Color-blind Specifications for Transformations of Reactive Synchronous Programs

Kim G. Larsen[1], Ulrik Larsen[1], and Andrzej Wasowski[2]

[1] CISS, Aalborg University, Denmark, {kgl,ulrikl}@cs.aau.dk
[2] IT University of Copenhagen, Denmark, wasowski@itu.dk

Abstract. Execution environments are used as specifications for specialization of input-output programs in the derivation of product lines. These environments, formalized as color-blind I/O-alternating transition systems, are tolerant to mutations in a given program's outputs. Execution environments enable new compiler optimizations, vastly exceeding usual reductions. We propose a notion of context-dependent refinement for I/O-alternating transition systems, which supports composition and hierarchical reuse. The framework is demonstrated by discussing adaptations to realistic design languages and by presenting an example of a product line.

1 Introduction

Modern software becomes increasingly customizable. This especially affects embedded software, since embedded devices are typically produced in multiple variants. Our long-term goal is to provide a theoretical foundation, tools, and methodology for maintaining a family of software for reactive synchronous systems. In the present work we focus on the theoretical basis for specifying correctness of transformations used in automatic derivation of family members.

A single general model is used as a description of all available functionality. Hierarchically organized specifications of environments define the family members by restricting input and output abilities of the general model. I/O alternating transition systems are used to model the semantics of both environments and the general model. Our environments are novel in that they not only restrict possible input traces, but also exhibit inabilities in distinguishing output traces. Some outputs are indistinguishable for a given environment in the same way as a color-blind person cannot distinguish some colors. Color-blindness can be used to model surprisingly many aspects of realistic environments (for example causality between the firing and timing-out of a stop-watch, boolean memory flags, or the use of a single actuator in place of two). The general model can be transformed according to the behavior of a specific environment, and individually optimized for that particular environment and purpose.

Section 2 motivates our work using a popular reactive language. I/O alternating transition systems are introduced in section 3, color-blindness in section 4, and composition operators in section 5. Remaining sections focus on practical applications: adaptation to realistic design languages (section 6) and an example of a product line (section 7). Sections 8-9 refer the related work and conclude.

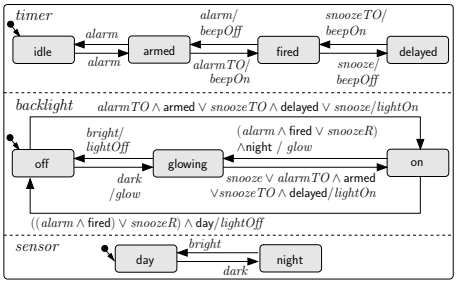

Fig. 1. Initial state/event model $\mathcal{C}_0$

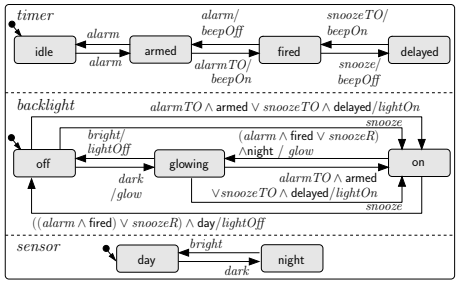

Fig. 2. The specialized model $\mathcal{C}_1$

2 State/Event Systems

Let *Event* and *Action* be finite sets of environment stimuli and system outputs respectively. A *state/event machine* $M_i = (S_i, s_i^0, T_i)$ is a triple comprising a set of local states S_i, the initial state $s_i^0 \in S_i$ and a set of syntactic transitions T_i. A *state/event system* consists of n machines $\mathcal{M} = \{M_1, \ldots, M_n\}$ with mutually disjoint sets of states. A global state of the system is a tuple of local states: $State = S_1 \times S_2 \times \cdots \times S_n$. Transitions in $T_i \subseteq S_i \times Event \times Guard \times Action \times S_i$ describe reactions undertaken by M_i in reply to a given event, in a given local and global state. Global states are described by transition guards: simple Boolean expressions over activity of states, which can be evaluated in any given global state, giving rise to a natural satisfaction relation $\vDash \subseteq State \times Guard$.

State/event systems are *input-enabled*: the local transition relation includes not only the syntactical transitions but also self loops for all configurations for which reactions are not specified. We write $s \xrightarrow{e \quad o}_i s_i'$, meaning that the reaction of machine M_i to arrival of event e in global state s is, to change the local state to s_i' and generate the set of actions o:

$$s \xrightarrow{e \quad \{a\}}_i s_i' \qquad \text{iff } \exists g.\ (\pi_i(s), e, g, a, s_i') \in T_i \wedge s \vDash g$$
$$s \xrightarrow{e \quad \emptyset}_i \pi_i(s) \qquad \text{otherwise (where } \pi_i(s) \text{ denotes the } i\text{'th projection of } s)$$

The global transition relation $T \subseteq State \times Event \times \mathcal{P}(Action) \times State$ subsumes all local reactions: $s \xrightarrow{e \quad o} s' \Leftrightarrow_{def} \forall i. s \xrightarrow{e \quad o_i}_i \pi_i(s')$ where $o = o_1 \cup \ldots \cup o_n$.

Fig. 1 depicts a state/event model $\mathcal{C}_0$ of an alarm clock. The essentials of the alarm clock are handled by the *timer* machine. If the *timer* is in the armed state and the hardware sends an alarm time-out event (*alarmTO*) then the beeper is turned on. The user can postpone the alarm by pressing the snooze button (event *snooze*), which allows him to continue sleeping until the snooze timer times out (*snoozeTO*). Releasing the button sends a *snoozeR* event to the model. The *backlight* machine controls the built-in lamps. Only a faint light is displayed in the glowing state, such that the display can be read in the dark. The full light is on while the alarm is beeping or the snooze button is being pressed. The *sensor* machine models the current external light level. Proper events (*dark*, *bright*) are generated by the sensor driver whenever the ambient light passes some threshold.

We would like to support automatic derivation of variants for discrete control systems like the alarm clock. One such variant $\mathcal{C}_1$, which does not activate the backlight in reaction to the *snooze* button, is depicted on Fig. 2. Note the simplification of guards and the two new transitions in the *backlight* state machine. What is the relation between the two models? Both models are indistinguishable for some execution environment, namely the one, which becomes blind for the *lightOn* action immediately after producing the *snooze* event.

3 I/O Alternating Transition Systems

The reactive synchronous paradigm seems to be predominant in development of embedded software. The state/event systems of the previous section [17, 11] are just an example chosen from a multitude of available formalisms, like Esterel [2], statecharts [7], or Java Card [24]. A common assumption about these systems is that they react to any input event at any time. Each reaction occurs infinitely fast, so that the system is always able to observe the arrival of the next event. Such semantics is conveniently captured by *I/O-alternating transition systems*:

Definition 1. *An I/O-alternating transition system, or IOATS, is a tuple* $(In, Out, Gen, Obs, \xrightarrow{!}, \xrightarrow{?}, s^0)$*, where In and Out are sets of inputs and outputs, Gen and Obs are finite sets of generators and observers,* $\xrightarrow{!} \subseteq Gen \times Out \times Obs$ *is a generation relation,* $\xrightarrow{?} \subseteq Obs \times In \times Gen$ *is an observation relation, and* $s^0 \in Gen \cup Obs$ *is the initial state.*

We have distinguished two transition relations: $\xrightarrow{!}$ is a generation relation advancing from a generator to an observer, while $\xrightarrow{?}$ is an observation relation advancing from an observer to a generator. This alternation is inherent to the way synchronous systems operate. We write $S \xrightarrow{o!} s$, instead of $(S, o, s) \in \xrightarrow{!}$ and $s \xrightarrow{i?} S$ instead of $(s, i, S) \in \xrightarrow{?}$. Small letters are used for observers and capital letters for generators. In addition observers are required to be input-enabled:

$$\forall s \in Obs.\ \forall i \in In.\ \exists S, o, s'.\ s \xrightarrow{i?} S \wedge S \xrightarrow{o!} s' \tag{1}$$

With these assumptions we can propose a simulation based refinement relation:

Definition 2. *Let* $\mathcal{S}_1 = (In, Out, Gen_1, Obs_1, \xrightarrow{!}_1, \xrightarrow{?}_1, s_1^0)$ *and* $\mathcal{S}_2 = (In, Out, Gen_2, Obs_2, \xrightarrow{!}_2, \xrightarrow{?}_2, s_2^0)$ *be IOATSs. A binary relation* $R \in Obs_1 \times Obs_2$ *constitutes a simulation on observers of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *iff* $(s_1, s_2) \in R$ *implies that:*

$$\textit{whenever } s_1 \xrightarrow{i?} S_1 \wedge S_1 \xrightarrow{o!} s_1' \textit{ then also } s_2 \xrightarrow{i?} S_2 \wedge S_2 \xrightarrow{o!} s_2' \textit{ and } (s_1', s_2') \in R\ .$$

Let R be the largest of such relations ordered by inclusion. An observer s_2 *simulates an observer* s_1*, written* $s_1 \leqslant s_2$*, iff* $(s_1, s_2) \in R$*. Finally* $\mathcal{S}_2$ *simulates* $\mathcal{S}_1$*, written* $\mathcal{S}_1 \leqslant \mathcal{S}_2$*, iff* $s_1^0 \leqslant s_2^0$*.*

We distinguish the actual systems from the environments, in which they operate. Environments are free in choice of inputs, while systems independently determine the outputs. A system $\mathcal{S} = (In_{\mathcal{S}}, Out_{\mathcal{S}}, Gen_{\mathcal{S}}, Obs_{\mathcal{S}}, \xrightarrow{!}_{\mathcal{S}}, \xrightarrow{?}_{\mathcal{S}}, s_{\mathcal{S}})$ operates embedded in some environment $\mathcal{E} = (In_{\mathcal{E}}, Out_{\mathcal{E}}, Gen_{\mathcal{E}}, Obs_{\mathcal{E}}, \xrightarrow{!}_{\mathcal{E}}, \xrightarrow{?}_{\mathcal{E}}, s_{\mathcal{E}})$.

Systems always begin execution in an observer state, so $s_{\mathcal{S}} \in Obs_{\mathcal{S}}$. Environments always begin execution in a generator state, so $s_{\mathcal{E}} \in Gen_{\mathcal{E}}$. System $\mathcal{S}$ is *compatible* with the environment $\mathcal{E}$ if $In_{\mathcal{S}} = Out_{\mathcal{E}}$ and $Out_{\mathcal{S}} = In_{\mathcal{E}}$. Composition of a system $\mathcal{S}$ with a compatible environment $\mathcal{E}$ is defined in the usual way, by synchronization on identical labels (and complimentary transition types). The initial observer of the system is composed with the initial generator of the environment. Due to the compatibility requirement and input-enabledness of observers, the closed system is able to advance for any input that can be generated by the environment. For a closed system it is known, which of its states cannot be exercised by the environment. A given environment may not be able to distinguish two systems from each other, even though they are not identical. We capture this with a notion of *relativized simulation*:

Definition 3. *Consider three IOATSs: an environment* $\mathcal{E} = (Out, In, Gen, Obs, \xrightarrow{!}, \xrightarrow{?}, E^0)$ *and two systems:* $\mathcal{S}_1 = (In, Out, Gen_1, Obs_1, \xrightarrow{!}_1, \xrightarrow{?}_1, s_1^0)$ *and* $\mathcal{S}_2 = (In, Out, Gen_2, Obs_2, \xrightarrow{!}_2, \xrightarrow{?}_2, s_2^0)$*. A Gen-indexed family of binary relations* $R{:}\,Gen \to \mathcal{P}(Obs_1 \times Obs_2)$ *is a relativized simulation iff* $(s_1, s_2) \in R_E$ *implies that:*

$$\begin{array}{l} \textit{whenever } E \xrightarrow{i!} e \wedge e \xrightarrow{o?} E' \\ \quad \textit{then whenever } s_1 \xrightarrow{i?} S_1 \wedge S_1 \xrightarrow{o!} s_1' \\ \quad\quad \textit{then also } s_2 \xrightarrow{i?} S_2 \wedge S_2 \xrightarrow{o!} s_2' \textit{ and } (s_1', s_2') \in R_{E'}. \end{array}$$

Let R be the largest of such families ordered by component-wise inclusion. We say that an observer s_2 simulates an observer s_1 in the generator E, written $s_1 \leqslant_E s_2$, iff $(s_1, s_2) \in R_E$. The system $\mathcal{S}_2$ simulates $\mathcal{S}_1$ in the context of $\mathcal{E}$, written $\mathcal{S}_1 \leqslant_{\mathcal{E}} \mathcal{S}_2$, iff $s_1^0 \leqslant_{E^0} s_2^0$.

The choice of simulation as the preorder underlying our methodology is somewhat arbitrary. Most other behavioral preorders of the linear-time/branching-time hierarchy of van Glabbeek [6] would be adequate, such as testing preorder, $^2/_3$ bisimulation (ready simulation) and bisimulation. What is important is that the particular preorder preserves properties of interest and that the preorder may be relativized with respect to environmental restrictions.

4 Color-blind I/O-alternating Transition Systems

In the previous section we were able to state that two systems are in a refinement relation with respect to a certain context if this context cannot activate their incompatible parts. However, in industrial development, it often happens that the environment cannot distinguish two systems, not because it makes incompatible parts unreachable, but because its ability to distinguish the *different outputs* it observes might be limited depending on its actual state. A variant of the alarm clock may have only one physical lamp installed, which should be lit whenever the backlight is on or glowing. The environment, being a model of the hardware in this case, will treat the two outputs *glow* and *lightOn* as being identical, allowing for powerful optimizations when generating code for this specific type

of hardware. For this particular example, the distinguishing capability of the environment is clearly static and hence the specification of code optimization is realizable using simple process algebraic operations such as relabelling and hiding. However, environmental restrictions can be dynamically changing as was the case for the environment leading to the specialized model $\mathcal{C}_1$ (Fig. 2). Here the environment only becomes blind for the *lightOn* action after the production of the *snooze* event. To give a proper treatment of such situations we relax the equivalence of labels in relativized simulation and label observation transitions of environments with sets of inputs called *observation classes*. Such transitions can be taken in the presence of any of the inputs in their observation class.

Definition 4. *A* color-blind *IOATS is a tuple* $\mathcal{E} = (In, Out, Gen, Obs, \xrightarrow{!}, \xrightarrow{?}, E^0)$*, where In and Out are sets of inputs and outputs, Gen and Obs are finite sets of generators and color-blind observers,* $\xrightarrow{!} \subseteq Gen \times Out \times Obs$ *is a generation relation,* $\xrightarrow{?} \subseteq Obs \times \mathcal{P}(In) \times Gen$ *is a color-blind observation relation, and* $E^0 \in Gen$ *is the initial state.*

A color-blind environment $\mathcal{E} = (In_\mathcal{E}, Out_\mathcal{E}, Gen_\mathcal{E}, Obs_\mathcal{E}, \xrightarrow{!}_\mathcal{E}, \xrightarrow{?}_\mathcal{E}, E)$ and a usual IOATS $\mathcal{S} = (In_\mathcal{S}, Out_\mathcal{S}, Gen_\mathcal{S}, Obs_\mathcal{S}, \xrightarrow{!}_\mathcal{S}, \xrightarrow{?}_\mathcal{S}, s)$ are compatible if their signatures match: $In_\mathcal{E} = Out_\mathcal{S} \wedge Out_\mathcal{E} = In_\mathcal{S}$. Since we only consider compatible systems and environments, we fix the meaning of the input and output, choosing the system's perspective. We denote the set of inputs of the system by *In* (which is also the set of outputs of the environment). Similarly *Out* is the set of outputs of the system (but the set of inputs for the environment). A single input will be denoted by i, single output by o, and classes of outputs by capital O. We still write $E \xrightarrow{i!} e$ instead of $(E, i, e) \in \xrightarrow{!}$ and $e \xrightarrow{O?} E$ instead of $(e, O, E) \in \xrightarrow{?}$.

We require that the observers in color-blind IOATS are deterministic and input enabled, so that the observation classes on the transitions outgoing from a single state form a partitioning of the inputs into equivalence classes. Formally:

$$\begin{aligned}
&\forall e \in Obs_\mathcal{E}. \forall O_1, O_2 \subseteq Out. \forall E_1, E_2 \in Gen_\mathcal{E}.\ e \xrightarrow{O_1?} E_1 \wedge e \xrightarrow{O_2?} E_2 \\
&\qquad \Rightarrow O_1 \cap O_2 = \emptyset \vee (O_1 = O_2 \wedge E_1 = E_2) \\
&\forall e \in Obs_\mathcal{E}. \forall o \in Out. \exists O \subseteq Out. \exists E \in Gen_\mathcal{E}. e \xrightarrow{O?} E \wedge o \in O. \qquad (2)
\end{aligned}$$

The generation relation should also be deterministic: $\forall E \in Gen_\mathcal{E}.\ \forall i \in In.\ \forall e_1, e_2 \in Obs_\mathcal{E}. E \xrightarrow{i!} e_1 \wedge E \xrightarrow{i!} e_2 \Rightarrow e_1 = e_2$. Note that determinism in this sense does not limit the freedom of the environment in choosing inputs, but means that each input choice uniquely determines the target state.

Consider a blind environment $\mathcal{B}$ with two states, a generator $\mathbf{B}$ and an observer $\mathbf{b}$. Intuitively $\mathcal{B}$ can execute all parts of the system, but does not care about the responses it gets: $\forall i \in In.\ \mathbf{B} \xrightarrow{i!} \mathbf{b}$ and $\mathbf{b} \xrightarrow{Out?} \mathbf{B}$. Dually, a perfect vision environment $\mathcal{V}$ observes all the outputs: $\forall i \in In. \mathbf{V} \xrightarrow{i!} \mathbf{v}$ and $\forall o \in Out. \mathbf{v} \xrightarrow{\{o\}?} \mathbf{V}$.

A compatible environment–system pair forms a closed system, advancing in lock-steps. However, now the generation transition of the system, synchronizes with the observation transition of the environment, whenever the output produced falls into the right observation class. We enrich our previous definition of relativized simulation to accommodate this new synchronization principle:

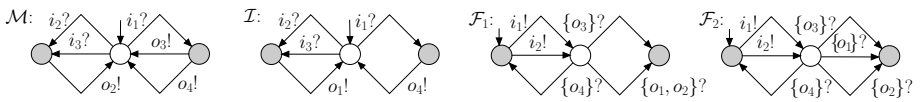

Fig. 3. Systems $\mathcal{M}$ and $\mathcal{I}$ and compatible environments $\mathcal{F}_1$, $\mathcal{F}_2$

Definition 5. *Let* $\mathcal{E}=(Out, In, Gen, Obs, \xrightarrow{!}, \xrightarrow{?}, E^0)$ *be a color-blind environment IOATS and* $\mathcal{S}_1=(In, Out, Gen_1, Obs_1, \xrightarrow{!}_1, \xrightarrow{?}_1, s_1^0)$, $\mathcal{S}_2=(In, Out, Gen_2, Obs_2, \xrightarrow{!}_2, \xrightarrow{?}_2, s_2^0)$ *be two system IOATSs. A Gen-indexed family of relations* $R: Gen \to \mathcal{P}(Obs_1 \times Obs_2)$ *is a relativized simulation iff* $(s_1, s_2) \in R_E$ *implies that:*

$$\begin{array}{l} \textit{whenever } E \xrightarrow{i!} e \wedge e \xrightarrow{O?} E' \\ \quad \textit{then whenever } s_1 \xrightarrow{i?} S_1 \wedge S_1 \xrightarrow{o_1!} s_1' \wedge o_1 \in O \\ \quad\quad \textit{then also } s_2 \xrightarrow{i?} S_2 \wedge S_2 \xrightarrow{o_2!} s_2' \wedge o_2 \in O \textit{ and } (s_1', s_2') \in R_{E'}. \end{array}$$

Let R *be the largest of such families ordered by component-wise inclusion. An observer* s_2 *simulates an observer* s_1 *in the context of generator* E*, written* $s_1 \leqslant_E s_2$*, iff* $(s_1, s_2) \in R_E$*. An IOATS* $\mathcal{S}_2$ *simulates another IOATS* $\mathcal{S}_1$ *in the context of a compatible color-blind IOATS* $\mathcal{E}$*, written* $\mathcal{S}_1 \leqslant_\mathcal{E} \mathcal{S}_2$*, iff* $s_1^0 \leqslant_{E^0} s_2^0$*. Finally* $\mathcal{S}_1$ *is equivalent to* $\mathcal{S}_2$ *in the context of* $\mathcal{E}$*, written* $\mathcal{S}_1 \lesseqgtr_\mathcal{E} \mathcal{S}_2$*, iff* $\mathcal{S}_1 \leqslant_\mathcal{E} \mathcal{S}_2$ *and* $\mathcal{S}_2 \leqslant_\mathcal{E} \mathcal{S}_1$.

Even though we have initially postulated that most of the execution contexts do not exercise all possible traces of the system, we shall now require that environments can always produce any of the inputs in In. This requirement surprisingly does not defeat our initial goal. We can direct all transitions producing impossible inputs to the observer **b** and embed the blind environment $\mathcal{B}$ in every environment. Instead of specifying that the environment cannot produce i, we state that i can be produced, but the subsequent system behavior is irrelevant. Proposition 1 states this formally:

Proposition 1. *For any two observers* S_1, S_2 *of the same IOATS* $S_1 \leqslant_\mathbf{B} S_2$.

Fig. 3 presents two systems and two compatible color-blind environments. Environment transitions from generators to the blind observer **b** have been omitted. There is one such transition for each input–generator pair, for which the transition is not drawn. Observe that the system $\mathcal{M}$ simulates $\mathcal{I}$ in the environment $\mathcal{F}_1$ ($\mathcal{I} \leqslant_{\mathcal{F}_1} \mathcal{M}$) not due to the fact that $\mathcal{F}_1$ is not able to exercise the differing parts of the two systems, but because $\mathcal{F}_1$ cannot distinguish between the outputs (o_1, o_2) produced by $\mathcal{I}$ and $\mathcal{M}$. The $\mathcal{F}_2$ environment distinguishes $\mathcal{I}$ and $\mathcal{M}$, by observing the outputs o_1 and o_2 with two separate transitions.

Relativized simulation is a weaker notion than usual simulation and the perfect vision environment $\mathcal{V}$ is the most discriminating environment:

Proposition 2. *For any two systems* $\mathcal{S}_1$, $\mathcal{S}_2$ *and for any compatible color-blind environment* $\mathcal{E}$ *it holds that* $\mathcal{S}_1 \leqslant \mathcal{S}_2 \Rightarrow \mathcal{S}_1 \leqslant_\mathcal{E} \mathcal{S}_2$ *and* $\mathcal{S}_1 \leqslant \mathcal{S}_2 \iff \mathcal{S}_1 \leqslant_\mathcal{V} \mathcal{S}_2$.

With the above propositions we have hinted at the notion of *discrimination*—the ability of environment to distinguish systems from each other:

Definition 6. *A color-blind IOATS $\mathcal{F}$ is more discriminating than $\mathcal{E}$, written $\mathcal{E} \sqsubseteq \mathcal{F}$, iff $\mathcal{F}$ distinguishes more processes: $\mathcal{E} \sqsubseteq \mathcal{F}$ iff $\forall \mathcal{S}_1, \mathcal{S}_2 . \mathcal{S}_1 \leqslant_{\mathcal{F}} \mathcal{S}_2 \Rightarrow \mathcal{S}_1 \leqslant_{\mathcal{E}} \mathcal{S}_2$.*

The blind environment $\mathcal{B}$ is the least discriminating—it cannot distinguish any two systems from each other (proposition 1). By proposition 2 the perfect vision environment $\mathcal{V}$ is the most discriminating one.

The notion of discrimination will soon prove fundamental for our developments. We shall use it to design composition operators for behavioral properties, facilitating hierarchical modeling of product lines. Unfortunately the definition of the discrimination is rather abstract. The quantification over all systems, makes it infeasible for mechanical treatment. To remedy this obstacle we introduce a new preorder on environments: a simulation for color-blind IOATSs.

Definition 7. *Let $\mathcal{E} = (Out, In, Gen_{\mathcal{E}}, Obs_{\mathcal{E}}, \xrightarrow{!}_{\mathcal{E}}, \xrightarrow{?}_{\mathcal{E}}, E^0)$ and $\mathcal{F} = (Out, In, Gen_{\mathcal{F}}, Obs_{\mathcal{F}}, \xrightarrow{!}_{\mathcal{F}}, \xrightarrow{?}_{\mathcal{F}}, F^0)$ be color-blind environments. A pair of binary relations, $R_1 \subseteq Gen_{\mathcal{E}} \times Gen_{\mathcal{F}}$ and $R_2 \subseteq Obs_{\mathcal{F}} \times Obs_{\mathcal{E}}$, constitutes a simulation between states of color-blind IOATSs iff $(E, F) \in R_1$ implies that*

$$\text{whenever } E \xrightarrow{i!} e \text{ then also } F \xrightarrow{i!} f \text{ and } (f, e) \in R_2 \ ,$$

and $(f, e) \in R_2$ implies that whenever $f \xrightarrow{O_f ?} F$

$$\text{then also } e \xrightarrow{O_e ?} E \text{ and } O_f \subseteq O_e \text{ and } (E, F) \in R_1 \ .$$

Let (R_1, R_2) be the largest such pair of relations (ordered by point-wise inclusion). A generator F simulates a generator E, written $E \leqslant F$, iff $(E, F) \in R_1$. An observer e simulates an observer f, written $f \leqslant e$, iff $(f, e) \in R_2$. An environment $\mathcal{F}$ simulates $\mathcal{E}$, written $\mathcal{E} \leqslant \mathcal{F}$, iff $E^0 \leqslant F^0$.

The simulation preorder can be established mechanically for finite state systems using state exploration techniques [3]. Thanks to the following central result, these techniques can also be used to verify discrimination properties:

Theorem 1. *For any two color-blind environments $\mathcal{E}$ and $\mathcal{F}$: $\mathcal{E} \sqsubseteq \mathcal{F}$ iff $\mathcal{E} \leqslant \mathcal{F}$.*

5 Composition of Behavioral Properties

Typical code generators do not use any context information, assuming that the model is combined with the perfect vision environment $\mathcal{V}$. Another extreme would be a program synthesis tool requiring a precise environment model, imposing a significant burden on engineers. We propose light-weight, composable, partial specifications of environments in the form of behavioral properties like: that certain events always come interleaved (e.g. on/off switch), or that there is causality between an input and an output (e.g. a timer only timeouts after it has been started). Each property can be expressed as a simple color-blind IOATS. In this section we consider ways of composing such properties.

As said before, every observer e of a color-blind IOATS induces a partitioning of Out into observation classes. Let us denote this partitioning by P_e. The set of

all equivalence relations (and hence the set of all partitionings) over Out, ordered by inclusion, forms a complete lattice. Consequently for any set of partitionings $\{P_k\}_{k\in L}$ there exist the greatest lower bound $\bigsqcap_{k\in L} P_k$, which is the coarsest partitioning finer than any of P_k and the least upper bound $\bigsqcup_{k\in L} P_k$, which is the finest partitioning coarser than all P_k.

The composition is defined for environments with the same I/O signatures. We consider two kinds of composition: a sum and a product. Sums intuitively correspond to disjunction of properties (or sums in CCS [19]). Products correspond to conjunctions (or synchronous composition in CSP [9]).

$$\frac{E_1 \xrightarrow{i!} e_1 \ \dots \ E_n \xrightarrow{i!} e_n}{\sum_{k=1}^{n} E_k \xrightarrow{i!} \prod_{k=1}^{n} e_k} \text{(SG)} \qquad \frac{O \in \bigsqcup_{k=1}^{n} P_{e_k} \quad \mathbb{E} = \{E | \exists 1 \leq k \leq n. \exists O' \subseteq O. e_k \xrightarrow{O'?} E\}}{\sum_{k=1}^{n} e_k \xrightarrow{O?} \prod \mathbb{E}} \text{(SO)}$$

$$\frac{E_1 \xrightarrow{i!} e_1 \ \dots \ E_n \xrightarrow{i!} e_n}{\prod_{k=1}^{n} E_k \xrightarrow{i!} \sum_{k=1}^{n} e_k} \text{(PG)} \qquad \frac{O \in \bigsqcap_{k=1}^{n} P_{e_k} \quad \mathbb{E} = \{E | \exists 1 \leq k \leq n. \exists O' \subseteq Out. e_k \xrightarrow{O'?} E \wedge O \subseteq O'\}}{\prod_{k=1}^{n} e_k \xrightarrow{O?} \sum \mathbb{E}} \text{(PO)}$$

The result of a composition is a well-formed color-blind IOATS. The rules for the sum of generators (SG) and for the product of generators (PG) are very simple, due to the determinism and input-enabledness of our generators. The composition is synchronous: all composed generators take identical steps simultaneously. From the system's perspective a single input is generated. The observer rules are more complex, due to the embedding of determinisation. Consider the product of observers (PO) first. The observation classes O of the composed environment will be finer than observation classes of any of the composed processes. Whenever any o is observed by the result of the composition we advance to the state $\mathbb{E}$ composed of states reachable by o from all e_k's. Since O is finer than some class in any of these observers there is always exactly n such reachable generators. Dually in the sum (SO) observational classes are coarser than classes of any of the composed observers. The transition relation follows to those generators that can be reached by any output belonging to such an extended class. The size of $\mathbb{E}$ can exceed the number of original observers n.

Our compositions enjoy the following essential property:

Theorem 2. *$\sum_{k=1}^{n}\{\mathcal{E}_k\}$ is the least environment, which simulates all summands, while $\prod_{k=1}^{n}\{\mathcal{E}_k\}$ is the greatest environment, which is simulated by all the factors.*

Since discrimination and simulation coincide (Thm. 1) $\sqsubseteq$ can replace $\leqslant$ in the above theorem: *The sum of environments is the least discriminating environment, more discriminating than each of the summands. The product is the most discriminating environment, less discriminating than each of the factors.* These in turn are standard expectations about conjunction and disjunction. A conjunction (product) of two properties expressing inability to observe two behaviors, will result in a property expressing inability to observe either. Disjunction (sum) of two properties expressing ability to observe something, results in a property expressing the ability to observe both. See example on Fig. 4.

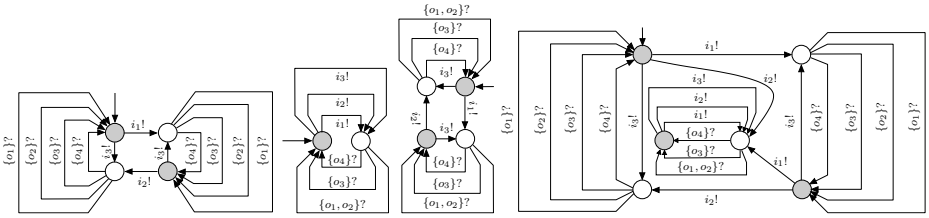

Fig. 4. Environments *Interleave* $i_1\ i_2$ (a) and *Equiv* $o_1\ o_2$ (b), their product (c) and sum (d) ($In = \{i_1, i_2, i_3\}$, $Out = \{o_1, \ldots, o_4\}$). Transitions to the **b** observer are suppressed. The product only generates what both of the factors could generate. It can distinguish only what both of them could. The sum can generate what any of the summands could observe, and it observers what any of them could. In particular o_1 and o_2 are distinguished in the traces for which the *Interleave* property is preserved and not otherwise.

6 Toward Realistic Design Languages

Until now we have assumed that outputs of systems are atomic. This assumption however often does not hold for realistic languages, which typically support structured output: sets, multisets, sequences or even sequences of sets of atomic actions produced in a single step. We have successfully applied our framework to the semantics of languages producing sets (state/event systems of section 2, Harel's statecharts [7], synchronous languages [2]) and sequences (Java Card [24], UML state diagrams [21]). We describe some intricacies of the latter here, while simpler set-based environments are demonstrated by example in section 7.

Let *Event* and *Action* be finite sets of atomic events and actions respectively. Each observation transition of the system awaits a single input from *Event*, while each generation transition produces an output which is a finite sequence of actions from *Action*: $In = Event$ and $Out = Action^*$. The first step in adapting the theory is linking the concrete states of models (for example state configurations in statecharts, or variable store in Java Card) to abstract states of the IOATS. This can normally be done in a direct way (at least for finite state models). Subsequently the observation and generation relations must be extracted from the semantics of the language in question. Observation classes on the environment side (color-blind) become sets of sequences of actions. Partitioning of $Action^*$ into classes that are regular languages can be described by a finite automaton.

Definition 8. *A classifier DFA over alphabet A is a quadruple $c = (S, A, s, \rightarrow)$, where S is a finite set of states, A is a finite set of symbols, $s \in S$ is an initial state and $\rightarrow \in S \rightarrow A \rightarrow S$ is an input-enabled transition function, meaning that for every $s \in S$ function $\rightarrow(s)$ is defined for each element of its domain A. We usually write $s \xrightarrow{a} s'$ instead of $\rightarrow(s)(a) = s'$.*

A classifier DFA consecutively applies $\rightarrow$ to a state and the head of the input sequence obtaining a new state and input sequence. An execution over a list of symbols $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$ is abbreviated with $s \xrightarrow{a_1 \ldots a_n}{}^* s_n$.

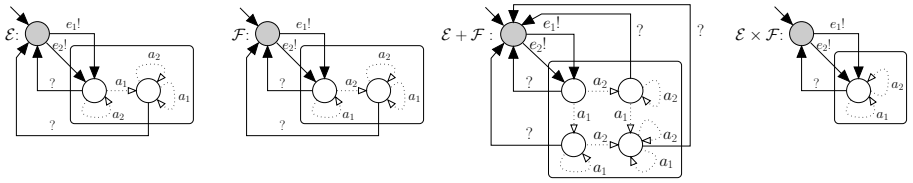

Fig. 5. Environments $\mathcal{E}$ and $\mathcal{F}$ observing sequences, their sum and product.

Definition 9. *Let $c=(S,A,s,\rightarrow)$ be a classifier. Sequences $\sigma_1,\sigma_2\in A^*$ are equivalent with respect to c if both advance c to the same state: $\exists s'.s\xrightarrow{\sigma_1}{}^*s'\wedge s\xrightarrow{\sigma_2}{}^*s'$.*

The equivalence with respect to a classifier is an equivalence relation and partitions A^* into a finite set of classes, isomorphic with the reachable states.

For a classifier $e=(S_e,Action,s_e,\rightarrow_e)$ consider a mapping of its states to generators $\gamma_e:S_e\rightarrow Gen$. Each observer of the environment comprises a classifier and a generator mapping. Environments advance from an observer (e,γ_e) to a generator $\gamma_e(s)$ if it observes a sequence σ advancing the classifier to a state s:

$$(e,\gamma_e)\xrightarrow{\{\sigma\mid s_e\xrightarrow{\sigma}{}^*s\}\ ?}\gamma_e(s)\ .$$

Fig. 5 shows two color-blind IOATSs $\mathcal{E}$ and $\mathcal{F}$ of signature: $Event=\{e_1,e_2\}$ and $Action=\{a_1,a_2\}$. $\mathcal{E}$ distinguishes reactions containing at least one occurrence of a_1 from those not containing a_1 at all. Similarly $\mathcal{F}$ distinguishes between sequences containing at least one a_2 from those not containing a_2 at all. Observers are drawn as boxes containing classifier DFAs. Classifier transitions are represented as dotted arrows to distinguish them from IOATS transitions.

The product of classifiers is a central construction in computing products of observers, supporting composition of environments:

Definition 10. *Let $e=(S_e,A,s_e,\rightarrow_e)$ and $f=(S_f,A,s_f,\rightarrow_f)$ be classifiers. A product of e and f is a classifier $e\otimes f=(S_e\times S_f,A,(s_e,s_f),\rightarrow)$, where $(s_e,s_f)\xrightarrow{a}(s'_e,s'_f)$ if $s_e\xrightarrow{a}s'_e$ and $s_f\xrightarrow{a}s'_f$.*

Proposition 3. *Let $\sim_e$ and $\sim_f$ be two equivalences on $Action^*$ induced by classifiers e and f. Their greatest lower bound $\sim_e\sqcap\sim_f$ exists and is induced by $e\otimes f$.*

Figure 5 presents the sum $\mathcal{E}+\mathcal{F}$ obtained by application of operational rules of section 5 (SG,PO) and the above proposition. $\mathcal{E}+\mathcal{F}$ distinguishes four classes of outputs: an empty sequence, sequences consisting of occurrences of a_1, consisting of occurrences of a_2, and containing occurrences of both a_1 and a_2.

The least upper bound of two partitionings $\sim_e\sqcup\sim_f$ is usually computed using a Union-Find algorithm, which unifies any two overlapping classes, until all classes are disjoint. In our case classes are represented by states in the classifiers e and f. We need to apply the algorithm to states of e and f, ultimately producing a classifier, whose states are sets of states of f and e. The two classes s_1 and s_2 overlap, whenever there is an output sequence, that can advance one classifier to a state in s_1, and the other classifier to a state in s_2. The initial set of classes is given by reachable states of the product classifier $e\otimes f$:

i. $S := \{\{e_i, f_j\} \mid (e_i, f_j)$ is reachable in $e \otimes f\}$.
ii. If there exist $s_1, s_2 \in S$ such that $s_1 \cap s_2 \neq \emptyset$ then $S := S \setminus \{s_1, s_2\} \cup \{s_1 \cup s_2\}$.
iii. Repeat (ii) until no more classes can be unified.

The final value of S is the set of states of the new classifier DFA. The initial state is the class that contains initial states of e and f (note that both of them will be in the same class). The transition function $\rightarrow$ is a sum of transition functions $\rightarrow_e$ and $\rightarrow_f$ lifted to sets of states. For $s_1, s_2 \in S$: $s_1 \xrightarrow{a} s_2$ if $\exists . p_1 \in S_1 . \exists p_2 \in S_2 . p_1 \xrightarrow{a}_e p_2 \vee p_1 \xrightarrow{a}_f p_2$. The following proposition claims that this function is well-defined, deterministic and input-enabled:

Proposition 4. *Let $s_1, s_2 \in S$ be any two of the sets of states (not necessarily distinct) constructed with the above algorithm. Then for any states $p_1, p_2 \in s_1$, $p'_1, p'_2 \in s_2$ of the original classifiers and any symbol a: $p_1 \xrightarrow{a}_1 p'_1$ and $p'_1 \in s_2$ iff $p_2 \xrightarrow{a}_2 p'_2$ and $s'_2 \in s_2$, where $\longrightarrow_i$ denotes $\longrightarrow_e$ if $s_i \in S_e$ or $\longrightarrow_f$ if $s_i \in S_f$.*

It follows that the classifier $g = (S, A, s, \longrightarrow)$ constructed above is a well defined classifier DFA. Moreover, the observation classes that it induces are coarser than any class of $\sim_e$ and $\sim_f$. Due to the properties of the union-find algorithm, $\sim_g$ is actually the least equivalence encompassing both $\sim_e$ and $\sim_f$:

Proposition 5. *Let $\sim_e$ and $\sim_f$ be equivalences over Action*, induced by classifiers $e = (S_e, Action, s_e, \longrightarrow_e)$ and $f = (S_f, Action, s_f, \longrightarrow_f)$. The equivalence $\sim_e \sqcup \sim_f$ is induced by a classifier g such that its states are computed applying the* Union-Find *algorithm to the set $\{\ \{e_i, f_j\} \mid (e_i, f_j)$ reachable in $e \otimes f\ \}$, where two sets s_1,s_2 are unifiable if $s_1 \cap s_2$ is not empty. The union operation is a set union, the initial state is the set containing initial states of e and f, and the transition function is a sum of transition functions lifted to sets of states.*

The rightmost IOATS on Fig. 5 is a product of $\mathcal{E}$ and $\mathcal{F}$ obtained by application of the composition rules from section 5 (PG,SO) and the above algorithm. This product gives rise to the observer which does not distinguish any sequences.

7 Environment Driven Specialization

We shall now broaden the meaning of a model of a system to encompass a family of systems, and let it represent functionality, which in its entire richness may not be present in any of the actual members being produced. Particular family members will be specified using models of environments, and derived by transformations preserving relativized equivalence in a given color-blind environment. We shall informally demonstrate a product line derivation scenario, hinting at what techniques could be used to make such automatic derivation viable.

Our family will consist of several state/event systems. The transition relation of state/event systems (see section 2) produces sets of actions during a single reaction step. In such a setting the observational classes of environments become sets of sets (powersets) of actions.

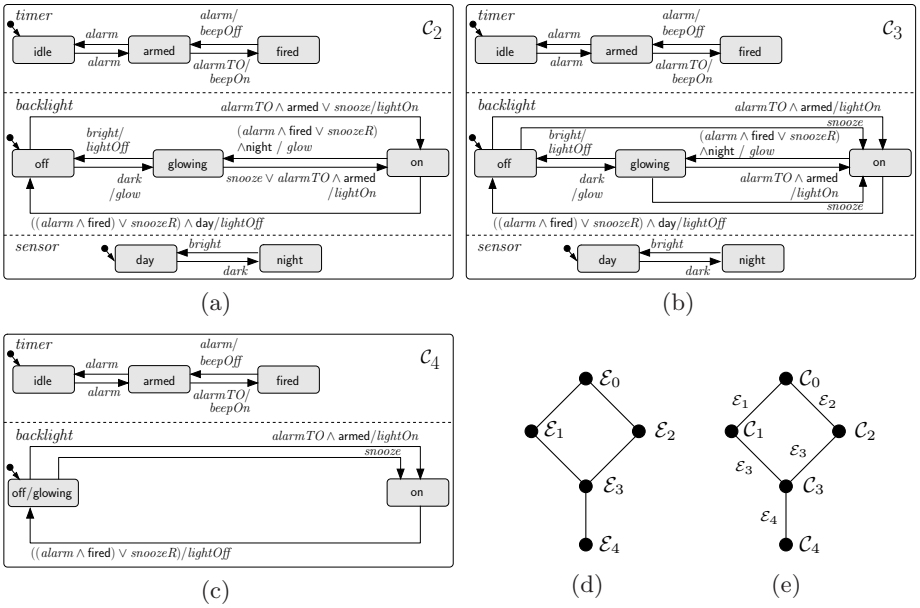

Fig. 6. Specialized models $\mathcal{C}_2$ (a), $\mathcal{C}_3$ (b) and $\mathcal{C}_4$ (c). Overview of the relationship between the environments (d) and the specialized models (e).

For a set $A \subseteq Action$ let *ignore* A denote observation classes, which ignore elements of A, but distinguish all the other actions:

$$ignore\ A = \left\{\ \{o \cup o' | o' \in \mathcal{P}(A)\}\ \middle|\ o \in \mathcal{P}(Action \setminus A)\right\}$$

Note that ignoring the empty set, *ignore* $\emptyset$, means observing all differences in outputs. Another abbreviation *equiv* A denotes observation classes, which are unable to distinguish between any actions in A:

$$equiv\ A = \left\{\ \{o \cup o' | o' \in \mathcal{P}(A) \setminus \emptyset\}\ \middle|\ o \in \mathcal{P}(Action \setminus A)\right\} \cup \left\{o\ \middle|\ o \in \mathcal{P}(Action \setminus A)\right\}$$

We shall begin with stating general requirements, which hold for all the environments used to execute the alarm clock. These general requirements usually reflect the physical nature of actuators and sensors. In the case of our alarm clock events *dark*/*bright* and *snooze*/*snoozeR* are always generated in an alternating fashion: $\mathcal{E}_0 =$ *Interleave snooze snoozeR* $\wedge$ *Interleave dark bright*. Figure 7a demonstrates how *Interleave* could be defined using a set-based semantics.

The first member of the family $\mathcal{C}_1$ was introduced in section 2 (Fig. 2). This model operates in an environment, which becomes blind for the *lightOn* action right after generating the *snooze* event. Formally $\mathcal{E}_1 = \mathcal{E}_0 \wedge \mathcal{E}'$, where $\mathcal{E}'$ is defined on Fig. 7b. Figure 6a presents a new clock $\mathcal{C}_2$, which is devoid of the actual snooze function. The user of this clock can still press the snooze button, but the only effect it has is turning the backlight on for a short while. This

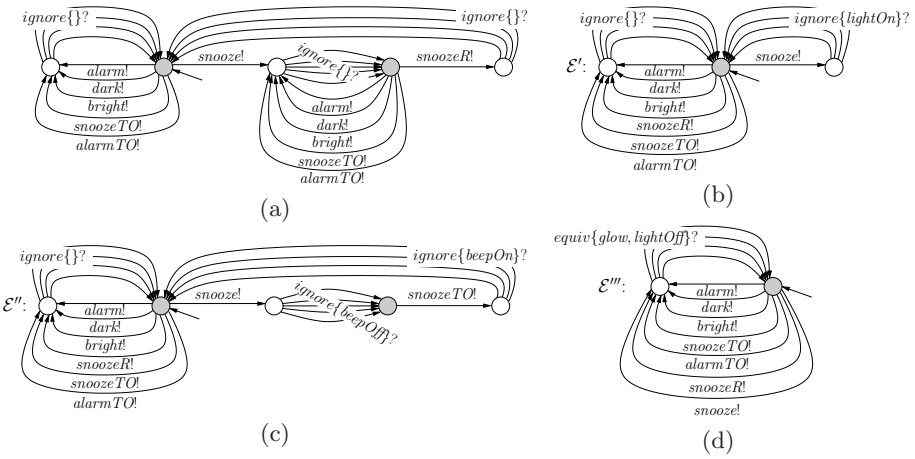

Fig. 7. (a) *Interleave snooze snoozeR.* (b) Environment $\mathcal{E}'$ ignoring the *lightOn* output produced in reaction to the *snooze* button. (c) Environment $\mathcal{E}''$ ignoring the snooze function of the clock. (d) Environment $\mathcal{E}'''$ *Equiv glow lightOff.*

user becomes blind to *beepOn* and *beepOff* actions initiated by the *snooze* and *snoozeTO* events. Formally $\mathcal{E}_2 = \mathcal{E}_0 \wedge \mathcal{E}''$, where $\mathcal{E}''$ is defined on Fig. 7c.

The third clock variant $\mathcal{C}_3$ is a combination of $\mathcal{C}_1$ and $\mathcal{C}_2$. It has neither the snooze function nor the snooze activated backlight function. We obtain it by specialization against the $\mathcal{E}_3$ environment, where $\mathcal{E}_3 = \mathcal{E}_1 \wedge \mathcal{E}_2$. The model is presented on Fig. 6b. Note that this clock still needs a snooze button, which exhibits a slight anomaly in turning on the glow mode, namely that the glow mode will not be activated, while this button is pressed. This is a perfectly correct reminiscence of our original model, which could be easily remedied by adding another constraint to the environment, that event *snooze* never occurs.

We would like to consider yet another restriction of the clock behavior. The clock denoted $\mathcal{C}_4$, shall be deprived of the glowing mode (Fig. 6c). The glow-mode lamp is not installed and the *glow* action is reimplemented to turn off the main lamp instead. A corresponding environment $\mathcal{E}'''$ is defined on Fig. 7d. This environment is itself interesting as it specifies a less shiny alarm clock, which may find its happy customers. Nevertheless, we decided to combine its characteristics with the restrictions of $\mathcal{E}_3$, giving rise to an even more simple alarm clock with neither the snooze related functions nor the glow mode: $\mathcal{E}_4 = \mathcal{E}_3 \wedge \mathcal{E}'''$.

One can describe surprisingly many more reasonable variants even for such a simple system as our alarm clock. Figures 6d-6e present an overview of environments and systems in our product line. Edges represent simulation and relativized simulation. Proposition 6 explains how to interpret transitivity in the hierarchy of systems (Fig. 6e).

Proposition 6. *For any systems $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ and any two compatible color-blind environments $\mathcal{E}$ and $\mathcal{F}$ it holds that:* $\mathcal{S}_1 \leqslant_{\mathcal{E}} \mathcal{S}_2 \wedge \mathcal{S}_2 \leqslant_{\mathcal{F}} \mathcal{S}_3 \wedge \mathcal{E} \leqslant \mathcal{F} \Rightarrow \mathcal{S}_1 \leqslant_{\mathcal{E}} \mathcal{S}_3$.

8 Related Work

Derivation of product lines is conventionally associated with partial evaluation [13, 4, 8]. There have been limited approaches to enable partial evaluation based on execution traces instead of fixed input values [10, 20, 5], nevertheless they were never implemented for realistic languages. We fear that these transformations, designed for abstract process calculi, can be barely applied in such contexts. This is why we intend to define transformations on the language level, and only prove correctness on the abstract level. Our framework allows more transformations than known before due to the color-blindness, which allows some non-reductive mutations in the program.

Previously Wasowski [25] presented a static framework for specifying environments for reactive models, which relies solely on state independent properties. The present paper provides a theoretical foundation for a product line management setup similar to Wasowski's [25], but based on behavioral properties.

Relativized simulation has been originally introduced by Larsen [16, 15, 14]. Our framework is modeled after this work, rephrased in the setting of I/O alternating transition systems and extended with the notion of color-blindness. In Larsen's formulation, based on simple labeled transition systems [19], it was impossible to express an environment's inability to distinguish outputs.

The study of behaviors of systems embedded into execution contexts is relatively mature [15, 1, 18, 22, 12]. Our work stems out from this series, by its extended support for observability specifications via color-blindness. This support is needed, if the tools based on this framework, are to be useful for development of product lines of embedded systems.

9 Conclusion & Future Work

We have presented the semantics of a specification language for environments of reactive synchronous systems, together with a notion of context-dependent refinement based on color-blindness. This refinement relation is more liberal than usual in allowing some mutations to program outputs, instead of bare reductions. We have explained and demonstrated how partial specifications of behaviors can be composed and used to define families of products. The framework was designed as a core of an upcoming tool for compact code generation and product line derivation for discrete control embedded systems. Our specifications shall be used as preconditions for advanced model optimizers/specializers. We have thoroughly discussed issues, which arise in the implementation of the theory for realistic languages, especially focusing on languages with sequences as outputs.

An implementation [23] of a powerful context-aware optimizer for models based on model-checking and program analysis is currently underway. This prototype tool is supposed to be compatible with an industrial development environment for embedded systems [11], which will allow for realistic case studies.

We would like to attempt a formulation of a corresponding theory for distributed asynchronous systems. We hope that a similarly appealing construction

can be proposed for such systems. The main difficulty appears to be a notion of simulation between nondeterministic color-blind environments. The simulation of definition 7 is too weak to imply theorem 1 in a nondeterministic setting.

References

1. L. de Alfaro and T. A. Henzinger. Interface automata. In *Foundations of Software Engineering (FSE)*, pp. 109–120, Vienna, September 2001. ACM Press.
2. G. Berry. The foundations of Esterel. In G. Plotkin, etal. eds., *Proof, Language and Interaction. Essays in Honour of Robin Milner*, pp. 425–454. MIT Press, 2000.
3. E. M. Clarke. *Model Checking*. The MIT Press, 1999.
4. O. Danvy et al. eds., *Partial Evaluation*, *LNCS* 1110, Feb. 1996. Springer-Verlag.
5. S. Etalle and M. Gabbrieli. Partial evaluation of concurrent constraint languages. *ACM Computing Surveys*, 30(3es), September 1998.
6. Rob van Glabbeek. The linear time–branching time spectrum In J.C.M Beaten and J.W. Klop eds., *CONCUR'90*, *LNCS* 458, Springer–Verlag
7. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
8. J. Hatcliff, T. Æ. Mogensen, and P. Thiemann, editors. *Partial Evaluation: Practice and Theory*, *LNCS 1706*. Springer-Verlag, Copenhagen, Denmark, 1999.
9. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
10. H. Hosoya, N. Kobayashi, A. Yonezawa. Partial evaluation scheme for concurrent languages and its correctness. In L. Bougé et al eds., *Euro-Par'96*, *LNCS* 1123.
11. IAR Inc. IAR visualSTATE®. `http://www.iar.com/Products/VS/`.
12. A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. In *POPL 2001*. ACM Press.
13. Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
14. K.G. Larsen and R. Milner. A compositional protocol verification using relativized bisimulation. *Information and Computation*, 99(1):80–108, 1992.
15. K. Larsen. *Context Dependent Bisimulation Between Processes*. PhD thesis, Edinburgh University, 1986.
16. K. Larsen. A context dependent equivalence between processes. *Theoretical Computer Science*, 49:184–215, 1987.
17. J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. Kristoffersen, and K. G. Larsen. Verification of large state/event systems using compositionality and dependency analysis. *Formal Methods in System Design*, 18(1):5–23, 2001.
18. N. Lynch. I/O automata: A model for discrete event systems. In *Annual Conference on Information Sciences and Systems*, pp. 29–38, Princeton, N.J., 1988.
19. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
20. M. Murakami. Partial evaluation of reactive communciating processes using temporal logic formulas. In *Algebraic and Object-Oriented Approaches to Software Science*, 1995.
21. Object Management Group. OMG Unified Modelling Language specification, 1999.
22. S. K. Rajamani, J. Rehof. Conformance checking for models of asynchronous message passing software. In *CAV'02*, LNCS 2404, Springer-Verlag.
23. Scope: a statechart compiler. `http://www.mini.pw.edu.pl/~wasowski/scope`.
24. Sun Microsystems, Inc. Java Card(TM) specification. `http://java.sun.com/`.
25. A. Wasowski. Automatic generation of program families by model restrictions. In *SPLC 2004*, LNCS 3154, Boston, USA, September 2004. Springer-Verlag.