Under consideration for publication in Formal Aspects of Computing

Composing and Refining Dense Temporal Logic Specifications[1]

Antonio Cau

Software Technology Research Laboratory, Science and Engineering Research Centre,
De Montfort University, The Gateway, Leicester LE1 9BH, UK

Keywords: Temporal Logic; Compositionality; Refinement

Abstract. A dense temporal logic development method for the specification, refinement, composition and verification of reactive systems is introduced. A reactive system is specified by a pair consisting of a machine and a condition that indicate the valid computations of this machine. Compositionality is achieved by adding to each machine step whether it is a environment, system or communication step. Refinement can be expressed straightforward in the logic because the stutter problem is elegantly solved by using the dense structure of the logic. Compositionality enables us to break refinement between complex systems into refinement between small and simple systems. The latter can then be verified by existing proof rules for refinement which are reformulated in our formalism.

1. Introduction

We present a compositional refinement method for reactive systems. A system is called reactive if it maintains some ongoing interaction with its environment, for example an operating system. This contrasts with transformational systems where from some input, without further interaction, output is produced. Because of this characteristic reactive systems are described as sets of behaviours (histories). Here we present a framework which can model both CSP based and shared variable based concurrency, using the work of [Sta84, BKP84, BKP86, DK90, KMP93]. We will use a *basis* to provide syntactic information about the

Correspondence and offprint requests to: Antonio Cau, Software Technology Research Laboratory, Science and Engineering Research Centre, De Montfort University, The Gateway, Leicester LE1 9BH, UK. E-mail: `cau@dmu.ac.uk`

[1] This work is an extended abstract of the author's Ph.D. thesis [Cau95].

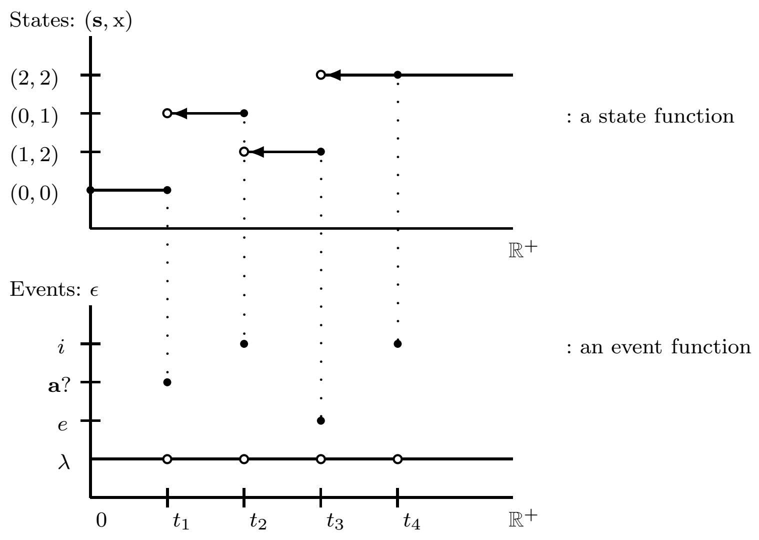

Fig. 1. Illustration of the notion of state and event function, which together characterise the notion of computation of a machine. It illustrates the following computation: initially $(\mathbf{s}, \mathrm{x}) = (0, 0)$, the event **a**? changes x into 1, i.e., **s** doesn't change. In the interval $[0, t_1)$ there are only λ events. Event i at point t_2 changes $(\mathbf{s}, \mathrm{x})$ into $(1, 2)$. At point t_3, the event e changes **s** into 2 and at point t_4, the event i doesn't change **s** or x.

channels and variables of the specified system. A history is a pair consisting of an event and a state function. The domains of these functions are the non-negative real numbers (the underlying dense model). The event function maps each non-negative real number to an event (an action occurring during the operation of the system and its environment) and the state function maps each real number to a state of the system and its environment. The intuition is that an occurrence of an action causes (potentially) a state change as illustrated in Figure 1 (Note: an λ action is a "special" action that doesn't change the state of the system).

The use of real numbers as domain for the event and state function handles the stutter problem in refinement. This problem, first observed by Lamport [Lam83], is as follows. Given two behaviours of a system, let the first behaviour contain only consecutive snap-shots of the system that differ from each other whereas the second behaviour, besides containing these same snapshots, contains additional consecutive duplicates of these. This is called stuttering. From the viewpoint of an observer these behaviours are considered as equivalent. Consequently, any formalism that allows to distinguish between these behaviours is not abstract enough and has a power of discrimination which is too strong w.r.t. the criterion of observable behaviour chosen. An example of such a too discriminating formalism is linear temporal logic with the next operator $\bigcirc$. In the present formalism this excessive expressive power is avoided as follows: state changes caused by events happen only now and then, so that in between each two consecutive changes there are uncountably many instants of time at which *nothing* happens. Consequently, it is impossible to count, or express, stutter steps because the model is "saturated" with them. Furthermore the use of real numbers for defining the event and state function enables us to express hiding of variables as existential quantification and consider refinement as implication,

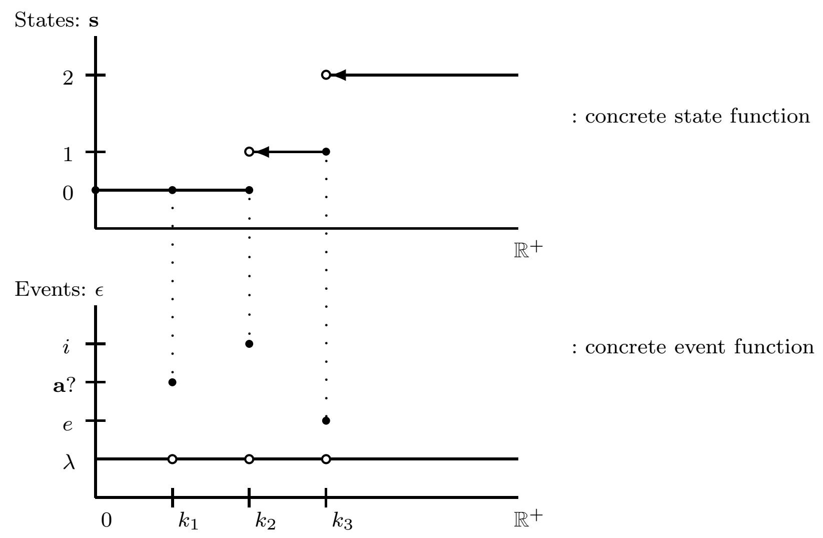

Fig. 2. Illustration of the following concrete computation: initially $\mathbf{s} = (0)$, the event $\mathbf{a}?$ doesn't change $\mathbf{s}$, the event i changes $\mathbf{s}$ into 1, and event e changes $\mathbf{s}$ into 2.

even if there are more "states" on the abstract level than on the concrete level. Let the history illustrated in Figure 1 be a history at the abstract level where x is the variable that should be hidden and let the history illustrated in Figure 2 describe the concrete level. The history of Figure 2 is a refinement of the history of Figure 1. In [FM92, FM94] refinement takes place between different logics (each corresponding to a level of granularity) and not within a single logic that accommodates all levels of granularity. It is argued that one single logic plus substitution-based techniques, as used in this paper, leads to stuttering or dense structures.

The assumption/commitment approach is used to achieve compositionality in CSP based concurrency [MC81] and shared variable concurrency [Jon83]. These two approaches are unified in [CC96]. These unification ideas are used here, i.e., we use an event variable to store "compositionality information" like "this is a system step" or "this is an environment step" or "this is a communication step". The use of event variable is inspired by the work of [BKP84]. *This enables us to describe parallel composition of reactive systems by conjunction.* Note that in for instance Lamport's work on TLA [Lam94, AL93] this is not always the case: $x := 1 \| x := 1$ must be modelled as disjunction because conjunction leads to a "one process" specification $x := 1$. In our model however, it can be modelled as conjunction because the specification of one component also contains environmental information, especially about the other component. With a "conjoining" operator the histories of both components are merged into a history of the composite one. The merge operator will "pair" events that occur at the same time, e.g., an environment event in one history with an system event in the other history. This conjoining operator, which is based on [CC96], corresponds in our model almost to conjunction. This operator is an extended version of Aczel's parallel composition operator [Acz83], it can also handle CSP based concurrency whereas Aczel's can only handle shared variable based concurrency.

We also investigate how composition relates to refinement, i.e., in fact we obtain the notion of compositional refinement [ZCdR92]. Compositional refinement means intuitively that if *the components* of an abstract composed system are refined by *the components* of a concrete composed system then the total abstract composed system is refined by the total concrete composed system, i.e., refinement is preserved under composition. This enables the reduction of refinement problems for large, complex, systems to a number of refinement problems between small, simple, systems. In fact without this property we wouldn't have been able to prove correctness of some complicated examples like stable storage and Dijkstra's solution to the readers and writers problem.

Our dense temporal logic (DTL) with histories as semantic model is based on the work of [Sta84, BKP86, DK90, KMP93]. Other work that use dense structures include [Dil88, AL91b, AD94, Dil88, GSSAL94, LV95]. A salient feature of DTL is the "immediately after" operator "′", in a version which is stutter insensitive. The logic is used to (1) specify our systems, (2) express refinement and composition between systems, and (3) verify refinement between systems.

1.1. Structure of the Paper

In section 2 we will introduce the basic formal framework of histories, basis and the logic DTL for specifying reactive systems. In section 3 we will introduce the notion of refinement and composition of reactive systems and in section 4 we explain the proof technique for refinement of reactive systems together with an example illustrating this technique.

2. Specification of Reactive Systems

This section explains how reactive systems can be specified. Firstly they will be specified at the semantical level, i.e., by sets of histories. A history intuitively specifies which event occurs at a particular point and in what state the system is at that particular point. Secondly reactive systems are specified using the dense temporal logic DTL.

2.1. Semantic Specification of Reactive Systems

Reactive systems will be specified by sets of histories. A history is a pair consisting of an event function and a state function. An event function records at each point (i.e., for each positive real, including zero) which *event* occurs. An event is an instantaneous occurrence of an action during the operation of a system, that can be generated by that system or by its environment and that is of interest at the given level of abstraction. Four kinds of actions are distinguished:

1. *Communication* actions $\mathbf{a}?, \mathbf{b}!$, i.e., actions that transmit information over a channel. A channel is a connection between the system and its environment.
2. *System* actions i, i.e., non-communication actions of the system.
3. *Environment* actions e, i.e., non-communication actions of the environment.
4. *Silent* actions λ, i.e., actions that don't influence the status of the system.

Note: when we specify a system we will fix the set of actions beforehand in the basis, i.e., we can only specify static (with respect to events) systems. The Object calculus [FM92] allows a more general approach.

Event states are introduced in order to record which event occurs during the operation of the system. An event state corresponds to the usual notion of state in that instead of normal program variables *event variables* are used.

Definition 1 (Event state). Let Chan denote a nonempty set of channels names. Let $\mathfrak{E}$ denote the set of event variables with typical elements $\epsilon, \epsilon_0, \epsilon_1, \ldots$. Event variable ϵ will record which action occurs during the operation of the system, and the event variables $\epsilon_0, \epsilon_1, \ldots$ are auxiliary event variables recording which actions occur in components of the system. Let $\mathfrak{A}$ denote the set of actions, with typical elements i (denoting system actions), e (denoting environment actions), $\mathbf{a}?, \mathbf{b}!$[2] $\ldots$ denoting respectively an input communication action over channel $\mathbf{a}$ and an output communication action over channel $\mathbf{b}$, and λ denoting the silent action. An event state is a mapping δ from $\mathfrak{E}$ to $\mathfrak{A}$. Let Δ denote the set of all event states.

A state function records at each point (a non-negative real number) the *process state*, i.e., the usual notion of state of a system and its environment. In order to distinguish the normal variables from the event variables the normal variables are called here *process variables.* Three kind of process variables are distinguished:

1. *Shared* process variables which are "shared" between a system and its environment.
2. *Local* process variables which are only accessible by a system.
3. *Rigid* variables which are not changed by the system and its environment, i.e., which are only used for specification purposes.

Definition 2 (Process state). A process state is a mapping from variables to values. Let $\mathfrak{V}$ denote the set of shared variables with typical elements $\mathbf{s}, \ldots$, and $\mathfrak{X}$ the set of local variables ($\mathfrak{V} \cap \mathfrak{X} = \emptyset$) with typical elements $\mathrm{x}, \ldots$, and $\mathfrak{N}$ the set of rigid variables with typical elements $n, \ldots$. A state is a mapping σ from $\mathfrak{V} \cup \mathfrak{X} \cup \mathfrak{N}$ to the set of values Val. Let Σ denote the set of all process states.

As already said above, event and state functions are mappings from the non-negative reals to, respectively event and process states. Because of this strategy, some requirements are needed in order to specify "reasonable" histories. Here "reasonable" is used in the sense that in a bounded interval only a finite number of non-silent actions and process state changes can occur. This requirement is called the finite variability condition [BKP86]. Next several notions for functions from $\mathbb{R}^+$ (the positive reals including 0) to some domain D are introduced in order to define this requirement and to formally define the notions of event and state functions. Note: as argued in the introduction we need a dense structure, here in this paper we take $\mathbb{R}^+$ but in fact any dense structure (provided it has a first element and no last element) would do.

Definition 3 ((Dis)continuous). Given function $f : \mathbb{R}^+ \to D$. f is called
left continuous, if $\forall t > 0 \,.\, f(t) = \lim_{t_1 \to t} f(t_1)$,
right continuous, if $\forall t \geq 0 \,.\, f(t) = \lim_{t \leftarrow t_1} f(t_1)$,

[2] In this paper we omit the value part of the communication, i.e., which value is transmitted, in order to ease the formalism a little bit.

discontinuous at t, if $f(t) \neq \lim_{t_1 \to t} f(t_1) \vee f(t) \neq \lim_{t \leftarrow t_1} f(t_1)$,
strongly discontinuous at t, if $f(t) \neq \lim_{t_1 \to t} f(t_1) \wedge f(t) \neq \lim_{t \leftarrow t_1} f(t_1)$.
f has the *finite variability* property iff f has only finitely many points of discontinuity in any interval $[a, b]$, $0 \leq a \leq b$, $a, b \in \mathbb{R}^+$.

Now *event* and *state* functions can be defined. In [DK90] it is explained why initial stuttering is needed to express refinement in a logic with the help of existential quantification and implication. To adopt this result, we must first define what notion of stuttering in our setting corresponds with the one of [DK90]. Within our setting a stutter step is regarded as a step in which a non-communication action doesn't change the state. As such initial stuttering can expressed by requiring that in the first interval the event function has the constant value λ and the state function remains constant. Furthermore a state should remain constant for some continuous interval of points in order to be observable. Also non-λ events are considered to be single points. Another possibility would be for the events to remain constant during an interval of points. The intuitive meaning of a history is that the points at which non-λ events occur mark the state changes. For these non-λ events the question to be answered is: At which point of the interval should the state change take place? Answer: at the last point of the interval which describes the event. So for events only the last point of the interval is interesting because it marks the state change. So why consider an interval if only its last point is interesting? This is explained by our choice for letting non-λ events occur only at single points, as captured by the following definitions.

Definition 4 (restriction).
For $g : A_1 \to A_2$, $A_0 \subseteq A_1$ define $g|^1_{A_0} : A_0 \to A_2$ as $g|^1_{A_0}(x) = g(x)$ for $x \in A_0$.
For $g : A_1 \to (A_2 \to A_3)$, $A_0 \subseteq A_2$ define $g|^2_{A_0} : A_1 \to (A_0 \to A_3)$ as $g|^2_{A_0}(t)(x) = g(t)(x)$ for $x \in A_0$.

Definition 5 (Event function). An *event function* ψ is a function from $\mathbb{R}^+$ to Δ, such that $\psi|^2_\epsilon$ has the finite variability condition, $\psi(0)(\epsilon) = \lambda$ (i.e., initial stuttering) and for all points t, ψ is strongly discontinuous at t iff $\psi(t)(\epsilon) \neq \lambda$ (i.e., an event function is almost constant λ). Let Ψ denote the set of all event functions.

Figure 1 illustrates the notion of event function. At point t_1 event $\mathbf{a}?$ occurs, at point t_2 event i occurs, at point t_3 event e occurs, and at all other points event λ occurs. Points t_1, t_2 and t_3 are here the strongly discontinuous points.

Definition 6 (State function). A *state function* θ is a left continuous function from $\mathbb{R}^+$ to Σ such that for all $n \in \mathfrak{R}$ and $t \in \mathbb{R}^+$, $\theta(t)(n) = \theta(0)(n)$ (i.e., the rigid variables don't change at all), and for all $x \in \mathfrak{V} \cup \mathfrak{X}$, $\theta|^2_x$ satisfies the finite variability property and $\theta|^2_x(0)(x) = \lim_{0 \leftarrow t_1} \theta|^2_x(t_1)(x)$ (i.e., initial stuttering). Let Θ denote the set of all state functions.

Figure 1 illustrates the notion of state function. In interval $[0, t_1]$ the system is in state $(\mathbf{s}, \mathrm{x}) = (0, 0)$, in interval $(t_1, t_2]$ in state $(\mathbf{s}, \mathrm{x}) = (0, 1)$, in interval $(t_2, t_3]$ in state $(\mathbf{s}, \mathrm{x}) = (1, 2)$ and in interval (t_2, ∞) in state $(\mathbf{s}, \mathrm{x}) = (2, 2)$. The event i at t_4 is an illustration of a non-λ stutter step.

The following definition combines the notions of state function and event function into the notion of history. Two requirements are imposed on the combination of event and state function for it to result in a history. The first requirement is that silent actions don't give rise to process state changes. The second requirement is that communication actions don't change the shared variables; this requirement is imposed in order to model CSP-like processes [Hoa84] like processes.

Definition 7 (History). A *history* h is a pair $\langle \psi, \theta \rangle$, where ψ is an event function and θ is a state function s.t. a λ action doesn't change the values of variables from $\mathfrak{V} \cup \mathfrak{X}$, i.e., $\forall t : \psi(t)(\epsilon) = \lambda \rightarrow \theta(t) = \lim_{t \leftarrow t_1} \theta(t_1)$ and a communication action doesn't change the values of shared variables, i.e., $\forall t : \psi(t)(\epsilon) = \mathbf{a}? \vee \psi(t)(\epsilon) = \mathbf{a}! \rightarrow \theta(t)|^1_{\mathfrak{V}} = \lim_{t \leftarrow t_1} \theta(t_1)|^1_{\mathfrak{V}}$. Let $\mathcal{H}$ denote the set of all histories.

The following definition defines when a history is stutter equivalent to another history. A *history collapse* function $\natural_{\mathfrak{h}}(h)$ is introduced that takes a history and collapses it in such a way that the non-stutter steps only occur at discrete points (elements of $\mathbb{N}$) and at all remaining points stutter steps occur. Also a restricted version of the history stutter equivalence relation is defined, namely, restricted to the process state information. The last one will be used to define a "process state history stutter insensitive" logic DTL. Restricted to a special kind of formulae we obtain the "stutter insensitive" logic.

Definition 8 (History collapse, stutter equivalent). Given history $h \in \mathcal{H}$, the *history collapse* denoted $\natural_{\mathfrak{h}}(h)$ is a function from $\mathcal{H}$ to $\mathcal{H}$ defined as $\natural_{\mathfrak{h}}(h) \doteq h \circ \mathrm{di}(h)$ where $\mathrm{di}(h)$ is the *discretisation* bijection for h from $\mathbb{R}^+$ to $\mathbb{R}^+$ and is defined as follows:
Let $\mathrm{tt}(h, k)$ be the function from $\mathcal{H} \times \mathbb{N}$ to $\mathbb{R}^+$ that gives the point in $\mathbb{R}^+$ of the k-th change in h, formally:

$$\begin{aligned} &\mathrm{tt}(h, 0) = 0 \\ &\mathrm{tt}(h, k+1) \doteq \mathit{min}(t \mid t > \mathrm{tt}(h,k) \wedge (\psi(t)(\epsilon) \notin \{\lambda, i, e\} \vee \\ &\qquad\qquad\qquad \theta(t) \neq \lim_{\mathrm{tt}(h,k) \leftarrow t_1} \theta(t_1))) \end{aligned}$$

Let $nn(h)$ denote the number of non-stutter points of h. Then the discretisation bijection $\mathrm{di}(h)$ for h is defined as follows:

$$\mathrm{di}(h)(t) \doteq \begin{cases} \mathrm{tt}(h,k) + (t-k) * (\mathrm{tt}(h,k+1) - \mathrm{tt}(h,k)) & nn(h) < \infty \\ & \wedge\ 0 \le k < nn(h) \\ & \wedge\ k \le t \le k+1 \\ \mathrm{tt}(h,k) + (t-k) & nn(h) < \infty \\ & \wedge\ k = nn(h) \wedge k \le t \\ \mathrm{tt}(h,k) + (t-k) * (\mathrm{tt}(h,k+1) - \mathrm{tt}(h,k)) & nn(h) = \infty \wedge 0 \le k \\ & \wedge\ k \le t \le k+1 \end{cases}$$

The inverse discretisation of h is denoted $\mathrm{di}^{-1}(h)$. Given histories $h_0, h_1 \in \mathcal{H}$, h_0 is *history stutter equivalent* to h_1 denoted $h_0 \simeq_h h_1$ iff $nn(h_0) = nn(h_1)$ and $\theta_{\natural_{\mathfrak{h}}(h_0)} = \theta_{\natural_{\mathfrak{h}}(h_1)}$ and $\psi_{\natural_{\mathfrak{h}}(h_0)}(k) = \psi_{\natural_{\mathfrak{h}}(h_1)}(k), k \le nn(h_0)$, i.e., the number of non-stutter steps should be equal, the state information should be equal in both collapsed histories and the event information should be equal in the points of non-stuttering. h_0 is *history process state stutter equivalent* to h_1 denoted $h_0 \simeq_{\theta_h} h_1$ iff $nn(h_0) = nn(h_1)$ and $\theta_{\natural_{\mathfrak{h}}(h_0)} = \theta_{\natural_{\mathfrak{h}}(h_1)}$, i.e., only the process state information is considered.

These notions are best understood by an example. What's done is that by stretching, respectively., compressing an interval as a rubber string, the non-stuttering

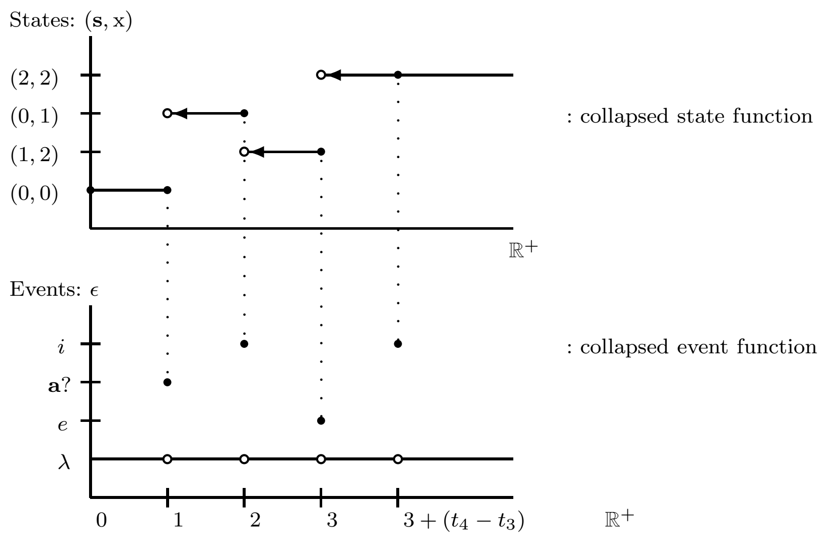

Fig. 3. Illustration of the collapsed history of Figure 1.

events are made to occur at an initial interval of the positive natural numbers, as shown in Figure 3.

The basis provides syntactic info and consists of a *process signature*, specifying the local and shared variables of the system, and a *action signature* which specifies the input and output communication channels of a system. The following definition introduces the notions of basis and history sets. The latter constrain a specific process signature, i.e., only its own set of shared variables and local variables are constrained to change in specific ways, whereas the variables outside this process signature can change without restriction, i.e., a history set should not restrict (specify) variables outside its own process signature.

Definition 9 (Basis, history set constraining a process signature). A *basis* (denoted by B) is a pair (B^A, B^P), where B^A (called *action signature*) is a pair $(\mathrm{In}, \mathrm{Out})$ where In is a set of input communication channels and Out is a set of output communication channels, and where B^P (called *process signature*) is a tuple (V, X) where V a finite set of shared variables and X a finite set of local variables. Given a history $h \in \mathcal{H}$ and process signature B^P, the *process signature restriction* of h denoted $h|^2_{B^P}$ is defined as $\langle \psi, \theta|^2_{\mathrm{V} \cup \mathrm{X}} \rangle$. Given a set of histories H and process signature B^P, H *is constrained by* B^P iff $\forall h_1, h_2 \in \mathcal{H} : h_1|^2_{B^P} = h_2|^2_{B^P} \rightarrow (h_1 \in H \leftrightarrow h_2 \in H)$.

Next we introduce the notion of *history specification* as a pair consisting of a basis and a set of histories constraining the process signature.

Definition 10 (History specification of a system). A *history specification* of a system (denoted $\mathcal{S}$) is a pair (B, H) where B is a basis and H is a set of histories constraining process signature B^P such that an environment action e doesn't change the local variables of the system: $\forall t : \psi(t)(\epsilon) = e \rightarrow \theta(t)|^1_{\mathrm{X}} = \lim_{t \leftarrow t_1} \theta(t_1)|^1_{\mathrm{X}}$.

We also borrow several notions from topology for the definition of safety and liveness sets of histories, based on [AS85]. Informally, a safety set of histories consists of histories where nothing "bad" happens and a liveness set of histories consists of histories where something "good" eventually happens.

Definition 11 (Safety and liveness set).
Let H be a set of histories and $h \in \mathcal{H}$.
The prefix of h of length t, denoted $h\downharpoonleft_t$, is defined as

$$h\downharpoonleft_t(t_0) \doteq \begin{cases} \langle \psi(t_0), \theta(t_0) \rangle & 0 \leq t_0 \leq t \\ \langle \psi(0), \theta(t) \rangle & t_0 > t \end{cases}$$

The *distance function* d from $\mathcal{H} \times \mathcal{H} \to \mathbb{R}^+$ is defined as:

$$d(h_1, h_2) \doteq \begin{cases} 0 & \text{if } h_1 = h_2 \\ 1 & \text{if } h_1(0) \neq h_2(0) \\ 2^{-\sup\{t \in \mathbb{R}^+ \mid h_1\downharpoonleft_t = h_2\downharpoonleft_t\}} & \text{otherwise} \end{cases}$$

H is called *d-open* iff $\forall h \in H : \exists \varepsilon > 0 : \forall h_1 : d(h, h_1) < \varepsilon \to h_1 \in H$.
The topology with $\{H \subseteq \mathcal{H} \mid H \text{ is } d\text{-open}\}$ as its basis is called the *d induced topology* of $(\mathcal{H}, d)$ denoted τ_d.
H is called a *τ_d-environment* of h iff $\exists H_1 \in \tau_d : h \in H_1 \wedge H_1 \subseteq H$.
The *interior* of H denoted $in(H)$ is defined as $\{h \in \mathcal{H} \mid H \text{ is a } \tau_d \text{ environment of } h\}$.
The *closure* of H denoted $cl(H)$ is defined as $\mathcal{H} \setminus (in(\mathcal{H} \setminus H))$.
H is a *safety set* iff $cl(H) = H$. H is a *liveness set* iff $cl(H) = \mathcal{H}$.

Note: the only set that is both a safety and a liveness set is $\mathcal{H}$ [AS85].

Following Abadi and Lamport [AL91a], a specification method for systems uses the notion of machine. A machine consists of a set of states and a state-transition relation. Our intention is that the set of computations (i.e. histories) of a machine used for describing a system should correspond to the history specification of this system. A machine however can only generate safety sets of histories [AS87]. Therefore, a liveness set is specified as a condition on the set of computations (histories) of a machine. Next the formal definition of a machine is given.

Definition 12 (Machine). The machine specification M of a system is a triple (B, I, T) where:

- B: the *basis* of M; a tuple $((\mathrm{In}, \mathrm{Out}), (\mathrm{V}, \mathrm{X}))$. Note: the shared variables will be printed in bold faced style in order to distinguish them from the local variables.
- I : a non-empty subset of Σ, the set of initial states, such that
  - $\forall \sigma_0, \sigma_1 \in \Sigma : (\sigma_0|^1_{\mathrm{V} \cup \mathrm{X}} = \sigma_1|^1_{\mathrm{V} \cup \mathrm{X}}) \to (\sigma_0 \in I \leftrightarrow \sigma_1 \in I)$, i.e., it constrains the variables from $\mathrm{V} \cup \mathrm{X}$ only.
- T : the state-transition relation (finite), $T \subseteq \Delta \times \Sigma^2$, such that
  - $\forall \sigma_0, \sigma_1 \in \Sigma, \delta \in \Delta : \langle \delta, \sigma_0, \sigma_1 \rangle \in T \to \sigma_0|^1_{\mathfrak{R}} = \sigma_1|^1_{\mathfrak{R}}$, i.e., the rigid variables don't change at all.
  - $\forall \sigma_0, \sigma_1, \sigma_2, \sigma_3 \in \Sigma, \delta \in \Delta : (\sigma_0|^1_{\mathrm{V} \cup \mathrm{X}} = \sigma_2|^1_{\mathrm{V} \cup \mathrm{X}} \wedge \sigma_1|^1_{\mathrm{V} \cup \mathrm{X}} = \sigma_3|^1_{\mathrm{V} \cup \mathrm{X}}) \to (\langle \delta, \sigma_0, \sigma_1 \rangle \in T \leftrightarrow \langle \delta, \sigma_2, \sigma_3 \rangle \in T)$, i.e. T constrains B^P only.
  - $\forall \sigma_0, \sigma_1 \in \Sigma, \delta \in \Delta : (\langle \delta, \sigma_0, \sigma_1 \rangle \in T \wedge (\delta(\epsilon) = \mathbf{a}? \vee \delta(\epsilon) = \mathbf{a}!)) \to \sigma_0|^1_{\mathrm{V}} = \sigma_1|^1_{\mathrm{V}}$, i.e., a communication action doesn't change the values of shared variables, and

- $\forall \sigma_0, \sigma_1 \in \Sigma, \delta \in \Delta : (\langle \delta, \sigma_0, \sigma_1 \rangle \in T \wedge \delta(\epsilon) = e) \rightarrow \sigma_0|^1_{\mathrm{X}} = \sigma_0|^1_{\mathrm{X}}$, i.e., an environment action doesn't change the values of local variables of the system.
- $\forall \sigma_0, \sigma_1 \in \Sigma, \delta \in \Delta : \langle \delta, \sigma_0, \sigma_1 \rangle \in T \rightarrow (\delta(\epsilon) \notin \{\lambda, i, e\} \vee \sigma_0 \neq \sigma_1\ )$, i.e., no stutter transitions are specified.

Example 1. $M = (B, I, T)$ where the basis $B \doteq ((\{\mathbf{a}\}, \emptyset), (\{\mathbf{v}\}, \{\mathrm{u}\}))$, the initial states $I \doteq \{\sigma \in \Sigma \mid \sigma(\mathrm{u}) = 0$ and $\sigma(\mathbf{v}) = 0\}$ and the transition relation $T \doteq \{\langle \delta, \sigma_0, \sigma_1 \rangle \in \Delta \times \Sigma^2 \mid$

1. $(\delta(\epsilon) = \mathbf{a}?$ and $\sigma_0(\mathrm{u}) = 0$ and $\sigma_1(\mathrm{u}) = 1$ and $\sigma_1(\mathbf{v}) = \sigma_0(\mathbf{v}))$ or
2. $(\delta(\epsilon) = i$ and $\sigma_0(\mathrm{u}) = 1$ and $\sigma_0(\mathbf{v}) = 1$ and $\sigma_1(\mathrm{u}) = 2$ and $\sigma_1(\mathbf{v}) = 0)$ or
3. $(\delta(\epsilon) = e$ and $\sigma_1(\mathrm{u}) = \sigma_0(\mathrm{u})$ and $\sigma_1(\mathbf{v}) = \sigma_0(\mathbf{v}) + 1)\}$

The concepts of event and state functions are related by the notion of *computation* of a machine M. A computation of M intuitively expresses that an event function and a state function fit together in that at any point t any triple consisting of (1) the event occurring at t, (2) the state just before and including t, and (3) the state just after t, belongs to the state transition relation of M (see fig. 1). Because such a relation doesn't contain stutter steps but histories do, a set of stutter transitions should be defined in order to relate machine computations to histories.

Definition 13 (Computation).
Let $h = \langle \psi, \theta \rangle \in \mathcal{H}$, $t \in \mathbb{R}^+$ and $M = (B, I, T)$.
The *step* occurring at t in h is defined as: $Step_h(t) \doteq \langle \psi(t), \theta(t), \lim_{t \leftarrow t_1} \theta(t_1) \rangle$.
The set of *stutter* steps, is defined as $\mathrm{STU} \doteq \{\langle \delta, \sigma_0, \sigma_1 \rangle \mid \delta(\epsilon) \in \{\lambda, i, e\} \wedge \sigma_0 = \sigma_1\}$.
A *computation* of M is a history h s.t. $\theta(0) \in I$ and $\forall t : Step_h(t) \in T \vee Step_h(t) \in \mathrm{STU}$.
$Comp(M)$ denotes the set of all computations of M.

Lemma 1 (Machine is safety). Let M be a machine then $Comp(M)$ is a safety set.

The *machine specification* of a system now consists of a machine M and a set of histories L constraining the process signature of this machine such that the closure of the intersection of $Comp(M)$ and L equals $Comp(M)$. This is the *machine closedness* property of a system specification introduced in [AFK88, AL91a]. Let $A \rightarrow B$ denote $\bar{A} \bigcup B$. By a result of [AS85] every set of histories can be written as the intersection of a safety set and a liveness set namely $cl(Comp(M) \cap L) \bigcap cl(Comp(M) \cap L) \rightarrow (Comp(M) \cap L)$. By the machine closedness property this can be written as $Comp(M) \bigcap Comp(M) \rightarrow L$. This implies that $Comp(M)$ specifies the safety properties and $Comp(M) \rightarrow L$ the liveness properties of the system.

Definition 14 (Machine specification). A *machine specification* $\mathcal{S}$ of a system is a pair $(B, Comp(M) \cap L)$ where M is a machine with basis B and L a set of histories constraining only B^P such that $cl(Comp(M) \cap L) = Comp(M)$. The set of computations of $\mathcal{S}$, denoted $Comp(\mathcal{S})$, is defined as $Comp(M) \cap L$.

Table 1. Syntax of DTL. Let value $\mu \in Val$, rigid variable $n \in \mathfrak{R}$, observable variable $\mathbf{v} \in \mathfrak{V}$, local variable $\mathrm{x} \in \mathfrak{X}$, event variable $\epsilon \in \mathfrak{E}$ and channel $\mathbf{a} \in \mathrm{Chan}$.

	Rigid Expressions
$rexp ::=$	$\mu \mid n \mid n' \mid {}^{\backprime}n \mid rexp_1 + rexp_2 \mid \ldots$
	Expressions
$exp ::=$	$rexp \mid \mathbf{v} \mid \mathbf{v}' \mid {}^{\backprime}\mathbf{v} \mid \mathrm{x} \mid \mathrm{x}' \mid {}^{\backprime}\mathrm{x} \mid exp_1 + exp_2 \mid \ldots$
	Event Expressions
$evexp ::=$	$\mathbf{a}? \mid \mathbf{a}! \mid i \mid e \mid \lambda \mid \epsilon \mid \epsilon' \mid {}^{\backprime}\epsilon$
	Temporal formulae
$p ::=$	$\mathbf{true} \mid exp_1 = exp_2 \mid exp_1 < exp_2$ $\mid evexp_1 = evexp_2 \mid \neg p \mid p_1 \vee p_2$ $\mid p_1\ \widehat{\mathcal{U}}\ p_2 \mid p_1\ \widehat{\mathcal{S}}\ p_2 \mid \exists \mathrm{x}.p \mid \exists \epsilon.p \mid \exists n.p$

2.2. DTL Specification of Reactive Systems

As mentioned above, local properties are described by a machine and liveness properties as a set of histories. The dense temporal logic DTL, with syntax listed in table 1, describes both kinds of properties. To suit the main purpose of achieving a formalism for compositional refinement our logic is a carefully composed mixture of the dense temporal logics defined in [Sta84, BKP86, DK90, KMP93].

The ${}^{\backprime}$- and $'$-priming of a variable denotes respectively the previous and next value of the variable whereas the unprimed variable denotes the current value. $p_1\ \widehat{\mathcal{U}}\ p_2$ denotes strict (present not included in the future) until operator from temporal logic and $p_1\ \widehat{\mathcal{S}}\ p_2$ denotes strict (present not included in the past) since operator from temporal logic. $\exists \mathrm{x}.p$ denotes *existential quantification over local variable* x of p, i.e., hiding. A *state expression* is an expression without any primed variables. A *state formula* is a formula build from state expressions without $\widehat{\mathcal{U}}$ and $\widehat{\mathcal{S}}$ operators. Table 2 lists some frequently used abbreviations:

Example 2 (Some DTL formulae). $(\epsilon = \mathbf{a}_0 \wedge \mathrm{x} = 0 \wedge \mathrm{x}' = 1)$ (a state-transition), $\Box \mathrm{x} > 0$ (a safety property), and $\Box(\mathrm{x} = 0 \rightarrow \Diamond \mathrm{x} > 0)$ (a liveness property).

Before we give the semantics of DTL formulae we define for a variable x (local or event) the x-variant of a history.

Definition 15 (x-variant of a history). Let $h, h_1 \in \mathcal{H}$, $w \in \mathfrak{X} \cup \mathfrak{R}$ and $\epsilon \in \mathfrak{E}$. h_1 is a w-variant of h if $\psi_1 = \psi$ and $\theta_1|^2_{(\mathfrak{V}\cup\mathfrak{X}\cup\mathfrak{R})\setminus\{w\}} = \theta|^2_{(\mathfrak{V}\cup\mathfrak{X}\cup\mathfrak{R})\setminus\{w\}}$.
h_1 is a ϵ-variant of h if $\psi_1|^2_{\mathfrak{E}\setminus\{\epsilon\}} = \psi|^2_{\mathfrak{E}\setminus\{\epsilon\}}$ and $\theta_1 = \theta$.

In the following definition the semantics of DTL is given By convention, boolean values are not explicitly denoted, i.e., we shall write $(h, t) \models \mathbf{true}$ rather than $(h, t) \models \mathbf{true} \doteq tt$.

Definition 16 (Semantics of DTL).

Table 2. Frequently used abbreviations

$\widehat{\Diamond} p$	$\doteq$ **true** $\widehat{\mathcal{U}}\ p$	strict eventually p,
$\widehat{\Box} p$	$\doteq \neg \widehat{\Diamond} \neg p$	strict always p,
$\bigcirc p$	$\doteq p\ \widehat{\mathcal{U}}$ **true**	for some time in the future p holds without interruption,
$\Diamond p$	$\doteq p \vee \widehat{\Diamond} p$	non-strict eventually,
$\Box p$	$\doteq p \wedge \widehat{\Box} p$	non-strict always,
$p_1\ \mathcal{U}\ p_2$	$\doteq p_2 \vee (p_1 \wedge (p_1\ \widehat{\mathcal{U}}\ p_2))$	non-strict until,
$\widehat{\diamondminus} p$	$\doteq$ **true** $\widehat{\mathcal{S}}\ p$	strict once p in the past,
$\widehat{\boxminus} p$	$\doteq \neg \widehat{\diamondminus} \neg p$	strict has-always-been p,
$\ominus p$	$\doteq p\ \widehat{\mathcal{S}}$ **true**	for some time in the past p held without interruption,
$\widetilde{\ominus} p$	$\doteq \neg \ominus \neg p$	recently p held,
first	$\doteq \widetilde{\ominus}$**false**	first position in a history,
$\diamondminus p$	$\doteq p \vee \widehat{\diamondminus} p$	non-strict once in the past,
$\boxminus p$	$\doteq p \wedge \widehat{\boxminus} p$	non-strict has-always-been,
$p_1\ \mathcal{S}\ p_2$	$\doteq p_2 \vee (p_1 \wedge (p_1\ \widehat{\mathcal{S}}\ p_2))$	non-strict since,
$p_1 \Rightarrow p_2$	$\doteq \Box(p_1 \rightarrow p_2)$	p_1 entails p_2
$p_1 \Leftrightarrow p_2$	$\doteq \Box(p_1 \leftrightarrow p_2)$	p_1 is congruent p_2

Let $h = \langle \psi, \theta \rangle \in \mathcal{H}$, $t \in \mathbb{R}^+$, $n \in \mathfrak{N}$, $\mathbf{v} \in \mathfrak{V}$, $\mathrm{x} \in \mathfrak{X}$, and $\epsilon \in \mathfrak{E}$. Let $\mathcal{E}$ and $\mathcal{EE}$ be the usual valuation functions for respectively expressions and event expressions.

$$\begin{array}{lll}
\mathcal{E}_{(h,t)}[\mu] \doteq \mu & \mathcal{EE}_{(h,t)}[\mathbf{a}?] \doteq \mathbf{a}? & \mathcal{EE}_{(h,t)}[\mathbf{a}!] \doteq \mathbf{a}! \\
\mathcal{EE}_{(h,t)}[i] \doteq i & \mathcal{EE}_{(h,t)}[e] \doteq e & \mathcal{EE}_{(h,t)}[\lambda] \doteq \lambda \\
\mathcal{E}_{(h,t)}[n] \doteq \theta(0)(n) & \mathcal{E}_{(h,t)}[n'] \doteq \theta(0)(n) & \mathcal{E}_{(h,t)}[\grave{n}] \doteq \theta(0)(n) \\
\mathcal{E}_{(h,t)}[\mathrm{x}] \doteq \theta(t)(\mathrm{x}) & \mathcal{E}_{(h,t)}[\mathbf{v}] \doteq \theta(t)(\mathbf{v}) & \mathcal{EE}_{(h,t)}[\epsilon] \doteq \psi(t)(\epsilon) \\
\mathcal{E}_{(h,0)}[\grave{\mathrm{x}}] \doteq \theta(0)(\mathrm{x}) & \mathcal{E}_{(h,0)}[\grave{\mathbf{v}}] \doteq \theta(0)(\mathbf{v}) & \mathcal{EE}_{(h,0)}[\grave{\epsilon}] \doteq \psi(0)(\epsilon) \\
\text{for } t > 0: & & \\
\mathcal{E}_{(h,t)}[\grave{\mathrm{x}}] \doteq \lim\limits_{t_1 \rightarrow t} \theta(t_1)(\mathrm{x}) & \mathcal{E}_{(h,t)}[\grave{\mathbf{v}}] \doteq \lim\limits_{t_1 \rightarrow t} \theta(t_1)(\mathbf{v}) & \mathcal{EE}_{(h,t)}[\grave{\epsilon}] \doteq \lim\limits_{t_1 \rightarrow t} \psi(t_1)(\epsilon) \\
\mathcal{E}_{(h,t)}[\mathrm{x}'] \doteq \lim\limits_{t \leftarrow t_1} \theta(t_1)(\mathrm{x}) & \mathcal{E}_{(h,t)}[\mathbf{v}'] \doteq \lim\limits_{t \leftarrow t_1} \theta(t_1)(\mathbf{v}) & \mathcal{EE}_{(h,t)}[\epsilon'] \doteq \lim\limits_{t \leftarrow t_1} \psi(t_1)(\epsilon)
\end{array}$$

$\mathcal{E}_{(h,t)}[exp_1 + exp_2] \doteq \mathcal{E}_{(h,t)}[exp_1] + \mathcal{E}_{(h,t)}[exp_2]$
$\mathcal{E}_{(h,t)}[exp_1 - exp_2] \doteq \mathcal{E}_{(h,t)}[exp_1] - \mathcal{E}_{(h,t)}[exp_2]$
$(h,t) \models exp_1 = exp_2$ iff $\mathcal{E}_{(h,t)}[exp_1] = \mathcal{E}_{(h,t)}[exp_2]$
$(h,t) \models evexp_1 = evexp_2$ iff $\mathcal{EE}_{(h,t)}[evexp_1] = \mathcal{EE}_{(h,t)}[evexp_2]$
$(h,t) \models exp_1 < exp_2$ iff $\mathcal{E}_{(h,t)}[exp_1] < \mathcal{E}_{(h,t)}[exp_2]$
$(h,t) \models$ **true**
$(h,t) \models \neg p$ iff $(h,t) \not\models p$
$(h,t) \models p_1 \vee p_2$ iff $(h,t) \models p_1$ or $(h,t) \models p_2$
$(h,t) \models p_1\ \widehat{\mathcal{U}}\ p_2$ iff exists a $t_0 > t, (h,t_0) \models p_2$ and for all $t_1 \in (t,t_0), (h,t_1) \models p_1$
$(h,t) \models p_1\ \widehat{\mathcal{S}}\ p_2$ iff exists a $t_0 < t, (h,t_0) \models p_2$ and for all $t_1 \in (t_0,t), (h,t_1) \models p_1$
$(h,t) \models \exists \mathrm{x}.p$ iff $(h_1,t) \models p,$ for some $h_1,$ a x-variant of h
$(h,t) \models \exists \epsilon.p$ iff $(h_1,t) \models p,$ for some $h_1,$ a ϵ-variant of h
$(h,t) \models \exists n.p$ iff $(h_1,t) \models p,$ for some $h_1,$ a n-variant of h

Definition 17 (Satisfiability, validity).
Let p be a DTL formula, h be a history, and $\mathcal{S}$ be a system with basis B.
h *satisfies* p, denoted $h \models p$, iff $(h,0) \models p$.
p is *satisfiable* iff $h \models p$ for some history $h \in \mathcal{H}$.

p is *valid*, denoted $\models p$, iff $h \models p$ for all histories $h \in \mathcal{H}$.
p is $\mathcal{S}$-*valid*, denoted $\mathcal{S} \models p$, iff $h \models p$ for all histories $h \in Comp(\mathcal{S})$.
$Hist(p)$ denotes the set of all histories satisfying p.

The following theorem states that the logic DTL is history process state stutter insensitive. Later on a restricted stutter insensitive version of DTL is considered.

Theorem 1 (DTL is stutter insensitive).
For expression *exp*, event expression *evexp* and DTL formula p :

$$\begin{array}{l}
\forall t, h_0, h_1 : h_0 \simeq_{\theta_h} h_1 \rightarrow (\mathcal{E}_{(h_0,t)}[exp] = \mathcal{E}_{(h_1,\mathrm{di}(h_1)\circ\mathrm{di}^{-1}(h_0)(t))}[exp]) \\
\forall t, h_0, h_1 : h_0 \simeq_{\theta_h} h_1 \rightarrow (\mathcal{E}\mathcal{E}_{(h_0,t)}[evexp] = \mathcal{E}\mathcal{E}_{(h_1,\mathrm{di}(h_1)\circ\mathrm{di}^{-1}(h_0)(t))}[evexp]) \\
\forall t, h_0, h_1 : h_0 \simeq_{\theta_h} h_1 \rightarrow ((h_0, t) \models p \text{ iff } (h_1, \mathrm{di}(h_1) \circ \mathrm{di}^{-1}(h_0)(t)) \models p)
\end{array}$$

The proof system for DTL listed in tables 3 and 4 is inspired on [Bur82, Bur84, BKP86, KMP93]. Furthermore a link with the proof system of [KMP93] is established via axioms $Ax7b$–$Ax7f$, i.e., since these axioms are needed for deriving their proof system. Note: the models of [Bur82, Bur84] need not to satisfy the finite variability condition, the persistence conditions (once within an interval "going a little bit back or forward" doesn't bring you outside that interval), and the induction axiom. This reflects the crucial difference between the model of [KMP93] and ours on the one side and the model of [Bur82, Bur84] on the other side. The difference between the model of [KMP93] and our model is that we introduced additional compositionality information reflected by axioms $Ax0$, $Ax5$ and $Ax6$.

The proof system is for the pure logic, i.e., it is not meant for a specific reactive system. Axiom $Ax0$ states that non-λ events are "points", axiom $Ax1$ states that histories begin with the λ event, axiom $Ax2$ states that state interval are right closed, axiom $Ax3$ states that a change in state exists for some time, axiom $Ax4$ states that rigid variables don't change, axiom $Ax5$ states that a communication event doesn't change the shared variables, axiom $Ax6$ states that a *lam* event doesn't change the state, $Ax7a$ states that histories are dense, axiom $Ax7f$ is the induction axiom, axiom $Ax8$ states there is no last element, i.e, we have infinite histories, and axiom $Ax9$ states there is a first element, i.e., histories start at 0, $Ax10$ and $Ax11$ are the axioms for substitution and quantification. Axioms $F1$–$F7$ are the axioms of the future part of DTL and $P1$–$P7$ the past part. Axiom $F1$ ($P1$) states that $\widehat{\mathcal{U}}$ ($\widehat{\mathcal{S}}$) is monotonic in its second argument, axiom $F2$ ($P2$) states that $\widehat{\mathcal{U}}$ ($\widehat{\mathcal{S}}$) is monotonic in its first argument, axiom $F3$ ($P3$) states that reflection holds between past and future, and axiom $F7$ ($P7$) states that histories are linear.

As rules we take standard ones, i.e., the Modus Ponus, Generalisation, Specialisation, Instantiation and Universal Generalisation.

The following definition characterises a machine M in DTL. This kind of DTL formulae is history stutter insensitive.

Definition 18 (Machine in DTL). Given basis $B = ((\mathrm{In}, \mathrm{Out}), (\mathrm{V}, \mathrm{X}))$. Let In? be defined as $\{\mathbf{a}? \mid a \in \mathrm{In}\}$ and let Out! be defined as $\{\mathbf{a}! \mid a \in \mathrm{Out}\}$. Let I be a DTL formula over $\mathrm{V} \cup \mathrm{X}$ without the $\widehat{\mathcal{S}}$, $\widehat{\mathcal{U}}$ and $\exists$ operators. Let $\mathcal{T}$ be a finite set of DTL formulae τ of the form $(event_\tau \wedge trans_\tau)$ where $event_\tau$ is of the form $\epsilon = a_\tau$ where $a_\tau \in \{i, e\} \cup \mathrm{In}? \cup \mathrm{Out}!$, and $trans_\tau$ a DTL formula over $\mathrm{V} \cup \mathrm{X}$ and $\mathrm{V}' \cup \mathrm{X}'$ (variables primed with $'$) without the $\widehat{\mathcal{S}}$, $\widehat{\mathcal{U}}$ and $\exists$ operators such that $(\epsilon = e \Rightarrow \bigwedge_{\mathrm{x} \in \mathrm{X}} \mathrm{x}' = \mathrm{x})$, i.e., an environment action doesn't change

Table 3. Axioms. Let $n \in \mathfrak{R}$, $\mathbf{v} \in \mathfrak{V}$, $w \in \mathfrak{V} \cup \mathfrak{X}$, $\mathrm{x} \in \mathfrak{X}$ and $\epsilon \in \mathfrak{E}$.

$S0$: All the axioms for state formulae.

$Ax0: \ (\epsilon = \mathbf{a}? \vee \epsilon = \mathbf{a}! \vee \epsilon = i \vee \epsilon = e) \Rightarrow (\epsilon' = \lambda \wedge \grave{\epsilon} = \lambda)$

$Ax1: \ \mathbf{first} \rightarrow \epsilon = \lambda \wedge \mathbf{v}' = \mathbf{v} \wedge \mathrm{x}' = \mathrm{x}$ $\quad Ax2: \ \Box(\grave{\mathrm{x}} = \mathrm{x} \wedge \grave{\mathbf{v}} = \mathbf{v})$

$Ax3: \ \bigcirc(\mathrm{x}' = \mathrm{x} \wedge \mathbf{v}' = \mathbf{v}) \wedge (((\mathrm{x}' \neq \mathrm{x} \vee \mathbf{v}' \neq \mathbf{v}) \Rightarrow \bigcirc(\mathrm{x}'' = \mathrm{x}' \wedge \mathbf{v}'' = \mathbf{v}')))$

$Ax4: \ \Box(n = n' \wedge n = \grave{n})$ $\quad Ax5: \ (\epsilon = \mathbf{a}? \vee \epsilon = \mathbf{a}!) \Rightarrow \mathbf{v}' = \mathbf{v}$

$Ax6: \ \epsilon = \lambda \Rightarrow (\mathbf{v}' = \mathbf{v} \wedge \mathrm{x}' = \mathrm{x})$ $\quad Ax7a: \ \widehat{\Diamond} p \Rightarrow \widehat{\Diamond}\widehat{\Diamond} p$

$Ax7b: \ \neg\bigcirc p \Rightarrow \bigcirc\neg p$ $\quad Ax7c: \ \neg\ominus p \Rightarrow \ominus\neg p)$

$Ax7d: \ \bigcirc\ominus p \Rightarrow \bigcirc p$ $\quad Ax7e: \ \ominus\bigcirc p \Rightarrow \ominus p$

$Ax7f: \ (p \wedge p \Rightarrow \bigcirc p \wedge \ominus p \Rightarrow p) \rightarrow \Box p$

$Ax8: \ \Box\widehat{\Diamond}\mathbf{true}$ $\quad Ax9: \ \Box\widehat{\Diamond}\widehat{\diamondminus}\widehat{\boxminus}\mathbf{false}$

$Ax10: \ (exp_1 = exp_2) \Rightarrow (p\,[exp_1/w] \leftrightarrow p\,[exp_2/w])$
$\quad \wedge\, (rexp_1 = rexp_2) \Rightarrow (p\,[rexp_1/n] \leftrightarrow p\,[rexp_2/n])$
$\quad \wedge\, (evexp_1 = evexp_2) \Rightarrow (p\,[evexp_1/\epsilon] \leftrightarrow p\,[evexp_2/\epsilon])$

where p is a state formula and none of the variables appearing in respectively exp_1, exp_2, $rexp_1$, $rexp_2$, $evexp_1$ and $evexp_2$ are quantified in p.

$Ax11: \ (\forall \mathrm{x}.p) \Rightarrow p\,[exp/\mathrm{x}] \wedge (\forall n.p) \Rightarrow p\,[rexp/n] \wedge (\forall \epsilon.p) \Rightarrow p\,[evexp/\epsilon]$

where none of the variables appearing in exp, $rexp$ and $evexp$ are quantified in p.

$F1: \ \widehat{\Box}(p \rightarrow q) \Rightarrow (r\,\widehat{\mathcal{U}}\,p \rightarrow r\,\widehat{\mathcal{U}}\,q)$ $\quad F2: \ \widehat{\Box}(p \rightarrow q) \Rightarrow (p\,\widehat{\mathcal{U}}\,r \rightarrow q\,\widehat{\mathcal{U}}\,r)$

$F3: \ (p \wedge r\,\widehat{\mathcal{U}}\,q) \Rightarrow (r\,\widehat{\mathcal{U}}\,(q \wedge r\,\widehat{\mathcal{S}}\,p))$ $\quad F4: \ (q\,\widehat{\mathcal{U}}\,p \wedge \neg(r\,\widehat{\mathcal{U}}\,p)) \Rightarrow q\,\widehat{\mathcal{U}}\,(q \wedge \neg r)$

$F5: \ q\,\widehat{\mathcal{U}}\,p \Rightarrow (q \wedge q\,\widehat{\mathcal{U}}\,p)\,\widehat{\mathcal{U}}\,p$ $\quad F6: \ q\,\widehat{\mathcal{U}}\,(q \wedge q\,\widehat{\mathcal{U}}\,p) \Rightarrow q\,\widehat{\mathcal{U}}\,p$

$F7: \ (q\,\widehat{\mathcal{U}}\,p \wedge s\,\widehat{\mathcal{U}}\,r) \Rightarrow (q \wedge s)\,\widehat{\mathcal{U}}\,(p \wedge r) \vee (q \wedge s)\,\widehat{\mathcal{U}}\,(p \wedge s) \vee (q \wedge s)\,\widehat{\mathcal{U}}\,(q \wedge r)$

$P1: \ \widehat{\boxminus}(p \rightarrow q) \Rightarrow (r\,\widehat{\mathcal{S}}\,p \rightarrow r\,\widehat{\mathcal{S}}\,q)$ $\quad P2: \ \widehat{\boxminus}(p \rightarrow q) \Rightarrow (p\,\widehat{\mathcal{S}}\,r \rightarrow q\,\widehat{\mathcal{S}}\,r)$

$P3: \ (p \wedge r\,\widehat{\mathcal{S}}\,q) \Rightarrow (r\,\widehat{\mathcal{S}}\,(q \wedge r\,\widehat{\mathcal{U}}\,p))$ $\quad P4: \ (q\,\widehat{\mathcal{S}}\,p \wedge \neg(r\,\widehat{\mathcal{S}}\,p)) \Rightarrow q\,\widehat{\mathcal{S}}\,(q \wedge \neg r)$

$P5: \ q\,\widehat{\mathcal{S}}\,p \Rightarrow (q \wedge q\,\widehat{\mathcal{S}}\,p)\,\widehat{\mathcal{S}}\,p$ $\quad P6: \ q\,\widehat{\mathcal{S}}\,(q \wedge q\,\widehat{\mathcal{S}}\,p) \Rightarrow q\,\widehat{\mathcal{S}}\,p$

$P7: \ (q\,\widehat{\mathcal{S}}\,p \wedge s\,\widehat{\mathcal{S}}\,r) \Rightarrow (q \wedge s)\,\widehat{\mathcal{S}}\,(p \wedge r) \vee (q \wedge s)\,\widehat{\mathcal{S}}\,(p \wedge s) \vee (q \wedge s)\,\widehat{\mathcal{S}}\,(q \wedge r)$

Table 4. Rules. Let $w \in \mathfrak{R} \cup \mathfrak{X} \cup \mathfrak{E}$.

$MP: \ \dfrac{p, \ p \rightarrow q}{q}$ $\qquad SPEC: \ \dfrac{\Box p}{p}$ for state formula p

$GEN: \ \dfrac{p}{\widehat{\Box} p}$ for state formula p in which all occurrences of parameterised sentence symbols in p are rigid

$INST: \ \dfrac{p}{p\,[p_1/p_0]}$ where p_1 doesn't contain variables which are bound in p

$UGEN - w: \ \dfrac{p_0 \Rightarrow p_1}{p_0 \Rightarrow \forall w.p_1}$ for w not free in p_0

the local variables of the system. Define the stutter step, denoted by **stut**, as $\epsilon = \lambda \vee (\epsilon = i \wedge (\mathrm{V}, \mathrm{X})' = (\mathrm{V}, \mathrm{X})) \vee (\epsilon = e \wedge (\mathrm{V}, \mathrm{X})' = (\mathrm{V}, \mathrm{X}))$. Let T be the DTL formula $\mathbf{stut} \vee \bigvee_{\tau \in \mathcal{T}} \tau$. A machine in DTL is defined as $(B, \mathrm{I} \wedge \Box\mathrm{T})$.

Lemma 2. Given a machine in DTL $(B, \mathrm{I} \wedge \Box\mathrm{T})$ there exists a semantic machine $M = (B, I, T)$ such that $Comp(M) = Hist(\mathrm{I} \wedge \Box\mathrm{T})$.

The following example illustrates this notion.

Example 3. Machine M of example 1 as DTL-formula: $(B, \mathrm{I} \wedge \Box\mathrm{T})$ where basis $B \doteq ((\{a\}, \emptyset, (\{\mathbf{v}\}, \{\mathrm{u}\}))$, initial states formula $\mathrm{I} \doteq (\mathbf{v}, \mathrm{u}) = (0, 0)$ and transition formula $\mathrm{T} \doteq (\epsilon = \mathbf{a}? \wedge \mathrm{u} = 0 \wedge (\mathbf{v}, \mathrm{u})' = (\mathbf{v}, 1)) \vee (\epsilon = i \wedge (\mathbf{v}, \mathrm{u}) = (1, 1) \wedge (\mathbf{v}, \mathrm{u})' = (0, 2)) \vee (\epsilon = e \wedge (\mathbf{v}, \mathrm{u})' = (\mathbf{v} + 1, \mathrm{u})) \vee \mathbf{stut}$

Definition 19 (Machine specification of a system in DTL). Given a machine $(B, \mathrm{I} \wedge \Box\mathrm{T})$ in DTL. Let $\mathrm{WF} \subseteq \mathcal{T}$ be the set of weak fair transitions and $\mathrm{SF} \subseteq \mathcal{T}$ be the set of strong fair transitions. For $\tau \in \mathcal{T}$ define the enabledness condition for τ denoted $En(\tau)$ as $\exists \bar{v}_0.\tau\,[\bar{v}_0/\bar{v}']$ where $\tau\,[\bar{v}_0/\bar{v}']$ denotes the substitution of $\bar{v}_0$ (a list of variables not in $\mathrm{V} \cup \mathrm{X}$) for $\bar{v}'$ (the list of primed variables in τ). Let L be the DTL formula $\bigwedge_{\tau\in \mathrm{WF}}(\Diamond\Box En(\tau) \rightarrow \Box\Diamond\tau) \wedge \bigwedge_{\tau\in \mathrm{SF}}(\Box\Diamond En(\tau) \rightarrow \Box\Diamond\tau)$. The *machine specification* of a system in DTL is then a tuple $(B, \mathrm{I} \wedge \Box\mathrm{T} \wedge \mathrm{L})$.

Note: in above definition, liveness property L is such that $cl(Hist(\mathrm{I} \wedge \Box\mathrm{T}) \cap Hist(\mathrm{L})) = Hist(\mathrm{I} \wedge \Box\mathrm{T})$, i.e., it satisfies the machine closedness property. With this proviso the following lemma becomes straightforward.

Lemma 3. Given DTL machine specification $(B, \mathrm{I} \wedge \Box\mathrm{T} \wedge \mathrm{L})$ of a system, there exists a semantic machine specification $\mathcal{S} = (B, Comp(M) \cap L)$ such that $Comp(M) \cap L = Hist(\mathrm{I} \wedge \Box\mathrm{T} \wedge \mathrm{L})$.

3. Refinement and Composition of Reactive System Specifications

We introduce the notion of *refinement* and *composition* of reactive systems. Intuitively, refinement implies that the set of histories of a concrete system is a subset of the set of histories of the abstract system. Composition implies that the histories of the component systems are "merged" into composite histories, i.e., the histories of the composed system. Our merge operator is based on the merge operator of Aczel [Acz83]. Both are first defined at the semantic level and then for the DTL specifications.

3.1. Semantic Refinement and Composition of Specifications

Refinement and composition of reactive systems is defined at the semantical level. Refinement implies that the set of histories of a concrete system is a subset of the set of histories of an abstract system. Because histories also contain local information the subset relation doesn't correspond directly with refinement. The local information should first be projected away. The following definition captures this projection of local information.

Definition 20 (Observable system specification).
Given system specification $\mathcal{S} = (B, H)$ where $B = ((\mathrm{In}, \mathrm{Out}), (\mathrm{V}, \mathrm{X}))$. The *observable system specification* is a pair $(\mathfrak{O}(B), \mathcal{O}_{\mathrm{X}}(H))$ where $\mathfrak{O}(B)$ denotes the observable basis and is defined as $\mathfrak{O}(B) \doteq ((\mathrm{In}, \mathrm{Out}), \mathrm{V}, \emptyset)$ and $\mathcal{O}_{\mathrm{X}}(H)$ denotes the set of observable histories corresponding to H and is defined as $\{h \in \mathcal{H} \mid \exists h_1 \in H : h \text{ is an X-variant of } h_1\}$.

Definition 21 (Refinement of systems).
Given concrete system $\mathcal{S}_c \doteq (B_c, H_c)$ and abstract system $\mathcal{S}_a \doteq (B_a, H_a)$. $\mathcal{S}_c$ *refines* $\mathcal{S}_a$, denoted $\mathcal{S}_c$ **ref** $\mathcal{S}_a$, iff $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$ and $\mathcal{O}_{\mathrm{X}_c}(H_c) \subseteq \mathcal{O}_{\mathrm{X}_a}(H_a)$.

A more general definition of refinement would be one in which both the abstract and concrete system are composed of subsystems. Therefore the notion of *composition* is introduced. Intuitively the composition of two systems is that *matching* histories are merged into one history. The matching condition is illustrated in Figure 4, i.e., a history of one system matches a history of the other system if for all time points t the following two conditions hold: (1) the state information of the two histories at time t are same and also (2a) in both histories the λ-action occurs at time t *or* (2b) in both histories the environment action e occurs at time

$$\begin{tikzcd}
h_1 : & s_0 \arrow[r, "i"'] & s_1 \arrow[r, "e"'] & s_2 \arrow[r, "\mathbf{a}!"'] & s_3 \arrow[r, "\mathbf{c}?"'] & s_4 \\
h_2 : & s_0 \arrow[r, "e"'] & s_1 \arrow[r, "e"'] & s_2 \arrow[r, "\mathbf{a}?"'] & s_3 \arrow[r, "e"'] & s_4 \\
h_1 \text{ merged with } h_2 : & s_0 \arrow[r, "i"'] & s_1 \arrow[r, "e"'] & s_2 \arrow[r, "i"'] & s_3 \arrow[r, "\mathbf{c}?"'] & s_4
\end{tikzcd}$$

Fig. 4. Merging two matching histories.

t *or* (2c) in one history at time t a process action i occurs and in the other one an environment action e occurs at time t *or* (2d) in both histories at time t a communication action $\mathbf{a}$ occurs which is an input action in one of them and an output action in the other one, *or* (2e) in one history at time t a communication action occurs which is not an communication action in the other one and in the other history an environment action e occurs. So it is prohibited that the two components each perform an i action simultaneously because we intend to model interleaving, where only communication actions can occur simultaneously. Two matching histories are then merged into one history by (1) "copying" the state-information of the two histories; in case (2a) the resulting event becomes λ; in case (2b) the resulting event becomes e;in case (2c) the resulting event becomes i;in case (2d) the resulting event becomes i; and in case (2e) the resulting event becomes the communication action. See again Figure 4 for an illustration of this merging.

Definition 22 (Composition of two systems).
Given systems $\mathcal{S}_i = (B_i, H_i)$ with $B_i = ((\mathrm{In}_i, \mathrm{Out}_i), (\mathrm{V}_i, \mathrm{X}_i))$ $(i = 1, 2)$ such that $\mathrm{In}_1 \cap \mathrm{In}_2 = \emptyset$, $\mathrm{Out}_1 \cap \mathrm{Out}_2 = \emptyset$ and $\mathrm{X}_1 \cap \mathrm{X}_2 = \emptyset$. The *composed system* $\mathcal{S} = \mathcal{S}_1 \parallel \mathcal{S}_2$ is defined as (B, H) with $B \doteq ((\mathrm{In}_1 \setminus \mathrm{Out}_2 \cup \mathrm{In}_2 \setminus \mathrm{Out}_1, \mathrm{Out}_1 \setminus \mathrm{In}_2 \cup \mathrm{Out}_2 \setminus \mathrm{In}_1), (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{X}_1 \cup \mathrm{X}_2))$ and $H \doteq H_1 \bigotimes H_2$. The $\bigotimes$ symbol denotes a *merge* operator which merges the histories $h_1 \in H_1$ and $h_2 \in H_2$ into one history h and is defined as $H_1 \bigotimes H_2 \doteq \{h \in \mathcal{H} \mid \exists h_1 \in H_1, h_2 \in H_2 . \otimes(h, h_1, h_2)\}$ where for $h = \langle \psi, \theta \rangle$ and $h_j = \langle \psi_j, \theta_j \rangle$ $(j = 1, 2)$, $\otimes(h, h_1, h_2)$ iff

$$\begin{array}{lll}
1 & & \theta = \theta_1 \wedge \theta = \theta_2 \\
2 & \forall t : & \\
 & \vee & \psi(t)(\epsilon) = \lambda \wedge \psi_1(t)(\epsilon) = \lambda \wedge \psi_2(t)(\epsilon) = \lambda \\
 & \vee & \psi(t)(\epsilon) = e \wedge \psi_1(t)(\epsilon) = e \wedge \psi_2(t)(\epsilon) = e \\
 & \vee & \psi(t)(\epsilon) = i \wedge \psi_1(t)(\epsilon) = i \wedge \psi_2(t)(\epsilon) = e \\
 & \vee & \psi(t)(\epsilon) = i \wedge \psi_1(t)(\epsilon) = e \wedge \psi_2(t)(\epsilon) = i \\
 & \vee & \exists \mathbf{a} \in \mathrm{In}_1 \cap \mathrm{Out}_2 : \psi(t)(\epsilon) = i \wedge \psi_1(t)(\epsilon) = \mathbf{a}? \wedge \psi_2(t)(\epsilon) = \mathbf{a}! \\
 & \vee & \exists \mathbf{a} \in \mathrm{In}_2 \cap \mathrm{Out}_1 : \psi(t)(\epsilon) = i \wedge \psi_1(t)(\epsilon) = \mathbf{a}! \wedge \psi_2(t)(\epsilon) = \mathbf{a}? \\
 & \vee & \exists \mathbf{a} \in \mathrm{In}_1 \setminus \mathrm{Out}_2 : \psi(t)(\epsilon) = \mathbf{a}? \wedge \psi_1(t)(\epsilon) = \mathbf{a}? \wedge \psi_2(t)(\epsilon) = e \\
 & \vee & \exists \mathbf{a} \in \mathrm{Out}_1 \setminus \mathrm{In}_2 : \psi(t)(\epsilon) = \mathbf{a}! \wedge \psi_1(t)(\epsilon) = \mathbf{a}! \wedge \psi_2(t)(\epsilon) = e \\
 & \vee & \exists \mathbf{a} \in \mathrm{In}_2 \setminus \mathrm{Out}_1 : \psi(t)(\epsilon) = \mathbf{a}? \wedge \psi_1(t)(\epsilon) = e \wedge \psi_2(t)(\epsilon) = \mathbf{a}? \\
 & \vee & \exists \mathbf{a} \in \mathrm{Out}_2 \setminus \mathrm{In}_1 : \psi(t)(\epsilon) = \mathbf{a}! \wedge \psi_1(t)(\epsilon) = e \wedge \psi_2(t)(\epsilon) = \mathbf{a}!
\end{array}$$

The following Lemma expresses that the "making observable"-operation and the merge operator are both monotonic and that the "making observable"-operation on the composed system is equal to the "making observable"-operation on the components.

Lemma 4 (Properties of $\mathcal{O}$ and $\bigotimes$).
Given systems (B_1, H_0), (B_1, H_1), (B_2, H_2) and (B_2, H_3) then

$$\begin{array}{ll}
(a) & H_0 \subseteq H_1 \text{ implies } H_0 \bigotimes H_2 \subseteq H_1 \bigotimes H_2 \\
(b) & \mathcal{O}_{\mathrm{X}_{12}}(H_1 \bigotimes H_2) = \mathcal{O}_{\mathrm{X}_1}(H_1) \bigotimes \mathcal{O}_{\mathrm{X}_2}(H_2) \\
(c) & H_0 \subseteq H_1 \text{ implies } \mathcal{O}_{\mathrm{X}_1}(H_0) \subseteq \mathcal{O}_{\mathrm{X}_1}(H_1) \\
(d) & (H_0 \cap H_1) \bigotimes (H_2 \cap H_3) \subseteq (H_0 \bigotimes H_2) \cap (H_1 \bigotimes H_3)
\end{array}$$

Now the compositional refinement theorem can be inferred from above lemma 4 (a) and (b). Lemma 4(b) is actually the "heart" of the proof because it states that we have compositional semantics for reactive systems.

Theorem 2 (Compositional refinement).
Given abstract systems $\mathcal{S}_1 = (B_1, H_1)$ and $\mathcal{S}_2 = (B_2, H_2)$, and given concrete systems $\mathcal{S}_3 = (B_3, H_3)$ and $\mathcal{S}_4 = (B_4, H_4)$ such that $\mathfrak{O}(B_1) = \mathfrak{O}(B_3)$ and $\mathfrak{O}(B_2) = \mathfrak{O}(B_4)$ then

$$\mathcal{S}_3 \textbf{ ref } \mathcal{S}_1 \text{ and } \mathcal{S}_4 \textbf{ ref } \mathcal{S}_2 \text{ implies } \mathcal{S}_3 \parallel \mathcal{S}_4 \textbf{ ref } \mathcal{S}_1 \parallel \mathcal{S}_2.$$

3.2. Refinement and Composition of DTL Specifications

The refinement and composition notion of the previous section are translated into DTL by defining this notion for machine specifications (Def. 19). This means that first the *observable* machine specification should be defined in DTL.

Definition 23 (Observable machine specification in DTL).
Given machine specification $(B, \mathrm{I} \wedge \Box \mathrm{T} \wedge \mathrm{L})$ in DTL, the corresponding *observable* machine specification is defined as $(\mathfrak{O}(B), (\exists \mathrm{X} \,.\, (\mathrm{I} \wedge \Box \mathrm{T} \wedge \mathrm{L})))$.

The following lemma expresses that existential quantification relates to the semantic notion of observable histories.

Lemma 5. Given DTL machine specification $\mathcal{S} = (B, \mathrm{I} \wedge \Box \mathrm{T} \wedge \mathrm{L})$ then
$\mathcal{O}_{\mathrm{X}}(\mathit{Hist}(\mathrm{I} \wedge \Box \mathrm{T} \wedge \mathrm{L})) = \mathit{Hist}((\exists \mathrm{X} \,.\, (\mathrm{I} \wedge \Box \mathrm{T} \wedge \mathrm{L})))$

Theorem 3 (Refinement of machine specifications). Given concrete machine specification $\mathcal{S}_c \doteq (B_c, \mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)$ where $B_c \doteq (B_c^A, (\mathrm{V}_c, \mathrm{X}_c))$ and abstract machine specification $\mathcal{S}_a \doteq (B_a, \mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a)$ where $B_a \doteq (B_a^A, (\mathrm{V}_a, \mathrm{X}_a))$. Then $\mathcal{S}_c$ *refines* $\mathcal{S}_a$, denoted $\mathcal{S}_c$ **ref** $\mathcal{S}_a$, iff
$\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$ and $(\exists \mathrm{X}_c \,.\, (\mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)) \rightarrow (\exists \mathrm{X}_a \,.\, (\mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a))$.

Composition of DTL machine specifications can be defined in the same way as in the previous section.

Definition 24 (Composition of two DTL machine specifications).

Given DTL machine system specifications $\mathcal{S}_i \doteq (B_i, \mathrm{I}_i \wedge \Box \mathrm{T}_i \wedge \mathrm{L}_i)$ where $B_i \doteq (B_i^A, B_i^P)$, for $i = 1, 2$. Let ${}_{B_1^A}\odot_{B_2^A}(\epsilon, \epsilon_1, \epsilon_2)$ be defined as

$$\begin{array}{ll}\Box(& \epsilon = \lambda \wedge \epsilon_1 = \lambda \wedge \epsilon_2 = \lambda \\ \vee & \epsilon = e \wedge \epsilon_1 = e \wedge \epsilon_2 = e \\ \vee & \epsilon = i \wedge \epsilon_1 = i \wedge \epsilon_2 = e \\ \vee & \epsilon = i \wedge \epsilon_1 = e \wedge \epsilon_2 = i \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{In}_1 \cap \mathrm{Out}_2} \epsilon = i \wedge \epsilon_1 = \mathbf{a}? \wedge \epsilon_2 = \mathbf{a}! \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{In}_2 \cap \mathrm{Out}_1} \epsilon = i \wedge \epsilon_1 = \mathbf{a}! \wedge \epsilon_2 = \mathbf{a}? \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{In}_1 \setminus \mathrm{Out}_2} \epsilon = \mathbf{a}? \wedge \epsilon_1 = \mathbf{a}? \wedge \epsilon_2 = e \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{Out}_1 \setminus \mathrm{In}_2} \epsilon = \mathbf{a}! \wedge \epsilon_1 = \mathbf{a}! \wedge \epsilon_2 = e \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{In}_2 \setminus \mathrm{Out}_1} \epsilon = \mathbf{a}? \wedge \epsilon_1 = e \wedge \epsilon_2 = \mathbf{a}? \\ \vee & \bigvee_{\mathbf{a} \in \mathrm{Out}_2 \setminus \mathrm{In}_1} \epsilon = \mathbf{a}! \wedge \epsilon_1 = e \wedge \epsilon_2 = \mathbf{a}! \quad)\end{array}$$

Then the *composed* machine system specification $\mathcal{S}$ is defined as (B, H) where

$$\begin{array}{rcl}\mathrm{H} & \doteq & \exists \epsilon_1, \epsilon_2 . {}_{B_1^A}\odot_{B_2^A}(\epsilon, \epsilon_1, \epsilon_2) \wedge (\mathrm{I}_1 \wedge \Box \mathrm{T}_1 \wedge \mathrm{L}_1)\,[\epsilon_1/\epsilon] \wedge (\mathrm{I}_2 \wedge \Box \mathrm{T}_2 \wedge \mathrm{L}_2)\,[\epsilon_2/\epsilon] \\ B & \doteq & ((\mathrm{In}_1 \setminus \mathrm{Out}_2 \cup \mathrm{In}_2 \setminus \mathrm{Out}_1, \mathrm{Out}_1 \setminus \mathrm{In}_2 \cup \mathrm{Out}_2 \setminus \mathrm{In}_1), (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{X}_1 \cup \mathrm{X}_2)).\end{array}$$

This definition can be easily extended for n DTL specifications. One has only to define a predicate $\odot_{\bar{B}^A}(\epsilon, \bar{\epsilon})$ corresponding to the operation of merging n components.

Theorem 4 (Semantic merge is almost conjunction). Given the component machine system specifications $\mathcal{S}_1$ and $\mathcal{S}_2$, and the composed machine system specification $\mathcal{S}$ of definition 24 then $\mathit{Hist}(\mathrm{I}_1 \wedge \Box \mathrm{T}_1 \wedge \mathrm{L}_1) \bigotimes \mathit{Hist}(\mathrm{I}_2 \wedge \Box \mathrm{T}_2 \wedge \mathrm{L}_2) = \mathit{Hist}(\mathrm{H})$.

Example 4.
Given systems $\mathcal{S}_1 \doteq (B_1, \mathrm{I}_1 \wedge \Box \mathrm{T}_1 \wedge \mathrm{L}_1)$ and $\mathcal{S}_2 \doteq (B_2, \mathrm{I}_2 \wedge \Box \mathrm{T}_2 \wedge \mathrm{L}_2)$ where

$$\begin{array}{rl}B_1 \doteq & ((\{\mathbf{c}\}, \{\mathbf{a}\}), (\{\mathbf{s}\}, \{\mathrm{t}\})), \;\; \mathrm{I}_1 \doteq (\mathbf{s}, \mathrm{t}) = (0, 0), \;\; \mathrm{L}_1 \doteq \mathbf{true} \\ \mathrm{T}_1 \doteq & (\epsilon = \mathbf{a}! \wedge \mathrm{t} = 0 \wedge (\mathbf{s}, \mathrm{t})' = (\mathbf{s}, 1)) \vee (\epsilon = \mathbf{c}? \wedge \mathrm{t} = 1 \wedge (\mathbf{s}, \mathrm{t})' = (\mathbf{s}, 2)) \vee \\ & (\epsilon = e \wedge (\mathbf{s}, \mathrm{t})' = (1, \mathrm{t})) \vee (\epsilon = e \wedge (\mathbf{s}, \mathrm{t})' = (0, \mathrm{t})) \vee \mathbf{stut}_1 \\ B_2 \doteq & ((\{\mathbf{b}\}, \{\mathbf{c}\}), (\{\mathbf{s}\}, \{\mathrm{u}\})), \;\; \mathrm{I}_2 \doteq (\mathbf{s}, \mathrm{u}) = (0, 0), \;\; \mathrm{L}_2 \doteq \mathbf{true} \\ \mathrm{T}_2 \doteq & (\epsilon = \mathbf{c}! \wedge \mathrm{u} = 0 \wedge (\mathbf{s}, \mathrm{u})' = (\mathbf{s}, 1)) \vee (\epsilon = \mathbf{b}? \wedge \mathrm{u} = 1 \wedge (\mathbf{s}, \mathrm{u})' = (\mathbf{s}, 2)) \vee \\ & (\epsilon = i \wedge (\mathbf{s}, \mathrm{u}) = (1, 2) \wedge (\mathbf{s}, \mathrm{u})' = (0, \mathrm{u})) \vee (\epsilon = e \wedge (\mathbf{s}, \mathrm{u})' = (1, \mathrm{u})) \vee \mathbf{stut}_2\end{array}$$

The transitions of machine 1 and 2 are illustrated in figure 5. Note: the stutter transitions are not drawn in all subsequent figures in order to minimise the number of edges.

According to definition 24 the composition of $\mathcal{S}_{c_1}$ and $\mathcal{S}_{c_2}$ is as follows:

$$\begin{array}{l}(\;((\{\mathbf{a}\}, \{\mathbf{b}\}), (\{\mathbf{s}\}, \{\mathrm{t}, \mathrm{u}\})), \\ \quad \exists \epsilon_1, \epsilon_2 . ({}_{B_1^A}\odot_{B_2^A}(\epsilon, \epsilon_1, \epsilon_2) \wedge (\mathrm{I}_1 \wedge \Box \mathrm{T}_1 \wedge \mathrm{L}_1)\,[\epsilon_1/\epsilon] \wedge (\mathrm{I}_2 \wedge \Box \mathrm{T}_2 \wedge \mathrm{L}_2)\,[\epsilon_2/\epsilon])\;)\end{array}$$

4. Proving Refinement of Reactive System Specifications

This section explains how refinement of reactive systems can be proved. The standard technique of Abadi and Lamport [AL91a] is used, i.e., refinement is proven by providing a refinement mapping from the concrete system to the abstract system. First we give its definition at the semantic level and then for DTL specifications.

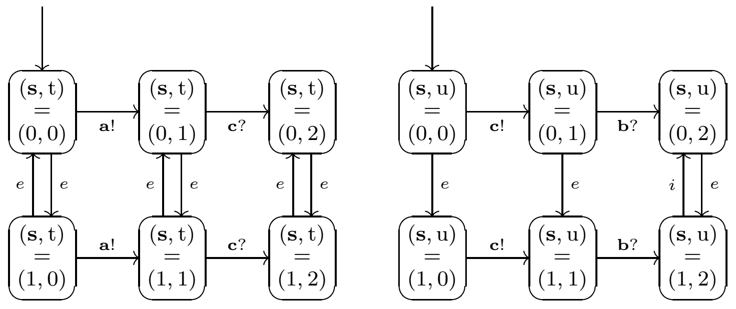

Fig. 5. Transitions of $\mathcal{S}_1$ and $\mathcal{S}_2$.

4.1. Proving Semantic Refinement of Specifications

Refinement of reactive systems is proved by means of a *refinement mapping* from the concrete system to the abstract system. A refinement mapping maps a history at the concrete level to a history at the abstract level, more specifically, it maps the states appearing in the concrete history to states appearing in the abstract history.

Definition 25 (Systems refinement mapping).
Given concrete system $\mathcal{S}_c \doteq (B_c, H_c)$ and abstract system $\mathcal{S}_a \doteq (B_a, H_a)$ s.t. $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$. A *refinement mapping* from $\mathcal{S}_c$ to $\mathcal{S}_a$ is a mapping f from states appearing in histories of H_c to states appearing in histories of H_a, i.e., $f : \Sigma \to \Sigma$ s.t. (1) the values of observable variables are not changed, i.e., for all $\sigma \in \Sigma$: $\sigma|^1_{\mathrm{V}_c} = f(\sigma)|^1_{\mathrm{V}_c}$, and (2) for all $h_c \in H_c$ there exists a $h_a \in H_a$ s.t. for all $t \in \mathbb{R}^+$, $\psi_c(t) = \psi_a(t)$ and $\theta_a(t) = f(\theta_c(t))$.

Lemma 6. Given concrete system $\mathcal{S}_c \doteq (B_c, H_c)$ and abstract system $\mathcal{S}_a \doteq (B_a, H_a)$ s.t. $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$. If there exists a refinement mapping from $\mathcal{S}_c$ to $\mathcal{S}_a$, then $\mathcal{S}_c$ **ref** $\mathcal{S}_a$.

Next the concept of refinement mappings is applied to machine specifications. Refinement means then that $f(Comp(M_c) \cap L_c) \subseteq Comp(M_a) \cap L_a$ for refinement mapping f. This can be split into (1) $f(Comp(M_c) \cap L_c) \subseteq Comp(M_a)$ and (2) $f(Comp(M_c) \cap L_c) \subseteq L_a$. From $f(Comp(M_c)) \subseteq Comp(M_a)$ follows (1) because $f(Comp(M_c) \cap L_c) \subseteq f(Comp(M_c))$. So the verification condition can be split into a condition on machines and a condition involving machines and supplementary conditions. This leads to the following definition.

Definition 26 (Machine specifications refinement mapping).
Given abstract machine specification $\mathcal{S}_a \doteq (B_a, Comp(M_a) \cap L_a)$, where $M_a \doteq (B_a, I_a, T_a)$, and concrete machine specification $\mathcal{S}_c \doteq (B_c, Comp(M_c) \cap L_c)$, where $M_c \doteq (B_c, I_c, T_c)$.
A *refinement mapping* from $\mathcal{S}_c$ to $\mathcal{S}_a$ is a mapping $f : \Sigma \to \Sigma$ s.t.
(1) $\forall \sigma \in \Sigma \,.\, \sigma|^1_{\mathrm{V}_c} = f(\sigma)|^1_{\mathrm{V}_c}$,
(2a) $\forall \sigma_c \in I_c \,.\, \exists \sigma_a \in I_a \,.\, \sigma_a = f(\sigma_c)$,
(2b) $\forall \langle d, \sigma_{c1}, \sigma_{c2} \rangle \in T_c \,.\, \langle d, f(\sigma_{c1}), f(\sigma_{c2}) \rangle \in T_a \vee$
$\qquad (f(\sigma_{c1}) = f(\sigma_{c2}) \wedge d(\epsilon) \in \{\lambda, i, e\}$,
(2c) $\forall h_c \in Comp(M_c) \cap L_c \,.$
$\qquad \exists h_a \in L_a \,.\, \forall t \in \mathbb{R}^+ \,.\, \psi_c(t) = \psi_a(t) \wedge f(\theta_c(t)) = \theta_a(t)$.

That refinement mappings are indeed sound for proving refinement of machine specifications, is stated in:

Lemma 7. Given concrete machine specification $\mathcal{S}_c \doteq (B_c, Comp(M_c) \cap L_c)$ and abstract machine specification $\mathcal{S}_a \doteq (B_a, Comp(M_a) \cap L_a)$ s.t. $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$. If there exists a refinement mapping from $\mathcal{S}_c$ to $\mathcal{S}_a$ then $\mathcal{S}_c$ **ref** $\mathcal{S}_a$.

4.2. Proving Refinement of DTL Specifications

Proving refinement of machine specifications in DTL amounts according to Theorem 3 to (1) the observable bases are equal and (2) that a certain formula with two existential quantifications is valid. More specifically:

Definition 27 (Refinement of DTL machine specifications). Given concrete machine specification $\mathcal{S}_c \doteq (B_c, \mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)$ and abstract machine specification $\mathcal{S}_a \doteq (B_a, \mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a)$. Then $\mathcal{S}_c$ *refines* $\mathcal{S}_a$, denoted $\mathcal{S}_c$ **ref** $\mathcal{S}_a$, is defined by $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$ and
$(\exists \mathrm{X}_c \,.\, (\mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)) \rightarrow (\exists \mathrm{X}_a \,.\, (\mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a))$.

So we must have a rule to prove the following implication: $\exists \mathrm{x}_0 . p_0 \rightarrow \exists \mathrm{x}_1 . p_1$. The following rule, derived from the proof system of table 3 and 4, does the job:

$$\frac{p_0 \rightarrow p_1\,[exp/\mathrm{x}_1]}{\exists \mathrm{x}_0 . p_0 \rightarrow \exists \mathrm{x}_1 . p_1} \quad \begin{array}{l}\text{for } \mathrm{x}_0 \text{ not free in } p_1 \text{ and none of the variables}\\ \text{appearing in } exp \text{ is quantified in } p_1\end{array}$$

From the previous section it should be clear that *exp* expresses exactly the refinement mapping f, and that the proof can be split in a safety part and a liveness part (i.e., the proof of $p_0 \rightarrow p_1\,[exp/\mathrm{x}_1]$ of above rule is split into a safety and a liveness part). This culminates in the following proof rule for refinement based on similar ones in [Lam94, KMP93].

Rule 1 (Proof rule for refinement).
Given concrete machine specification $\mathcal{S}_c \doteq (B_c, \mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)$ and abstract machine specification $\mathcal{S}_a \doteq (B_a, \mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a)$ s.t. $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$. Let f be a refinement mapping from $\mathcal{S}_c$ to $\mathcal{S}_a$ then

$$\frac{\mathcal{S}_c \models \mathrm{I}_c \rightarrow \mathrm{I}_a\,[f/\mathrm{X}_a]\,,\;\; \mathcal{S}_c \models \mathrm{T}_c \rightarrow \mathrm{T}_a\,[f/\mathrm{X}_a]\,,\;\; \mathcal{S}_c \models \mathrm{L}_a\,[f/\mathrm{X}_a]}{\models (\exists \mathrm{X}_c \,.\, (\mathrm{I}_c \wedge \Box \mathrm{T}_c \wedge \mathrm{L}_c)) \rightarrow (\exists \mathrm{X}_a \,.\, (\mathrm{I}_a \wedge \Box \mathrm{T}_a \wedge \mathrm{L}_a))}$$

Rule 1 and Theorem 2 are used in the following example.

Example 5.
As abstract machine specification we take the composition of the machine specifications $\mathcal{S}_1$ and $\mathcal{S}_2$ of example 4. As concrete machine specification we take the composition of following machine specifications $\mathcal{S}_3 \doteq (B_3, \mathrm{I}_3 \wedge \Box \mathrm{T}_3 \wedge \mathrm{L}_3)$ and again $\mathcal{S}_2$ where

$$\begin{array}{rl} B_3 \doteq & ((\{\mathbf{c}\}, \{\mathbf{a}\}), (\{\mathbf{s}\}, \{\mathrm{v}\})),\;\; \mathrm{I}_3 \doteq (\mathbf{s}, \mathrm{v}) = (0, 0),\;\; \mathrm{L}_3 \doteq \mathbf{true} \\ \mathrm{T}_3 \doteq & (\epsilon = \mathbf{a}! \wedge \mathrm{v} = 0 \wedge (\mathbf{s}, \mathrm{v})' = (\mathbf{s}, 1)) \vee (\epsilon = i \wedge \mathrm{v} = 1 \wedge (\mathbf{s}, \mathrm{v})' = (\mathbf{s}, 2)) \vee \\ & (\epsilon = \mathbf{c}? \wedge \mathrm{v} = 2 \wedge (\mathbf{s}, \mathrm{v})' = (\mathbf{s}, 3)) \vee (\epsilon = e \wedge (\mathbf{s}, \mathrm{v})' = (1, \mathrm{v})) \vee \\ & (\epsilon = e \wedge (\mathbf{s}, \mathrm{v})' = (0, \mathrm{v})) \vee \mathbf{stut}_3 \end{array}$$

So $\mathcal{S}_3$ differs from $\mathcal{S}_1$ in that it has an extra internal step before the $\mathbf{c}?$ communication.

We want to prove $\mathcal{S}_3 \parallel \mathcal{S}_2$ **ref** $\mathcal{S}_1 \parallel \mathcal{S}_2$. Instead of constructing the two

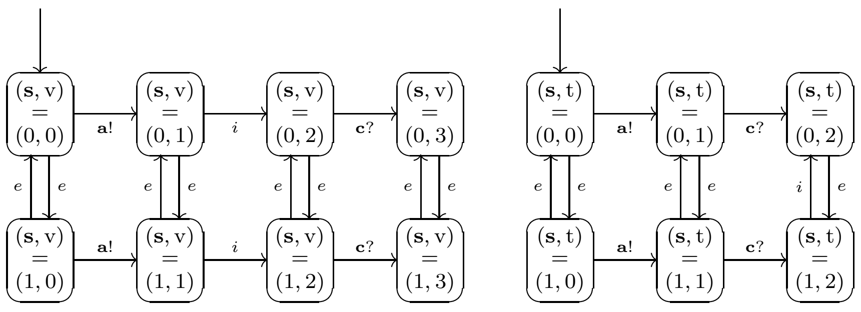

Fig. 6. Transitions of concrete $\mathcal{S}_3$ and abstract $\mathcal{S}_1$.

compositions and then applying Rule 1, we will first use Theorem 2 and then Rule 1. That is, we only have to prove that (1) $\mathcal{S}_3$ **ref** $\mathcal{S}_1$ and (2) $\mathcal{S}_2$ **ref** $\mathcal{S}_2$. Case (2) is trivial, so we will show the proof of (1). The condition that the observable bases are equal is trivial. So we must prove

$$(\exists X_3 \,.\, (I_3 \wedge \Box T_3 \wedge L_3)) \rightarrow (\exists X_1 \,.\, (I_1 \wedge \Box T_1 \wedge L_1))$$

This will be proven with Rule 1. This means one has to find a refinement mapping f. In order to find such a mapping the transitions of $\mathcal{S}_3$ and $\mathcal{S}_1$ are illustrated in fig. 6.

From fig. 6 one sees that f should be defined as follows:

$$\begin{array}{llll} \textit{if} & \mathrm{v}=0 \vee \mathrm{v}=1 & \textit{then} & f(\mathbf{s},\mathrm{v})=\mathrm{v} \\ & \mathrm{v}=2 \vee \mathrm{v}=3 & \textit{then} & f(\mathbf{s},\mathrm{v})=\mathrm{v}-1 \quad \textit{fi} \end{array}$$

The following premises should be valid in order to apply the rule:

- $\mathcal{S}_3 \models (\mathbf{s},\mathrm{v}) = (0,0) \rightarrow ((\mathbf{s},\mathrm{t}) = (0,0))\,[f/\mathrm{t}]$. Applying substitution results in $(\mathbf{s},\mathrm{v}) = (0,0) \rightarrow (\mathbf{s},\mathrm{v}) = (0,0)$ and this is valid.
- $\mathcal{S}_3 \models (\epsilon = \mathbf{a}! \wedge \mathrm{v} = 0 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},1)) \rightarrow (\epsilon = \mathbf{a}! \wedge \mathrm{t} = 0 \wedge (\mathbf{s},\mathrm{t})' = (\mathbf{s},1))\,[f/\mathrm{t}]$. Applying substitution results in: $\mathcal{S}_3 \models (\epsilon = \mathbf{a}! \wedge \mathrm{v} = 0 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},1)) \rightarrow (\epsilon = \mathbf{a}! \wedge \mathrm{v} = 0 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},1))$ and this is valid.
- $\mathcal{S}_3 \models (\epsilon = i \wedge \mathrm{v} = 1 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},2)) \rightarrow \mathbf{stut}_1\,[f/\mathrm{t}]$. Because of definition of $\mathbf{stut}_1$ it suffices to prove: $\mathcal{S}_3 \models (\epsilon = i \wedge \mathrm{v} = 1 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},2)) \rightarrow (\epsilon = i \wedge (\mathbf{s},\mathrm{t})' = (\mathbf{s},\mathrm{t}))\,[f/\mathrm{t}]$. Applying substitution results in: $\mathcal{S}_c \models (\epsilon = i \wedge \mathrm{v} = 1 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},2)) \rightarrow (\epsilon = i \wedge (\mathbf{s},\mathrm{v}-1)' = (\mathbf{s},\mathrm{v}))$ and this is valid.
- $\mathcal{S}_3 \models (\epsilon = \mathbf{c}? \wedge \mathrm{v} = 2 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},3)) \rightarrow (\epsilon = \mathbf{c}? \wedge \mathrm{t} = 1 \wedge (\mathbf{s},\mathrm{t})' = (\mathbf{s},2))\,[f/\mathrm{t}]$. Applying substitution results in: $\mathcal{S}_3 \models (\epsilon = \mathbf{c}? \wedge \mathrm{v} = 2 \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},3)) \rightarrow (\epsilon = \mathbf{c}? \wedge \mathrm{v} - 1 = 1 \wedge (\mathbf{s},\mathrm{v}-1)' = (\mathbf{s},2))$ and this is valid.
- $\mathcal{S}_3 \models (\epsilon = e \wedge (\mathbf{s},\mathrm{v})' = (1,\mathrm{v})) \rightarrow (\epsilon = e \wedge (\mathbf{s},\mathrm{t})' = (1,\mathrm{t}))\,[f/\mathrm{t}]$. Applying substitution results in: $\mathcal{S}_3 \models (\epsilon = e \wedge (\mathbf{s},\mathrm{v})' = (1,\mathrm{v})) \rightarrow (\epsilon = e \wedge (\mathbf{s},f)' = (1,f))$ and this is valid because of $\mathrm{v}' = \mathrm{v} \Rightarrow f' = f$.
- $\mathcal{S}_3 \models (\epsilon = e \wedge (\mathbf{s},\mathrm{v})' = (0,\mathrm{v})) \rightarrow (\epsilon = e \wedge (\mathbf{s},\mathrm{t})' = (0,\mathrm{t}))\,[f/\mathrm{t}]$. Applying substitution results in: $\mathcal{S}_3 \models (\epsilon = e \wedge (\mathbf{s},\mathrm{v})' = (0,\mathrm{v})) \rightarrow (\epsilon = e \wedge (\mathbf{s},f)' = (0,f))$ and this is valid because of $\mathrm{v}' = \mathrm{v} \Rightarrow f' = f$.
- $\mathcal{S}_3 \models \mathbf{stut}_3 \rightarrow \mathbf{stut}_1\,[f/\mathrm{t}]$. Definition of $\mathbf{stut}_3$ and $\mathbf{stut}_1$ results in $\mathcal{S}_2 \models \epsilon = \lambda \vee (\epsilon = i \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},\mathrm{v})) \vee (\epsilon = e \wedge (\mathbf{s},\mathrm{v})' = (\mathbf{s},\mathrm{v})) \rightarrow \epsilon = \lambda \vee (\epsilon = i \wedge (\mathbf{s},\mathrm{t})' = (\mathbf{s},\mathrm{t}))\,[f/\mathrm{t}] \vee (\epsilon = e \wedge (\mathbf{s},\mathrm{t})' = (\mathbf{s},\mathrm{t}))\,[f/\mathrm{t}]$ and this is valid.

- $\mathcal{S}_c \models \textbf{true}\,[f/\mathrm{t}]$. This is valid.

5. Conclusion

It is shown that a dense temporal logic can be devised in which you can express compositionality and refinement of reactive systems. Furthermore a compositional refinement rule can be derived which enables ones to prove refinement of large reactive systems from proofs of refinement between their components. In [Cau95] this rule is extensively used to prove the correctness of an implementation of stable storage and the correctness of Dijkstra's solution to the readers/writers synchronisation problem. A proof sketch of the stable storage example is given in the appendix.

Acknowledgements

This work would not be finished without the help of Willem-Paul de Roever, who infected me with his compositionality and refinement dogma and whose comments were priceless, Pierre Collette, Ruurd Kuiper, Yassine Lakhneche, Qiwen Xu and Kai Engelhardt, who all made many helpful comments.

References

[Acz83] P. Aczel. On an Inference Rule for Parallel Composition, 1983. Unpublished, University of Manchester.

[AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFK88] K. R. Apt, N. Francez, and S. Katz. Appraising Fairness in Languages for Distributed Programming. *Distributed Computing*, 2(4):226–241, 1988.

[AL91a] M. Abadi and L. Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[AL91b] M. Abadi and L. Lamport. An Old-Fashioned Recipe for Real Time. In J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *LNCS 600, Real-Time: Theory in Practice*, pages 1–27. Springer-Verlag, 1991.

[AL93] M. Abadi and L. Lamport. Composing Specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[AS85] B. Alpern and F.B. Schneider. Defining Liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[AS87] B. Alpern and F. Schneider. Proving Boolean Combinations of Deterministic Properties. In *Proceedings of the second symposium on logic in computer science*, pages 131–137. IEEE, June 1987.

[BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you May Compose Temporal Logic Specifications. In *Proceedings of 16th ACM Symposium on Theory of Computing*, pages 51–63, 1984.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and its Temporal Semantics. In *Proc. 13th ACM Symp. Princ. of Prog. Lang.*, pages 173–183, 1986.

[Bur82] J.P. Burgess. Axioms for Tense Logic, I. "Since" and "Until". *Notre Dame Journal of Formal Logic*, 23(4):367–374, October 1982.

[Bur84] J.P. Burgess. Basic Tense Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 89–133. Reidel Publishers, 1984.

[Cau95] A. Cau. *Compositional Verification and Specification of Refinement for Reactive Systems in Dense Time Temporal Logic.* PhD thesis, Christian-Albrechts-Universität zu Kiel, 1995. Available as report no. 9601.

[CC96] A. Cau and P. Collette. Parallel Composition of Assumption-Commitment Specifications: a unifying approach for shared variable and distributed message passing concurrency. *Acta Informatica*, 33(2):153–176, 1996.

[Dil88] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1988.

[DK90] E. Diepstraten and R. Kuiper. Abadi & Lamport and Stark: Towards a Proof Theory for stuttering, Dense Domains and Refinements Mappings. In *LNCS 430: Proc. of the REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*, pages 208–238. Springer-Verlag, 1990.

[FM92] J.L. Fiadeiro and T.S. Maibaum. Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects of Computing*, 4(3):239–272, 1992.

[FM94] J.L. Fiadeiro and T.S. Maibaum. Sometimes “Tomorrow” is “Sometime”. In D. Gabbay and H. Ohlbach, editors, *LNAI 827: Proc. of 1st International Conference on Temporal Logic*, pages 48–66, 1994.

[GSSAL94] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in Timed and Untimed Systems. In *Proc. of ICALP 21*, 1994.

[Hoa84] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, London, 1984.

[Jon83] C.B. Jones. Tentative Steps Towards a Development Method for Interfering Programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[KMP93] Y. Kesten, Z. Manna, and A. Pnueli. Temporal Verification of Simulation and Refinement. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *LNCS 803: A Decade of Concurrency, Reflections and Perspectives*, pages 273–346. Springer-Verlag, 1993.

[Lam83] L. Lamport. What Good is Temporal Logic. In R.E.A. Manson, editor, *Information Processing 83: Proc. of the IFIP 9th World Congress*, pages 657–668. IFIP, Elsevier Science Publishers, North Holland, 1983.

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM TOPLAS*, 16(3):872–923, May 1994.

[LV95] N.A. Lynch and F. Vaandraager. Forward and Backward Simulations – Part II: Timing-Based Systems. *Information and Computation*, 121(2):214–233, 1995.

[MC81] J. Misra and K.M. Chandy. Proofs of Networks of Processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, July 1981.

[Sta84] E.W. Stark. *Foundations of a Theory of Specification for Distributed Systems*. PhD thesis, Massachusetts Inst. of Technology, 1984. Available as Report No. MIT/LCS/TR-342.

[ZCdR92] J. Zwiers, J. Coenen, and W.-P. de Roever. A Note on Compositional Refinement. In C. B. Jones, R. C. Shaw, and T. Denvir, editors, *5th Refinement Workshop*, Workshops in Computing, pages 342–366, London, January 1992. BCS-FACS, Springer Verlag.

A. Stable Storage Example

We will sketch the proof of the correctness of a fault tolerant system consisting of a number of disks implementing stable storage. This stable storage stores and retrieves data in a reliable way under the assumption that faults from a certain class are recovered (corrected). This class consists of two kinds of faults. The first one consists of faults that damage the disk surface, the contents of the disk are said to be corrupted by these faults. The second one consists of faults that affect the disk control system, and results in the contents of the disk being read from or written to the wrong location. Notice that other kinds of faults, such as power failure or physical destruction of the whole stable storage system, are not taken into account. I.e., stable storage should function correctly provided such latter faults do not occur. Before we give the proof sketch we introduce the notion of *relative refinement* and *relative composition* of systems.

Ordinary refinement stipulates that the set of histories generated by the concrete system is included in the set of histories generated by the abstract system. Relative refinement means that this inclusion almost holds, i.e., if one leaves some of the histories generated at the concrete level out of account this inclusion holds. Histories generated by the abstract system can also be left out because a concrete system could be an abstract system in a next refinement step. Ordinary composition means that the histories of two components are merged into the histories of the composed system. Relative composition means that one leaves certain histories out of this merge, i.e., the merge is performed on subsets of histories of the components.

Definition 28 (Relative refinement of systems). Given concrete system $\mathcal{S}_c \doteq (B_c, H_c)$ and a set W_c of allowed histories for $\mathcal{S}_c$ ($W_c \subseteq \mathcal{H}$ constraining B_c) and abstract system $\mathcal{S}_a \doteq (B_a, H_a)$ together with a set W_a of allowed histories for $\mathcal{S}_a$ ($W_a \subseteq \mathcal{H}$ constraining B_a). Let $G_c \doteq H_c \cap W_c$ and $G_a \doteq H_a \cap W_a$. Then $\mathcal{S}_c$ *relatively refines* $\mathcal{S}_a$ with respect to (W_c, W_a), denoted by $\mathcal{S}_c \;{}_{W_c}\mathbf{ref}^{\,W_a}\, \mathcal{S}_a$, iff $\mathfrak{O}(B_c) = \mathfrak{O}(B_a)$ and $\mathcal{O}_{\mathrm{X}_c}(G_c) \subseteq \mathcal{O}_{\mathrm{X}_a}(G_a)$.

Definition 29 (Relative composition of systems). Given systems $\mathcal{S}_i = (B_i, H_i)$ where $B_i = ((\mathrm{In}_i, \mathrm{Out}_i), (\mathrm{V}_i, \mathrm{X}_i)), i = 1, 2$ such that $\mathrm{X}_1 \cap \mathrm{X}_2 = \emptyset$ and given sets $W_i \subseteq \mathcal{H}$ constraining B_i. Let $\overline{W}$ denote (W_1, W_2). Then the *relative composed system* $\mathcal{S}$ with respect to $\overline{W}$, denoted $\mathcal{S}_1 \mid_{\overline{W}} \mid \mathcal{S}_2$, is defined as (B, H) with $B \doteq ((\mathrm{In}_1 \setminus \mathrm{Out}_2 \cup \mathrm{In}_2 \setminus \mathrm{Out}_1, \mathrm{Out}_1 \setminus \mathrm{In}_2 \cup \mathrm{Out}_2 \setminus \mathrm{In}_1), (\mathrm{V}_1 \cup \mathrm{V}_2, \mathrm{X}_1 \cup \mathrm{X}_2))$, and $H \doteq H_1 \textcircled{\scriptsize $\overline{W}$} H_2 \doteq (H_1 \cap W_1) \bigotimes (H_2 \cap W_2)$.

The proof rule 1 for refinement can be changed accordingly to deal with relative refinement also compositional relative refinement can be defined see [Cau95].

The proof consists of the following 4 steps.

1. In this step we give the abstract specification $\mathcal{S} \doteq (B, \mathrm{H})$ where H is a DTL formula specifying the stable storage. In this specification no faults are visible, hence they don't occur as observables. The designer's task is to give an implementation of this system under the assumption that only faults from above class can occur. In step 2, 3 and 4 this implementation is developed.
2. This step *identifies* how the faults will affect an implementation $\mathcal{S}_P \doteq (B_P, \mathrm{H}_P)$. This implementation serves as first approximation to the final implementation of $\mathcal{S}$. It should be clear that $\mathcal{S}_P$ is not a refinement of $\mathcal{S}$ because of the possible occurrences of these faults. $\mathcal{S}_P$ is only a refinement when these faults do not occur, i.e., $\mathcal{S}_P$ is a *relative refinement* of $\mathcal{S}$. Let W_P express that these faults never occur then we must prove the following

   (1) $\mathcal{S}_P \;{}_{W_P}\mathbf{ref}\; \mathcal{S}$

3. In this step one specifies *how these faults are detected*, i.e., one has to specify a detection layer $\mathcal{S}_{Ds}$ for these faults. This layer is added in bottom-up fashion to the implementation $\mathcal{S}_P$ of the second step and stops upon detection of the first error, i.e., $\mathcal{S}_{Ds}$ is a *fail-stop* implementation. So the second approximation to the final implementation consists of the composition of $\mathcal{S}_P$ and $\mathcal{S}_{Ds}$. This approximation is clearly not a refinement because when in $\mathcal{S}_P$ a fault occurs, and $\mathcal{S}_{Ds}$ detects the corresponding error, the whole approximation stops. One would like to have (eventually) an approximation that doesn't stop, i.e., the physical disk isn't affected by faults and the detection layer should detect no

error. Let $\overline{W} \doteq (W_{Ds}, W_P)$ where W_P expresses that no faults occur and W_{Ds} expresses that no error is detected. Then we must prove the following:

$$(2) \quad \mathcal{S}_{Ds} \,|_{\overline{W}}|\, \mathcal{S}_P \ \mathbf{ref}^{\ W_P}\, \mathcal{S}_P.$$

From (1), (2) and the transitivity of relative refinement relation follows:

$$\mathcal{S}_{Ds} \,|_{\overline{W}}|\, \mathcal{S}_P \ \mathbf{ref}\ \ \mathcal{S}.$$

4. In the fourth step one *specifies the corrective action to be undertaken after detection of an error*. This means in general that one needs *redundancy*, i.e., several copies of $\mathcal{S}_P$ and $\mathcal{S}_D$ components, because when a detection layer $\mathcal{S}_D$ detects an error, the state before that error has to be recovered and that can only be done by accessing another copy of $\mathcal{S}_P$ through its corresponding detection layer $\mathcal{S}_D$. Note that the $\mathcal{S}_D$ component doesn't stop anymore on the detection of an error but merely waits for the corrective action to be undertaken. Say, we need N copies of $\mathcal{S}_P$ and $\mathcal{S}_D$. The final implementation consists then of those N copies of $\mathcal{S}_P$ and $\mathcal{S}_D$ plus a recovery layer $\mathcal{S}_R$. Let W_R express which kind of errors can be recovered. If the following holds:

$$(3) \quad \|_{i=1}^{N} (\mathcal{S}_{P_i} \parallel \mathcal{S}_{D_i}) \parallel \mathcal{S}_R \ {}_{W_R}\mathbf{ref}\ \ \mathcal{S}_{Ds} \,|_{\overline{W}}|\, \mathcal{S}_P$$

then from (1), (2), (3) and the transitivity of relative refinement follows the desired result, i.e.:

$$\|_{i=1}^{N} (\mathcal{S}_{P_i} \parallel \mathcal{S}_{D_i}) \parallel \mathcal{S}_R \ {}_{W_R}\mathbf{ref}\ \ \mathcal{S}$$