Checking and Expanding Automaton Signatures in IOA

Stephen J. Garland, Nancy A. Lynch, Joshua A. Tauber

October 25, 2001

The IOA language provides notations for defining both primitive and composite I/O automata. This note formalizes the constraints on the signature of a primitive automaton, the composability requirements for the components of a composite automaton, and the definition of the signature of a composite automaton. It does so both in general terms and with reference to simple examples.

1 Notation

We use the notation $\overline{x}$ to denote a sequence $x_1, x_2, \ldots$ of (not necessarily distinct) elements, and we use the notation $\overline{i\colon S}$ to denote a sequence $i_1\colon S_1,\ i_2\colon S_2,\ \ldots$ of distinct variables (i.e., identifier-type pairs). Given a sequence $\overline{i\colon S}$ of variables, $\overline{\exists i\colon S}$ and $\overline{\forall i\colon S}$ are sequences of existential and universal quantifiers applied to those variables; by convention, $\overline{i}$ and $\overline{S}$ are the corresponding sequences of identifiers and types for those variables.

A *renaming* is a map from simple types and type constructors to types and type constructors such that the image of every simple type is a type and the image of every n-ary type constructor is an n-ary type constructor. Any renaming r extends naturally to a map $\hat{r}$ defined for all simple types and type constructors (by letting it be the identity on elements not in the domain of r) and then to a map r' from types to types, as in Figure 1. When r maps $\overline{T}$ to $\overline{E}$, we write $S[\overline{T}\ \textbf{gets}\ \overline{E}]$ for $r'(S)$.

$$r'(S) = \begin{cases} \hat{r}(S) & \text{if } S \text{ is a simple type,} \\ \hat{r}(S_0)[r'(S_1), \ldots, r'(S_n)] & \text{if } S \text{ is a compound type } S_0[S_1, \ldots, S_n] \end{cases}$$

Figure 1: Extension of a renaming r to a map r' from types to types

A *substitution* is a map from variables to terms such that the image of any variable has the same sort as that variable. As for renamings, any substitution σ extends naturally to a map $\hat{\sigma}$ defined for all variables (by letting it be the identity on variables not in the domain of σ) and then to a map σ' from terms to terms, as in Figure 2. In that figure, $\sigma_{v \leftarrow t}$ denotes a substitution that maps v to t and is otherwise the same as σ. When σ maps $\overline{i\colon S}$ to $\overline{t}$ and $P(\overline{i})$ is a term whose free variables are among those in $\overline{i}$, we write $P(\overline{t})$ for $\sigma'(P)$.

2 Parameterized automata

We separate the parameters in the definition of an automaton A into a sequence $\overline{T_A}$ of type identifiers and a sequence $\overline{i_A\colon S_A}$ of variables. Thus the definition of a parameterized automaton begins with

$$\sigma'(u) = \begin{cases} \hat{\sigma}(v) & \text{if } u \text{ is a variable } v, \\ f(\sigma'(u_1), \ldots, \sigma'(u_n)) & \text{if } u \text{ is a term } f(u_1, \ldots, u_n), \\ \exists v\colon S\, \sigma'_{v \leftarrow v}(u') & \text{if } u \text{ is a term } \exists v\colon S\,(u') \text{ and } v \text{ is not free in any term} \\ & \text{in the range of } \sigma, \\ \forall v\colon S\, \sigma'_{v \leftarrow v}(u') & \text{if } u \text{ is a term } \forall v\colon S\,(u') \text{ and } v \text{ is as above}, \\ \exists v'\colon S\, \sigma'_{v \leftarrow v'}(u') & \text{if } u \text{ is a term } \exists v\colon S\,(u'),\ v \text{ is free in some term in the} \\ & \text{range of } \sigma, \text{ and } v'\colon S \text{ is some ``fresh'' variable neither in } u' \\ & \text{nor free in any term in the range of } \sigma, \\ \forall v'\colon S\, \sigma'_{v \leftarrow v'}(u') & \text{if } u \text{ is a term } \forall v\colon S\,(u') \text{ and } v,\ v' \text{ are as above.} \end{cases}$$

Figure 2: Extension of a substitution σ to a map σ' from terms to terms

a line equivalent to[1] **automaton** $A(\overline{T_A}\colon \textbf{type}, \overline{i_A\colon S_A})$.

Example 1 Suppose the definition of A begins with **automaton** $A(M, \mathit{Index}\colon \textbf{type}, i, j\colon \mathit{Index})$. Then

- $\overline{T_A}$ is M, Index,
- $\overline{i_A\colon S_A}$ is $i\colon \mathit{Index}$, $j\colon \mathit{Index}$,
- $\overline{i_A}$ is i, j, and
- $\overline{S_A}$ is Index, Index.

Static checks.

- No variable appears more than once in $\overline{i_A\colon S_A}$.

3 Signatures of primitive automata

The declarations for parameterized sets of actions named by a single identifier π in the definition of a primitive automaton A occur in a context of the form shown in Figure 3. In this context, $\overline{i_{A,\pi,in}\colon S_{A,\pi,in}}$, etc., are sequences of parameters for π that may appear, along with the automaton parameters in $\overline{i_A\colon S_A}$, in the **where** clauses $P^{A,\pi}_{\mathrm{in}}$, etc.

The description of an automaton's signature in Figure 3 is sufficiently general, because we can desugar **const** parameters for an action using **where** clauses. For example, we can desugar

$$\textbf{input } \pi(\textbf{const } f(\overline{i_A}, \overline{k}), \overline{k\colon S}) \textbf{ where } Q(\overline{i_A}, \overline{k})$$

as

$$\textbf{input } \pi(j\colon S_0, \overline{k\colon S}) \textbf{ where } j = f(\overline{i_A}, \overline{k}) \wedge Q(\overline{i_A}, \overline{k})$$

[1] We say "equivalent to" because, for notational simplicity, we have written the type parameters before the variables even though IOA allows them to be intermixed in any order.

automaton $A(\overline{T_A\colon \mathbf{type}}, \overline{i_A\colon S_A})$
  signature
    input $\ldots, \pi(\overline{i_{A,\pi,in}\colon S_{A,\pi,in}}), \ldots$ **where** $P_{\text{in}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi,in}})$
    output $\ldots, \pi(\overline{i_{A,\pi,out}\colon S_{A,\pi,out}}), \ldots$ **where** $P_{\text{out}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi,out}})$
    internal $\ldots, \pi(\overline{i_{A,\pi,int}\colon S_{A,\pi,int}}), \ldots$ **where** $P_{\text{int}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi,out}})$

Figure 3: General form for an automaton's signature

Here, S_0 is the sort of the term $f(\overline{i_A}, \overline{k})$, and the variable $j\colon S_0$ is a fresh variable, that is, different from all variables in $\overline{i_A\colon S_A}$ and $\overline{k\colon S}$. (To avoid having to replace $j\colon S_0$ by yet another fresh variable in an instantiation of A that maps S_0 and the type of another variable with identifier j to a common type, it may help to pick j to be an identifier not in $\overline{i_A}$ or $\overline{k}$.)

Static checks.

- Each action name (e.g., π) occurs at most three times in the signature of an automaton: at most once in a list of input actions, at most once in a list of output actions, and at most once in a list of internal actions.
- Each occurrence of an action name in the signature of an automaton A must be followed by the same number and types of parameters. Hence $\overline{S_{A,\pi,in}} = \overline{S_{A,\pi,out}} = \overline{S_{A,\pi,int}}$, and we write $\overline{S_{A,\pi}}$ rather than $\overline{S_{A,\pi,in}}$, etc.
- Variables used as action parameters must be distinct from variables used as automaton parameters, that is, the variables in the sequences $\overline{i_{A,\pi,in}\colon S_{A,\pi}}$, $\overline{i_{A,\pi,out}\colon S_{A,\pi}}$, and $\overline{i_{A,\pi,int}\colon S_{A,\pi}}$ do not occur in $\overline{i_A\colon S_A}$.[2]

Although IOA does not require that the sequences $\overline{i_{A,\pi,in}}$, $\overline{i_{A,\pi,out}}$, and $\overline{i_{A,\pi,int}}$ of variables be the same, it is possible to change variables in the definition of the signature of A so that they are: let $\overline{i_{A,\pi}\colon S_{A,\pi}}$ be one of these sequences, replace the other two by it, and substitute (as defined in Figure 2) the variables in $\overline{i_{A,\pi}\colon S_{A,\pi}}$ for the corresponding variables in the other sequences in the **where** clauses associated with those sequences.

Semantic checks.

The sets of input, output, and internal actions in an automaton must be disjoint. In terms of first-order logic,

- $\overline{\forall i_A\colon S_A}\ \overline{\forall i_{A,\pi}\colon S_{A,\pi}}\ \neg(P_{\text{in}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi}}) \wedge P_{\text{out}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi}}))$
- $\overline{\forall i_A\colon S_A}\ \overline{\forall i_{A,\pi}\colon S_{A,\pi}}\ \neg(P_{\text{in}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi}}) \wedge P_{\text{int}}^{A,\pi}(\overline{i_A}, \overline{i_{A,\pi}}))$

[2] This restriction is designed to avoid the confusion that might result if, for example, the same variable occurs in both $\overline{i_A\colon S_A}$ and $\overline{i_{A,\pi,in}\colon S_{A,\pi}}$. A stronger restriction would prohibit an identifier from appearing in two different variables (of different types) in $\overline{i_A\colon S_A}$ and $\overline{i_{A,\pi,in}\colon S_{A,\pi}}$; this restriction would avoid the need to pick a fresh variable when an instantiation of A causes two variables with the same identifier to clash by mapping their types to a common type.

- $\overline{\forall i_A\colon S_A}\ \overline{\forall i_{A,\pi}\colon S_{A,\pi}}\ \neg(P^{A,\pi}_{\text{out}}(\overline{i_A},\overline{i_{A,\pi}}) \wedge P^{A,\pi}_{\text{int}}(\overline{i_A},\overline{i_{A,\pi}}))$

Except in special cases, these checks cannot be performed automatically. If two of the three **where** clauses for π are literally *false*, the checks automatically succeed. If two are literally *true*, they automatically fail. Otherwise some amount of (possibly interactive) theorem proving is needed to perform the checks.

4 Signatures of composite automata

In general, the definition of a composite automaton D without hidden actions has the form shown in Figure 4. Here parameterized components named $c_1, \ldots, c_n$ are based on instantiations of automata named $A_1, \ldots, A_n$. The formal parameters of component c_j are $\overline{i_{D.c_j}\colon S_{D.c_j}}$, and the actual parameters of automaton A_j consist of a sequence $\overline{E_{D.c_j}}$ of types and a sequence $\overline{e_{D.c_j}}$ of expressions involving the variables in $\overline{i_D}, \overline{i_{D.c_j}}$. (IOA permits the specification of c_j to be abbreviated by deleting the colon and the following expression when c_j and A_j are named by the same identifier, $\overline{E_{D.c_j}}$ is empty, and $\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}) = \overline{i_{D.c_j}}$.)

automaton $D(\overline{T_D}\colon$ **type**$, \overline{i_D\colon S_D})$
  components
    $c_1[\overline{i_{D.c_1}\colon S_{D.c_1}}] : A_1(\overline{E_{D.c_1}}, \overline{e_{D.c_1}}(\overline{i_D}, \overline{i_{D.c_1}}))$ **where** $P^{D.c_1}(\overline{i_D}, \overline{i_{D.c_1}})$;
    $\ldots$
    $c_n[\overline{i_{D.c_n}\colon S_{D.c_n}}] : A_n(\overline{E_{D.c_n}}, \overline{e_{D.c_n}}(\overline{i_D}, \overline{i_{D.c_n}}))$ **where** $P^{D.c_n}(\overline{i_D}, \overline{i_{D.c_n}})$

Figure 4: General form of a composite automaton without hidden actions

Example 2 Figure 5 shows the definition of a composite automaton D formed by composing instances of the automaton A defined in Figure 3 with instances of two other automata, B and C, also defined in Figure 5. In this example,

- C and D are unparameterized, and B has no type parameters, that is, $\overline{T_B}$, $\overline{T_C}$, $\overline{T_D}$, $\overline{i_C\colon S_C}$, and $\overline{i_D\colon S_D}$ are empty,
- B has a single non-type parameter, that is, $\overline{i_B\colon S_B}$ is $i\colon Int$,
- π has a single parameter k in the definitions of B and C, that is, $\overline{i_{B,\pi}\colon S_{B,\pi}}$ and $\overline{i_{C,\pi}\colon S_{C,\pi}}$ are $k\colon Int$,
- the first parameterized component of D (named a) is based on the automaton A,
- the second and third parameterized components of D (named B and C) are based, by the conventions for abbreviating component descriptions, on the automata B and C,
- the parameters $\overline{i_{D.a}\colon S_{D.a}}$ and $\overline{i_{D.B}\colon S_{D.B}}$ of the parameterized components a and B of D are both $i\colon Int$,
- component C of D has no parameters, that is, $\overline{i_{D.C}\colon S_{D.C}}$ is empty, and

automaton $B(i : Int)$
  signature
    input $\pi(k\colon Int)$ **where** $P^{B,\pi}_{\text{in}}(i,k)$
    output $\pi(k\colon Int)$ **where** $P^{B,\pi}_{\text{out}}(i,k)$
    internal $\pi(k\colon Int)$ **where** $P^{B,\pi}_{\text{int}}(i,k)$
  ...
automaton C
  signature
    input $\pi(k\colon Int)$ **where** $P^{C,\pi}_{\text{in}}(k)$
    output $\pi(k\colon Int)$ **where** $P^{C,\pi}_{\text{out}}(k)$
    internal $\pi(k\colon Int)$ **where** $P^{C,\pi}_{\text{int}}(k)$
  ...
automaton D
  components
    $a[i\colon Int] : A(Int, i, i+1)$ **where** $P^{D.a}(i)$;
    $B[i\colon Int]$ **where** $P^{D.B}(i)$;
    C **where** $P^{D.C}$

Figure 5: Sample composite automaton

- the actual parameters for the automaton A on which component a is based are $\overline{E_{D.a}} = Int$ and $\overline{e_{D.a}} = i, i+1$. Hence $\overline{T_A}$ must consist of a single type identifier T, and $\overline{S_A}$ must be T, T or Int, Int or T, Int or Int, T.

Static checks.

- Each component automaton is supplied with the appropriate number of actual types, that is, $\overline{E_{D.c_j}}$ has the same length as $\overline{T_{A_j}}$.
- Each component automaton is supplied with the appropriate number and types of its other actual parameters, that is, $\overline{e_{D.c_j}}$ has the same length as $\overline{i_{A_j} \colon S_{A_j}}$ and, for each k, the type of $e_{D.c_j,k}(\overline{i_D}, \overline{i_{D.c_j}})$ is $S_{A_j,k}[\overline{T_{A_j}}\ \textbf{gets}\ \overline{E_{D.c_j}}]$.
- All occurrences of an action name π in all component automata have the same number and types of parameters, that is, if π is an action name in A_j and A_k, then $\overline{S_{A_j,\pi}}[\overline{T_{A_j}}\ \textbf{gets}\ \overline{E_{D.c_j}}] = \overline{S_{A_k,\pi}}[\overline{T_{A_k}}\ \textbf{gets}\ \overline{E_{D.c_k}}]$.

Explicit description of the signature of a composite automaton

Before describing the semantic checks associated with the signature of D, we first introduce notation to provide an explicit description for that signature. For any action name π in some component of D, we let $\overline{S_{D,\pi}}$ be the unique sequence of types for the parameters of π in D, that is, $\overline{S_{D,\pi}}$ is $\overline{S_{A_j,\pi}}[\overline{T_{A_j}}\ \textbf{gets}\ \overline{E_{D.c_j}}]$ for some (equivalently, any) parameterized component c_j of D. We also pick a canonical sequence $\overline{i_{D,\pi}}$ of identifiers for the parameters of π in D; for example, we take $\overline{i_{D,\pi}}$ to be $\overline{i_{A_j,\pi}}$ for some component c_j of D.

In order to define the signature of the composite automaton D, we must construct the formulas $P_{\text{out}}^{D,\pi}$, $P_{\text{in}}^{D,\pi}$, and $P_{\text{int}}^{D,\pi}$ that appear in the **where** clauses in this signature. We define them in terms of other formulas, $P_{\text{out}}^{D.c_j,\pi}$, $P_{\text{in}}^{D.c_j,\pi}$, and $P_{\text{int}}^{D.c_j,\pi}$, that define the sets of output, input, and internal actions obtained from each component c_j.

Output actions An action is an output action of D if and only if it is an output action in some component c_j of D. Hence we define $P_{\text{out}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ to be

$$\bigvee_{1 \leq j \leq n} P_{\text{out}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$$

where $P_{\text{out}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$\overline{\exists i_{D.c_j} \colon S_{D.c_j}}\ (P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge P_{\text{out}}^{A_j,\pi}[\overline{T_{A_j}} \textbf{ gets } \overline{E_{D.c_j}}](\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}))$$

that is, $\pi(\overline{i_{D,\pi}})$ is an output action in a parameterized component c_j of D if there are values $\overline{i_{D.c_j}}$ for the parameters of c_j that, along with values $\overline{i_D}$ for the parameters of D, satisfy the **where** clause $P^{D.c_j}$ for component c_j and, furthermore, along with the values of the parameters $\overline{i_{D,\pi}}$ of π, satisfy the **where** clause $P_{\text{out}}^{A_j,\pi}$ for π as an output action in the instance of the automaton A_j upon which c_j is based (in this **where** clause, the actual type parameters $\overline{E_{D.c_j}}$ supplied by component c_j for automaton A_j replace the formal type parameters $\overline{T_{A_j}}$ of A_j, the canonical parameters $\overline{i_{D,\pi}}$ for π in D are substituted for the parameters $\overline{i_{A_j,\pi,out}}$ of the output actions named by π in A_j, and the actual non-type parameters $\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}})$ supplied by component c_j for A_j are substituted for the formal nontype parameters $\overline{i_{A_j}}$ of A_j).

In Example 2,

$$\begin{array}{rll}
P_{\text{out}}^{D,\pi}(k) & \text{is} & P_{\text{out}}^{D.a,\pi}(k) \vee P_{\text{out}}^{D.B,\pi}(k) \vee P_{\text{out}}^{D.C,\pi}(k) \\
P_{\text{out}}^{D.a,\pi}(k) & \text{is} & \exists i \colon Int\,(P^{D.a}(i) \ \wedge\ P_{\text{out}}^{A,\pi}[T \textbf{ gets } Int](i, i+1, k)) \\
P_{\text{out}}^{D.B,\pi}(k) & \text{is} & \exists i \colon Int\,(P^{D.B}(i) \ \wedge\ P_{\text{out}}^{B,\pi}(i, k)) \\
P_{\text{out}}^{D.C,\pi}(k) & \text{is} & P^{D.C} \ \wedge\ P_{\text{out}}^{C,\pi}(k)
\end{array}$$

Internal actions Likewise, an action is an internal action of D if and only if it is an internal action in some component c_j of D, and we define $P_{\text{int}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ to be

$$\bigvee_{1 \leq j \leq n} P_{\text{int}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$$

where $P_{\text{int}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$\overline{\exists i_{D.c_j} \colon S_{D.c_j}}\ (P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge P_{\text{int}}^{A_j,\pi}[\overline{T_{A_j}} \textbf{ gets } \overline{E_{D.c_j}}](\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}))$$

Input actioms Finally, an action is an input action of D if and only if it is an input action in some component c_j of D, but not an output action in any component of D. Hence we define $P_{\text{in}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ to be

$$\neg P_{\text{out}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \ \wedge \bigvee_{1 \leq j \leq n} P_{\text{in}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$$

where $P_{\text{in}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$\neg P_{\text{out}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \wedge \overline{\exists i_{D.c_j} \colon S_{D.c_j}}\ (P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge P_{\text{in}}^{A_j,\pi}[\overline{T_{A_j}}\ \textbf{gets}\ \overline{E_{D.c_j}}](\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}))$$

Semantic checks.

- The components of a composite automaton have disjoint sets of output actions.
- The set of internal actions for any component is disjoint from the set of all actions of every other component.

In order to express these requirements in first-order logic, we break each of them into two parts. For output actions, we first require that different parameterized components have disjoint sets of output actions, that is, for all distinct components c_j and c_k of D, for all values $\overline{i_D \colon S_D}$ of D's non-type parameters, and all values $\overline{i_{D,\pi} \colon S_{D,\pi}}$ of π's parameters,

$$\neg(P_{\text{out}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \wedge P_{\text{out}}^{D.c_k,\pi}(\overline{i_D}, \overline{i_{D,\pi}}))$$

Second, we require that different instances of the same parameterized component have disjoint sets of output actions, that is, for all components c_j of D, all values $\overline{i_D \colon S_D}$ of D's non-type parameters, all values $\overline{i_{D.c_j} \colon S_{D.c_j}}$ and $\overline{i'_{D.c_j} \colon S_{D.c_j}}$ for c_j's parameters, and all values $\overline{i_{D,\pi} \colon S_{D,\pi}}$ of π's parameters,

$$\begin{aligned}
&P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge P^{D.c_j}(\overline{i_D}, \overline{i'_{D.c_j}}) \wedge \\
&P_{\text{out}}^{D.c_j,\pi}(\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}) \wedge P_{\text{out}}^{D.c_j,\pi}(\overline{e_{D.c_j}}(\overline{i_D}, \overline{i'_{D.c_j}}), \overline{i_{D,\pi}}) \\
&\qquad \Rightarrow \overline{i_{D.c_j}} = \overline{i'_{D.c_j}}
\end{aligned}$$

In Example 2, these requirements become

$$\begin{aligned}
&\neg((P_{\text{out}}^{D.a,\pi}(k) \wedge P_{\text{out}}^{D.B,\pi}(k)) \\
&\neg((P_{\text{out}}^{D.a,\pi}(k) \wedge P_{\text{out}}^{D.C,\pi}(k)) \\
&\neg((P_{\text{out}}^{D.B,\pi}(k) \wedge P_{\text{out}}^{D.C,\pi}(k)) \\
&P^{D.a}(i) \wedge P^{D.a}(i') \wedge P_{\text{out}}^{D.A,\pi}(i, i+1, k) \wedge P_{\text{out}}^{D.A,\pi}(i', i'+1, k) \ \Rightarrow\ i = i' \\
&P^{D.B}(i) \wedge P^{D.B}(i') \wedge P_{\text{out}}^{D.B,\pi}(i, k) \wedge P_{\text{out}}^{D.B,\pi}(i', k) \ \Rightarrow\ i = i'
\end{aligned}$$

Similarly, for the internal actions of D, we require that

$$\neg(P_{\text{int}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \wedge P_{\text{all}}^{D.c_k,\pi}(\overline{i_D}, \overline{i_{D,\pi}}))$$

and

$$\begin{aligned}
&P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge P^{D.c_j}(\overline{i_D}, \overline{i'_{D.c_j}}) \wedge \\
&P_{\text{int}}^{D.c_j,\pi}(\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}) \wedge P_{\text{all}}^{D.c_j,\pi}(\overline{e_{D.c_j}}(\overline{i_D}, \overline{i'_{D.c_j}}), \overline{i_{D,\pi}}) \\
&\qquad \Rightarrow \overline{i_{D.c_j}} = \overline{i'_{D.c_j}}
\end{aligned}$$

where $P_{\text{all}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$\overline{\exists i_{D.c_j} \colon S_{D.c_j}}\ (P^{D.c_j}(\overline{i_D}, \overline{i_{D.c_j}}) \wedge (P_{\text{out}}^{A_j,\pi} \vee P_{\text{int}}^{A_j,\pi} \vee P_{\text{in}}^{A_j,\pi})[\overline{T_{A_j}}\ \textbf{gets}\ \overline{E_{D.c_j}}](\overline{e_{D.c_j}}(\overline{i_D}, \overline{i_{D.c_j}}), \overline{i_{D,\pi}}))$$

which is equivalent to

$$P_{\text{out}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \vee P_{\text{int}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \vee P_{\text{in}}^{D.c_j,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$$

In Example 2, these requirements become

$$\begin{array}{l}
\neg(P_{\text{int}}^{D.a,\pi}(k) \wedge P_{\text{all}}^{D.B,\pi}(k)) \\
\neg(P_{\text{int}}^{D.a,\pi}(k) \wedge P_{\text{all}}^{D.C,\pi}(k)) \\
\neg(P_{\text{int}}^{D.B,\pi}(k) \wedge P_{\text{all}}^{D.C,\pi}(k)) \\
P^{D.a}(i) \wedge P^{D.a}(i') \wedge P_{\text{int}}^{D.A,\pi}(i, i+1, k) \wedge P_{\text{all}}^{D.A,\pi}(i', i'+1, k) \Rightarrow i = i' \\
P^{D.B}(i) \wedge P^{D.B}(i') \wedge P_{\text{int}}^{D.B,\pi}(i, k) \wedge P_{\text{all}}^{D.B,\pi}(i', k) \Rightarrow i = i'
\end{array}$$

where, for example, $P_{\text{all}}^{D.B,\pi}(k)$ is

$$\exists i\colon Int\ (P^{D.B}(i) \wedge (P_{\text{out}}^{B,\pi}(i,k) \vee P_{\text{int}}^{B,\pi}(i,k) \vee P_{\text{in}}^{B,\pi}(i,k))$$

5 Signatures of composite automata with hidden actions

Specifications can hide output actions in a composite automaton D by adding statements of the form

$$\textbf{hidden}\ \pi(\overline{e}(\overline{i_D}, \overline{i}))\ \textbf{for}\ \overline{i\colon S}\ \textbf{where}\ H(\overline{i_D}, \overline{i})$$

to the definition of D shown in Figure 4. Here $\overline{e}$ is a sequence of expressions of types $\overline{S_{D,\pi}}$ involving the variables in $\overline{i\colon S}$ and the parameters of D, and H is a predicate whose free variables are among these variables. As for **const** parameters in an automaton's signature, we desugar the expressions in $\overline{e}$ by rewriting the **hidden** statement as

$$\textbf{hidden}\ \pi(\overline{i_{D,\pi}})\ \textbf{for}\ \overline{i_{D,\pi}\colon S_{D,\pi}}\ \textbf{where}\ \overline{\exists i\colon S}(\overline{i_{D,\pi}} = \overline{e}(\overline{i_D}, \overline{i}) \wedge H(\overline{i_D}, \overline{i}))$$

where $\overline{i_{D,\pi}\colon S_{D,\pi}}$ are the canonical parameters for π in D, upon which we impose the further restriction that they do not occur among the variables $\overline{i\colon S}$ in the **hidden** statement.

Have we spelled out the requirements for fresh variables completely enough? E.g., that these variables are chosen to be different from the automaton parameters?

Let D' be a composite automaton defined with such a desugared **hidden** statement, and let D be the composite automaton defined as for D', but without the **hidden** statement. Then $P_{\text{out}}^{D',\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$P_{\text{out}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \wedge \neg H(\overline{i_D}, \overline{i_{D,\pi}})$$

and $P_{\text{int}}^{D',\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is

$$P_{\text{int}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \vee (P_{\text{out}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \wedge H(\overline{i_D}, \overline{i_{D,\pi}}))$$

Semantic checks

- Only output actions may be hidden, that is, $H(\overline{i_D}, \overline{i_{D,\pi}})$ must imply $P_{\text{out}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}})$.

Hence $P_{\text{int}}^{D',\pi}(\overline{i_D}, \overline{i_{D,\pi}})$ is in fact equivalent to $P_{\text{int}}^{D,\pi}(\overline{i_D}, \overline{i_{D,\pi}}) \vee H(\overline{i_D}, \overline{i_{D,\pi}})$.