q)^* \otimes (aq')^* \otimes (a'a'')^*$), and so it's possible to prove this simply by considering all prefixes of $q''qaq'a'a''$; for $u = \epsilon$, q'', $q''q$, $q''qa$, $q''qaq'$, $q''qaq'a'$, $q''qaq'a'a''$ respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ in the case of seq are both $\emptyset$, $\{q\}$, $\emptyset$, $\{q'\}$, $\emptyset$, $\{a''\}$, $\emptyset$ respectively, and considering $\mathrm{next}^i_K(u)$ and $\mathrm{next}^O_M(u)$ gives $\{q'', a, a'\} \supseteq \{q''\}$, $\{a, a'\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{a\}$, $\{q'', a'\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{a'\}$, $\{q'', a\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{q''\}$ respectively, proving the theorem in this case.
- $[\![\mathsf{der}]\!]^{\circledast} = \mathit{prefix}\left((q'qtt' + q'qff')^*\right)$, and $[\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] = \mathit{prefix}\left((q'q)^* \otimes (tt')^* \otimes (ff')^*\right)$. Preceding a play in $[\![\mathsf{der}]\!]^{\circledast}$ by any number of repetitions of $q''qtt'$ and/or $q''qff'$ (the only plays in der with their initial questions answered) will have no effect on any of the next-action sets (because $(q'qtt' + q'qff') \cdot \left((q'q)^* \otimes (tt')^* \otimes (ff')^*\right) \subseteq (q'q)^* \otimes (tt')^* \otimes (ff')^*$), and so it's possible to prove this simply by considering all prefixes of $q'qtt' + q'qff'$; for $u = \epsilon$, q', $q'q$, $q'qt$, $q'qtt'$, $q'qf$, $q'qff'$ respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ in the case of der are both $\emptyset$, $\{q\}$, $\emptyset$, $\{t'\}$, $\emptyset$, $\{f'\}$, $\emptyset$ respectively, and considering $\mathrm{next}^i_K(u)$ and $\mathrm{next}^O_M(u)$ gives $\{q', t, f\} \supseteq \{q'\}$, $\{t, f\} \supseteq \emptyset$, $\{q', t, f\} \supseteq \{t\}$, $\{q', f\} \supseteq \emptyset$, $\{q', t, f\} \supseteq \{f\}$, $\{q', t\} \supseteq \emptyset$, $\{q', t, f\} \supseteq \{q'\}$ respectively, proving the theorem in this case.
- $[\![\mathsf{asg}]\!]^{\circledast} = \mathit{prefix}\left((q''q't'w_taa'' + q''q'f'w_faa'')^*\right)$, and $[\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] \otimes [\![\mathit{WIRE}]\!] = \mathit{prefix}\left((q''q')^* \otimes (t'w_t)^* \otimes (f'w_f)^* \otimes (aa'')^*\right)$. Preceding a play in $[\![\mathsf{asg}]\!]^{\circledast}$ by any number of repetitions of $q''q't'w_taa''$ and/or $q''q'f'w_faa''$ (the only plays in asg with their initial questions answered) will have no effect on any of the next-action sets (because $(q''q't'w_taa'' + q''q'f'w_faa'') \cdot \left((q''q')^* \otimes (t'w_t)^* \otimes (f'w_f)^* \otimes (aa'')^*\right) \subseteq (q''q')^* \otimes (t'w_t)^* \otimes (f'w_f)^* \otimes (aa'')^*$), and so it's possible to prove this simply by considering all prefixes of $q''q't'w_taa'' + q''q'f'w_faa''$; for $u = \epsilon$, q'', $q''q'$, $q''q't'$, $q''q't'w_t$, $q''q't'w_ta$, $q''q't'w_taa''$, $q''q'f'$, $q''q'f'w_f$, $q''q'f'w_fa$, $q''q'f'w_faa''$ respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ in the case of asg are both $\emptyset$, $\{q'\}$, $\emptyset$, $\{w_t\}$, $\emptyset$, $\{a''\}$, $\emptyset$, $\{w_f\}$, $\emptyset$, $\{a''\}$, $\emptyset$ respectively, and considering $\mathrm{next}^i_K(u)$ and $\mathrm{next}^O_M(u)$ gives $\{q'', t', f', a\} \supseteq \{q''\}$, $\{t', f', a\} \supseteq \emptyset$, $\{q'', t', f', a\} \supseteq \{t', f'\}$, $\{q'', f', a\} \supseteq \emptyset$, $\{q'', t', f', a\} \supseteq \{a\}$, $\{q'', t', f'\} \supseteq \emptyset$, $\{q'', t', f', a\} \supseteq \{q''\}$, $\{q'', t', a\} \supseteq \emptyset$, $\{q'', t', f', a\} \supseteq \{a\}$, $\{q'', t', f'\} \supseteq \emptyset$, $\{q'', t', f', a\} \supseteq \{q''\}$ respectively, proving the theorem in this case.

- $[\![\mathsf{if}]\!]^{\circledast} = \mathit{prefix}\left((q'''qtq'a'a''' + q'''qfq''a''a''')^*\right)$, and $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![XOR]\!] = \mathit{prefix}\left((q'''q)^* \otimes (tq')^* \otimes (fq'')^* \otimes (a'a''' + a''a''')^*\right)$. Preceding a play in $[\![\mathsf{if}]\!]^{\circledast}$ by any number of repetitions of $q'''qtq'a'a'''$ and/or $q'''qfq''a''a'''$ (the only plays in if with their initial questions answered) will have no effect on any of the next-action sets (because $(q'''qtq'a'a''' + q'''qfq''a''a''') \cdot \left((q'''q)^* \otimes (tq')^* \otimes (fq'')^* \otimes (a'a''' + a''a''')^*\right) \subseteq (q'''q)^* \otimes (tq')^* \otimes (fq'')^* \otimes (a'a''' + a''a''')^*$), and so it's possible to prove this simply by considering all prefixes of $q'''qtq'a'a''' + q'''qfq''a''$; for $u = \epsilon$, q''', $q'''q$, $q'''qt$, $q'''qtq'$, $q'''qtq'a'$, $q'''qtq'a'a'''$, $q'''qf$, $q'''qfq''$, $q'''qfq''a''$, $q'''qfq''a''a'''$ respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ in the case of if are both $\emptyset$, $\{q\}$, $\emptyset$, $\{q'\}$, $\emptyset$, $\{a'''\}$, $\emptyset$, $\{q''\}$, $\emptyset$, $\{a'''\}$, $\emptyset$ respectively, and considering $\mathrm{next}^i_K(u)$ and $\mathrm{next}^O_M(u)$ gives $\{q''', t, f, a', a''\} \supseteq \{q'''\}$, $\{t, f, a', a''\} \supseteq \emptyset$, $\{q''', t, f, a', a''\} \supseteq \{t, f\}$, $\{q''', f, a', a''\} \supseteq \emptyset$, $\{q''', t, f, a', a''\} \supseteq \{a'\}$, $\{q''', t, f\} \supseteq \emptyset$, $\{q''', t, f, a', a''\} \supseteq \{q'''\}$, $\{q''', t, a', a''\} \supseteq \emptyset$, $\{q''', t, f, a', a''\} \supseteq \{a''\}$, $\{q''', t, f\} \supseteq \emptyset$, $\{q''', t, f, a', a''\} \supseteq \{q'''\}$ respectively, proving the theorem in this case.

- $[\![\mathsf{or_l}]\!]^{\circledast} = \mathit{prefix}\left((q''qtt'' + q''qfq't't'' + q''qfq'f'f'')^*\right)$, and $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![XOR]\!] = \mathit{prefix}\left((q''q)^* \otimes (fq')^* \otimes (f'f'')^* \otimes (tt'' + t't'')^*\right)$. Preceding a play in $[\![\mathsf{or_l}]\!]^{\circledast}$ by any number of repetitions of $q''qtt''$, $q''qfq't't''$ and/or $q''qfq'f'f''$ (the only plays in $\mathsf{or_l}$ with their initial questions answered) will have no effect on any of the next-action sets (because $(q''qtt'' + q''qfq't't'' + q''qfq'f'f'') \cdot \left((q''q)^* \otimes (fq')^* \otimes (f'f'')^* \otimes (tt'' + t't'')^*\right) \subseteq (q''q)^* \otimes (fq')^* \otimes (f'f'')^* \otimes (tt'' + t't'')^*$), and so it's possible to prove this simply by considering all prefixes of $q''qtt'' + q''qfq't't'' + q''qfq'f'f''$; for $u = \epsilon$, q'', $q''q$, $q''qt$, $q''qtt''$, $q''qf$, $q''qfq'$, $q''qfq't'$, $q''qfq't't''$, $q''qfq'f'$, $q''qfq'f'f''$, respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ are both $\emptyset$, $\{q\}$, $\emptyset$, $\{t''\}$, $\emptyset$, $\{q''\}$, $\emptyset$, $\{t''\}$, $\emptyset$, $\{f''\}$, $\emptyset$ respectively, and considering $\mathrm{next}^i_K(u)$ and $\mathrm{next}^O_M(u)$ gives $\{q'', t, f, t', f'\} \supseteq \{q''\}$, $\{t, f, t', f'\} \supseteq \emptyset$, $\{q'', t, f, t', f'\} \supseteq \{t, f\}$, $\{q'', f, f'\} \supseteq \emptyset$, $\{q'', t, f, t', f'\} \supseteq \{q''\}$, $\{q'', t, t', f'\} \supseteq \emptyset$, $\{q'', t, f, t', f'\} \supseteq \{t', f'\}$, $\{q'', f, f'\} \supseteq \emptyset$, $\{q'', t, f, t', f'\} \supseteq \{q''\}$, $\{q'', t, t', f\} \supseteq \emptyset$, $\{q'', t, f, t', f'\} \supseteq \{q''\}$ respectively, proving the theorem in this case. Note that an identical argument applies to $\mathsf{and_l}$, which is merely a relabeled version of $\mathsf{or_l}$.

- For par, note that unlike the other cases so far, the given $q''qq'aa'a''$ is not saturated already; saturating it gives $q''qq'aa'a'' + q''q'qaa'a'' + q''qq'a'aa'' + q''q'qa'aa'' + q''qaq'a'a'' + q''q'a'qaa''$, and $[\![\mathsf{par}]\!]^{\circledast}$ is the prefix-closure of the repeat of that. The circuit for it is $[\![FORK]\!] \otimes [\![C]\!] = (q''qq' + q''q'q)^* \otimes (aa'a'' + a'aa'')^*$. As before, preceding a play in $[\![\mathsf{par}]\!]^{\circledast}$ with repetitions of any of the six plays in par which have their initial questions answered will have no effect on any of the next-action sets (for the same reason as in the previous cases). As a result, all that is needed is to check all 26 cases; to reduce the number somewhat, note that both $[\![\mathsf{par}]\!]^{\circledast}$and $[\![FORK]\!] \otimes [\![C]\!]$ are unchanged by replacing q with q', a with a', and vice versa, so without loss of generality it can be assumed that $q''q'$ does not start the trace, leaving only 14 cases. For $u = \epsilon$, q'', $q''q$, $q''qq'$, $q''qq'a$, $q''qq'aa'$, $q''qq'aa'a''$, $q''qq'a'$, $q''qq'a'a$, $q''qq'a'aa''$, $q''qa$, $q''qaq'$, $q''qaq'a'$, $q''qaq'a'a''$ respectively, $\mathrm{next}^o_K(u)$ and $\mathrm{next}^P_M(u)$ in the case of par are both $\emptyset$,

$\{q, q'\}$, $\{q'\}$, $\emptyset$, $\emptyset$, $\{a''\}$, $\emptyset$, $\emptyset$, $\{a''\}$, $\emptyset$, $\{q'\}$, $\emptyset$, $\{a''\}$, $\emptyset$ respectively, and considering $\text{next}^i_K(u)$ and $\text{next}^O_M(u)$ gives $\{q'', a, a'\} \supseteq \{q''\}$, $\{a, a'\} \supseteq \emptyset$, $\{a, a'\} \supseteq \{a\}$, $\{q'', a, a'\} \supseteq \{a, a'\}$, $\{q'', a'\} \supseteq \{a'\}$, $\{q''\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{q''\}$, $\{q'', a\} \supseteq \{a\}$, $\{q''\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{q''\}$, $\{a'\} \supseteq \emptyset$, $\{q'', a'\} \supseteq \{a'\}$, $\{q''\} \supseteq \emptyset$, $\{q'', a, a'\} \supseteq \{q''\}$ respectively, proving the theorem in this case.

- $[\![\mathsf{while}]\!]^{\circledast} = \textit{prefix}\left(\left(q''q\,(tq'a'q)^*\,fa''\right)^*\right)$, and $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![XOR]\!] = \textit{prefix}\left((tq')^* \otimes (fa'')^* \otimes (q''q + a'q)^*\right)$. Inserting $tq'a'q$ immediately after any occurence of q that does not have a preceding t later than a preceding q' has no effect on the next-play set of a justified sequence in $[\![\mathsf{while}]\!]^{\circledast}$, because for each justified sequence in $[\![\mathsf{while}]\!]^{\circledast}$ which has that an occurence of that subsequence, there is another sequence in $[\![\mathsf{while}]\!]^{\circledast}$ that is identical except that it does not contain that occurence; likewise, it has no effect on the next-action set of a trace in $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![XOR]\!]$, for the same reason. The only plays in $[\![\mathsf{while}]\!]^{\circledast}$ that contain $tq'a'q$ have it in such a position; therefore, without loss of generality, it can be assumed that u contains no occurences of $tq'a'q$. Preceding a play in $[\![\mathsf{while}]\!]^{\circledast}$ by any number of repetitions of $q''qfa''$ (the only play in while with its initial question answered, given this assumption) will have no effect on any of the next-action sets (because $q''qfa'' \cdot \left((tq')^* \otimes (fa'')^* \otimes (q''q + a'q)^*\right) \subseteq (tq')^* \otimes (fa'')^* \otimes (q''q + a'q)^*$). Therefore, the cases that need consideration are all prefixes of $q''q\,(tq'a'q)^*\,fa''$ that do not contain $tq'aq$, which is the prefixes of $q''qfa'' + q''qtq'a'$; for $u = \epsilon$, q'', $q''q$, $q''qf$, $q''qfa''$, $q''qt$, $q''qtq'$, $q''qtq'a'$ respectively, $\text{next}^o_K(u)$ and $\text{next}^P_M(u)$ in the case of while are both $\emptyset$, $\{q\}$, $\emptyset$, $\{a''\}$, $\emptyset$, $\{q'\}$, $\emptyset$, $\{q\}$ respectively, and considering $\text{next}^i_K(u)$ and $\text{next}^O_M(u)$ gives $\{q'', t, f, a'\} \supseteq \{q''\}$, $\{t, f\} \supseteq \emptyset$, $\{q'', t, f, a'\} \supseteq \{t, f\}$, $\{q'', t, a'\} \supseteq \emptyset$, $\{q'', t, f, a'\} \supseteq \{q''\}$, $\{q'', f, a'\} \supseteq \emptyset$, $\{q'', t, f, a'\} \supseteq \{a'\}$, $\{t, f\} \supseteq \emptyset$ respectively, proving the theorem in this case.
- $[\![\mathsf{newvar}]\!]^{\circledast} = \textit{prefix}\left(\left(q''q'\,(w_ta\,(qt)^* + w_fa\,(qf)^*)^*\,a'a''\right)^*\right)$, and $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![CALL]\!] = \textit{prefix}\left((q''q')^* \otimes (a'a'')^* \otimes (w_ta\,(qt)^* + w_fa\,(qf)^*)^*\right)$. In order to observe that $\text{next}_K(u) \supseteq \text{next}_M(u)$ in the case of newvar, note that $[\![WIRE]\!] \otimes [\![WIRE]\!] \otimes [\![CALL]\!] \supseteq [\![\mathsf{newvar}]\!]^{\circledast}$ (which is obvious given the definitions above), and so the set of possible m such that $u{\cdot}m \in [\![\mathsf{newvar}]\!]^{\circledast}$ is a subset of the set of possible m such that $u{\cdot}m \in [\![WIRE]\!]{\otimes}[\![WIRE]\!]{\otimes}[\![CALL]\!]$; this immediately implies that $\text{next}^o_K(u) \supseteq \text{next}^P_M(u)$ and $\text{next}^i_K(u) \supseteq \text{next}^O_M(u)$ in this case. All that is left to show is that additionally, $\text{next}^o_K(u) \subseteq \text{next}^P_M(u)$ in this case. To do this, assume for contradiction that there is some P-move/output m such that $m \in \text{next}^o_K(u)$ but $m \notin \text{next}^P_M(u)$, i.e. that $u{\cdot}m \in [\![WIRE]\!]{\otimes}[\![WIRE]\!]{\otimes}[\![CALL]\!]$ but $u \cdot m \notin [\![\mathsf{newvar}]\!]^{\circledast}$. In $[\![\mathsf{newvar}]\!]^{\circledast}$, each q' must be immediately preceded by q'', each a'' by a', each f or t by q, and each a by w_f or w_t; so this must be true of u in particular. Therefore, each of u, $u \restriction \{q'', q'\}$, $u \restriction \{a', a''\}$, and $u \restriction \{w_t, w_f, a, q, t, f\}$ must consist of alternating inputs and outputs, starting with an input, and so in each of those four traces, either the number of inputs and outputs is equal, or there is one more input and output. Because $u \restriction \{q'', q'\}$, $u \restriction \{a', a''\}$, and $u \restriction \{w_t, w_f, a, q, t, f\}$ together make up the whole of u, only one of those three traces can contain more inputs than outputs, and therefore only

one of those three traces can end with an input; likewise, that trace (if it exists) must be the one that contains the last event in u, because otherwise its last event would have to be immediately followed by an output in a different one of those traces (barred by the definition of $[\![\textsf{newvar}]\!]^{\circledast}$) or by an input (barred by the fact that inputs and outputs must alternate in u). For both $WIRE$ and $CALL$, it is impossible to have two outputs in a row, and so none of $u \cdot m \upharpoonright \{q'', q'\}$, $u \cdot m \upharpoonright \{a', a''\}$, or $u \cdot m \upharpoonright \{w_t, w_f, a, q, t, f\}$ can end with two outputs; because m is by assumption an output, it must therefore be in the same one of those traces which contains the last event in u. As a result, the only possibilities for the last two events in u are $q''q'$, $a'a''$, $w_t a$, $w_f a$ (all of which violate the assumption that $u \cdot m \notin [\![\textsf{newvar}]\!]^{\circledast}$, because in each of those 4 cases, if u is in that set, $u \cdot m$ must be also), $w_t t$, $w_t f$, $w_f t$, $w_f f$, qa (which all violate the fact that $u \upharpoonright \{w_t, w_f, a, q, t, f\} = [\![CALL]\!]$, as those subsequences cannot be generated anywhere by its definition), or qt or qf. Note that qt cannot occur in $[\![CALL]\!]$ unless w_t has occurred, and later than the last w_f (if any); however, for all justified sequences in $[\![\textsf{newvar}]\!]^{\circledast}$ which contain a w_t that is later than any occurrence of w_f, and which end in q, the sequence formed by appending t to it is also in $[\![\textsf{newvar}]\!]^{\circledast}$, violating the assumption that $u \cdot m \notin [\![\textsf{newvar}]\!]^{\circledast}$; and a similar argument applies to qf. As all 11 cases have been shown to be impossible, the premise must also be impossible, so in other words there is no m such that $m \in \text{next}^{o}_{K}(u)$ but $m \notin \text{next}^{P}_{M}(u)$, and so $\text{next}^{o}_{K}(u) \subseteq \text{next}^{P}_{M}(u)$. Together with the earlier results, this proves the theorem for the case of newvar.

- In the case of δ_θ, the SCI strategy is merely copycat. Assume without loss of generality that an initial move in θ' (say I') has arrived more recently than any initial move in I''. Define an O-move m in θ to be *unmatched* in u if it appears later than its *copy* m' does in u; likewise, define an O-move m' in θ' to be unmatched in u if it appears later than its copy m does in u. (In other words, a move is unmatched in a play if some occurrence of it has been played but not yet copied). Then $\text{next}^{P}_{M}(u)$ is, by definition, the set of all copies of unmatched O-moves in u.

  To prove that $\text{next}^{o}_{K}(u) = \text{next}^{P}_{M}(u)$ in this case, it suffices to show that $\text{next}^{o}_{K}(u)$ obeys the same definition; this is accomplished by showing that the definition is the same when restricted to non-initial outputs in θ (from the point of view of δ_θ itself, these are inputs from the point of view of the circuit being diagonalised), to initial outputs in θ, to outputs in θ', and to outputs in θ'':

  · A non-initial output m in θ is in $\text{next}^{P}_{M}(u)$ if and only if m' is unmatched in u. Because it is impossible for m'' to also be unmatched in u (because all moves in θ'' must occur before any questions that (directly or indirectly) justify them are answered, thus before I'' is answered, and thus before the most recent I' occurs — I'' must be answered before I' can occur because θ is unit-bounded), then m in θ is in $\text{next}^{P}_{M}(u)$ if and only if m' is unmatched in u but m'' isn't. However, the definition of XOR implies that a non-initial output m in θ is in $\text{next}^{o}_{K}(u)$ if and only if m' is unmatched in u but m'' isn't, which is the same definition; thus $\text{next}^{o}_{K}(u) = \text{next}^{P}_{M}(u)$ in this subcase.

- An initial output I in θ is in $\mathrm{next}^P_M(u)$ if and only if I' is unmatched in u. For I' to be unmatched, all questions in u before the I' must have been answered (due to unit-boundedness of θ), and no moves in u cannot have happened since (no non-initial move could occur in θ because it could not be justified by anything, as there are no unanswered questions to do the justifying; no initial move other than I could occur in θ because that would imply that there were two pending initial moves in θ', violating unit-boundedness, or that there was a pending initial move in both θ' and θ'', violating the arena definition for δ_θ). An initial output I in θ is in $\mathrm{next}^o_K(u)$ only if I' is unmatched, because it can only happen if all the $CALL$ elements have output R_i more recently than the previous occurence of I (by the definition of C), $CALL$ will not output R_i except in response to I' or I'' (by definition), and the most recent I'' cannot have occured before the most recent I' (by assumption). Likewise, an initial output I in is in $\mathrm{next}^o_K(u)$ if I' is unmatched; taking v to be some interaction of the $CALL$, XOR and C elements such that $v \upharpoonright \theta \to \theta' \times \theta'' = u$, then $v \cdot R_1 R_2 \cdots R_i I$ is an interaction that is in K, which is $u \cdot I$ when restricted to the arena of M, and which is safe (because with the prefix of v ending with the last time I was played and/or the empty prefix, the C gate's next-event set contained all its inputs by definition, and no inputs to or outputs from the C gate have happened in the rest of v, so its next-event state is still the same). So $\mathrm{next}^o_K(u) = \mathrm{next}^P_M(u)$ in this subcase.
- An output in θ' or θ'' is in $\mathrm{next}^P_M(u)$ if and only if it is in θ' and unmatched in u. Such a move can only be in $\mathrm{next}^o_K(u)$ if I (the copy of the initial move I') has occurred (no move m can occur in θ without an initial move first occuring in that arena, as it could not otherwise be justified), more recently than I' (I will not occur in M except as the copy of I' or I'', and thus not in u in particular, and I' has happened more recently by assumption), which has occurred more recently than any I which is a copy of I'' (all occurences of I'' must have been answered before I' can occur due to θ being unit-bounded, and they could not be answered without I having occurred before the answer to I'' occuring, as moves cannot happen in θ at all except as the answer to some initial move). But I does not occur in K unless all the $CALL$ elements have output R_i more recently than the previous occurence of I (by the definition of C), $CALL$ will not output R_i except in response to I' or I'', and the most recent R_i is a response to the most recent occurrence of I' or I'' (by definition). In other words, each $CALL$ block must, at the time m' or m'' occurs, be currently have m' in its next-action set if u has an unmatched m; so an output in θ' or θ'' is in $\mathrm{next}^o_K(u)$ if and only if it is in θ' and unmatched in u, proving that $\mathrm{next}^o_K(u) = \mathrm{next}^P_M(u)$ in this subcase.

This shows that $\mathrm{next}^o_K(u) = \mathrm{next}^P_M(u)$ in this case; it remains to prove that, additionally, $\mathrm{next}^i_K(u) \supseteq \mathrm{next}^O_M(u)$. It is impossible for an O-move m unmatched in u to be in $\mathrm{next}^O_M(u)$ (otherwise the move would occur twice in θ' or θ'' without occuring in θ, it cannot be answered in θ' or θ'' without being answered in θ due to copycat, and it cannot be answered in θ without occuring in θ; so it would occur twice in the same arena without being answered, which violates unit-boundedness); therefore, any counterexample must be an O-move that is not unmatched in u, yet is in $\mathrm{next}^{O}_{M}(u)$ but not $\mathrm{next}^{i}_{K}(u)$. Neither I' nor I'' can be such a counterexample; these initial moves cannot occur without all questions in all three arenas having been answered (due to unit-boundedness), which implies that all no move at all in u can be unmatched (if one was, then it would mean that some question was answered in θ but unanswered in θ' or θ'', or vice versa), and this means that for each $CALL$ element, R_i has occurred as many times as $I' + I''$, and $P'_i + P''_i$ has occurred as many times as P_i, thus meaning that I' and I'' are both in the next-event set of each $CALL$ element and so in $\mathrm{next}^{i}_{K}(u)$ in this case. Likewise, no O-move m' in θ' can be such a counterexample; as because it is matched, and all moves in θ'' must be matched for m' to occur (as if one were unmatched, it would mean both m' and the move in m'' were justified somehow, which would imply a pending initial move in both θ' and θ'', violating the arena definition for δ_θ), the last element of $u \restriction \{m, m', m''\}$ must be m, and thus m' is in the next-event set of the relevant XOR element and so in $\mathrm{next}^{i}_{K}(u)$ in this case. (Likewise, for O-moves in θ''.) Finally, no O-move m in θ can be such a counterexample; the relevant $CALL$ element must have m in its next-action set, because the only cases in which it doesn't are if it m had occured more recently than m' or m'' (impossible because m is matched in u), or if I' or I'' had occurred more recently than R_i and thus more recently than I (impossible because then either m could not be justified, or else two initial moves on θ would have to be pending at once). This means that all possible counterexamples have been excluded, and so $\mathrm{next}^{i}_{K}(u) \supseteq \mathrm{next}^{O}_{M}(u)$; because it has already been shown that $\mathrm{next}^{o}_{K}(u) = \mathrm{next}^{P}_{M}(u)$, the theorem is true in this case.

Inductive steps:

- *Function application*: Let $\Gamma_1 \vdash M_1 : \theta' \to \theta$ and $\Gamma_2 \vdash M_2 : \theta'$ be unit-bounded SCI terms with event-logic representations K_1 and K_2 respectively such that for all traces u_i which are a trace on K_i which is also a unit-bounded justified sequence in $[\![M_i]\!]^{\circledast}$, $\mathrm{next}^{o}_{K_i}(u_i) = \mathrm{next}^{P}_{M_i}(u_i)$ and $\mathrm{next}^{i}_{K_i}(u_i) \supseteq \mathrm{next}^{O}_{M_i}(u_i)$. If $M = M_1(M_2)$, then $[\![M]\!] = [\![M_2]\!]; [\![M_1]\!]$ (where the arenas compose correctly, because $\theta' = \langle\emptyset,\emptyset,\emptyset\rangle \to \theta'$ by definition). Likewise, $[\![M]\!]^{\circledast} = ([\![M_2]\!]; [\![M_1]\!])^{\circledast} = \{v^{\circledast} \restriction \theta | v \restriction \theta' \in [\![M_2]\!]^{\circledast} \wedge v \restriction \theta' \to \theta \in [\![M_1]\!]\} = \{v \restriction \theta | v \restriction \theta' \in [\![M_2]\!]^{\circledast\circledast} \wedge v \restriction \theta' \to \theta \in [\![M_1]\!]^{\circledast}\} = [\![M_2]\!]^{\circledast}; [\![M_1]\!]^{\circledast}$. Each of M_1 and M_2 must be able to accept all O-moves on θ that the other can produce (as P-moves from its point of view), because if there were some play that could be produced by one but not accepted by the other, then either the play would be illegal or an SCI term would not be O-complete, neither of which is allowed; in other words, $\mathrm{next}^{O,\theta'}_{M_1}(v) \supseteq \mathrm{next}^{P,\theta'}_{M_2}(v)$ (and likewise if M_1 and M_2 are exchanged) for all interactions v between M_1 and M_2; taking $u_1 = v \restriction \theta' \to \theta$ and $u_2 = v \restriction \theta'$ gives $\mathrm{next}^{o,\theta'}_{K_1}(u_1) = \mathrm{next}^{P,\theta'}_{M_1}(u_1) \subseteq \mathrm{next}^{O,\theta'}_{M_2}(u_2) \subseteq \mathrm{next}^{i,\theta'}_{K_2}(u_2)$ (and likewise if M_1/K_1 and M_2/K_2 are exchanged). This implies that all interactions between K_2 and K_1 are safe.

  It is required to prove that given $K = K_2; K_1$ and $M = M_1(M_2)$, $\mathrm{next}^{o}_{K}(u) = \mathrm{next}^{P}_{M}(u)$, and $\mathrm{next}^{i}_{K}(u) \supseteq \mathrm{next}^{O}_{M}(u)$.

Assume for contradiction that there is an interaction $v \cdot w \cdot m$ in exactly one of the sets $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$ (with v and w interactions, and m a move), all moves in w are in θ', the last move in v is in θ, v is in both $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$, and at least one of the following statements is true: m is not in θ, m is a P-move in M_1, or the last move in v is an O-move in M_1 and $v \cdot w \cdot m \notin K_2 \bullet K_1$. Without loss of generality, assume $v \cdot w \cdot m$ is the shortest interaction with this property. $v \cdot w$ must be in at least one of the sets $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$, because it is a prefix of $v \cdot w \cdot m$ which is also in at least one of those sets, and those sets are prefix-closed. However, it is not in both; if it were, then:

· if m is a P-move in M_1, then m must be in exactly one of $\mathrm{next}^{o}_{K_1}(v \cdot w)$, and $\mathrm{next}^{P}_{M_1}(v \cdot w)$, violating the assumption that $\mathrm{next}^{o}_{K_1}(u_1) = \mathrm{next}^{P}_{M_1}(u_1)$;
· if m is an O-move in M_1 but not in in θ (and so a P-move in M_2), then m must be in exactly one of $\mathrm{next}^{o}_{K_2}(v \cdot w \upharpoonright \theta')$, and $\mathrm{next}^{P}_{M_2}(v \cdot w \upharpoonright \theta')$, violating the assumption that $\mathrm{next}^{o}_{K_2}(u_2) = \mathrm{next}^{P}_{M_2}(u_2)$;
· if $v \cdot w \cdot m \notin K_2 \bullet K_1$, then m must be in $\mathrm{next}^{O}_{M_1}(v \cdot w)$ but not $\mathrm{next}^{i}_{K_1}(v \cdot w)$, violating the assumption that $\mathrm{next}^{i}_{K_1}(u_1) \supseteq \mathrm{next}^{O}_{M_1}(u_1)$

This implies that $v \cdot w$ is in exactly one of those sets. Suppose that w has positive length; then take $w = w' \cdot m'$; because $v \cdot w' \cdot m'$ is in exactly one of the sets $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$, and it is shorter than $v \cdot w \cdot m$, then it must be the case that m' is in θ, violating the assumption that all moves in w are in θ'. However, w cannot have zero length either, because that violates the assumption that v is in both $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$.

The conclusion is that no shortest trace $v \cdot w \cdot m$ with the above properties can exist, and thus no such trace can exist (because if any did, one would be shortest). Now, if $\mathrm{next}^{o}_{K}(u) \neq \mathrm{next}^{P}_{M}(u)$, that would imply that some trace $u \cdot m$ existed (with m a P-move) which was in exactly one of the sets $[\![M_2]\!]^{\circledast}; [\![M_1]\!]^{\circledast}$ and $K_2; K_1$, which implies that $u \cdot m = v \cdot w \cdot m \upharpoonright \theta$ for some interaction $v \cdot w \cdot m \in u_2 \bullet u_1$ where the last move in v is the last move in u, $v \upharpoonright \theta = u$ and v is in both $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$ (because otherwise u could not be in both the compositions) but $v \cdot w \cdot m$ is not (or $u \cdot m$ would be in both compositions); but this implies that w consists entirely of moves in θ' (if it contained a move in θ, then either the last move in u is in w not v, or $v \cdot w \cdot m \upharpoonright \theta \neq u \cdot m$), and so this would give a $v \cdot w \cdot m$ with the above properties, which is a contradiction. Likewise, if $\mathrm{next}^{i}_{K}(u) \not\supseteq \mathrm{next}^{O}_{M}(u)$, that would imply that some trace $u \cdot m$ existed (with the last move in u an O-move not in $K_2; K_1$) which was in exactly one of the sets $[\![M_2]\!]^{\circledast}; [\![M_1]\!]^{\circledast}$ and $K_2; K_1$, which implies that $u \cdot m = v \cdot w \cdot m \upharpoonright \theta$ for some interaction $v \cdot w \cdot m \in u_2 \bullet u_1$ where the last move in v is the last move in u, $v \upharpoonright \theta = u$ and v is in both $[\![M_2]\!]^{\circledast} \bullet [\![M_1]\!]^{\circledast}$ and $K_2 \bullet K_1$ (because otherwise u could not be in both the compositions) but $v \cdot w \cdot m$ is not (or $u \cdot m$ would be in both compositions); but this again implies that w consists entirely of moves in θ' (if it contained a move in θ, then either the last move in u is in w not v, or $v \cdot w \cdot m \upharpoonright \theta \neq u \cdot m$), and so this would give a $v \cdot w \cdot m$ with the above properties, which is a contradiction. So in other words, $\mathrm{next}^{o}_{K}(u) = \mathrm{next}^{P}_{M}(u)$, and $\mathrm{next}^{i}_{K}(u) \supseteq \mathrm{next}^{O}_{M}(u)$, which is what we wanted to prove.

- *Contraction*: This can be seen as application of the two terms to contract to δ_θ, where θ is the type of the circuit; because both application and the implementation of contraction have been proved to follow the theorem in question already, the theorem holds in this case too, as it is just a special case of cases that have already been considered.
- *Function declaration*: Any function declaration can be seen as a currying relabeling of ports. This does not alter any of the traces of the circuit, or play of the SCI strategy, beyond relabeling them, and so if the theorem is true before the relabeling, it is still true afterwards.