

Linear State Machines Formal Model

Legalese.com

October 17, 2017

Click most terms (in [this color](#)) to jump to their first underlined usage.

Contents

1	Basics: events, time, and traces	1
1.1	Execution for simple contracts	4
2	Infinite state: Global Variables and Event Parameters	5
2.1	Execution	7
3	Event Schema	7
4	May-Later and Must-Later	7

1 Basics: events, time, and traces

This section defines a complete-but-limited model of contracts, called simple contracts, and also gives definitions that will be used for the full definition of contracts in Section 2.

Every [contract](#) specifies a time unit; it is the smallest unit of time that one writes constraints about or does arithmetic with. We expect it will most often be days. A time stamp is a natural number that we think of as being in units [time unit](#).

An event is composed of an action, a role, a [time stamp](#), and optionally some parameters (but parameters will not be introduced until Section 2). The [actions](#) and [roles](#) are fixed finite sets. In this first version of the L4

mathematical model, there is exactly one participant of each **role**. **All events are modelled as actions**, and a special **role Env** is used to model events that have no agent (i.e. **role**).

A **trace** is a sequence of **events**. The **time stamps** of the **events** must be nondecreasing. Thus, within the smallest unit of time, any number of **events** can happen; however, they are always strictly ordered. The idea here is that we want **events** to be strictly ordered for simplicity and to minimize the size of the space of execution traces, but if we made the **time stamps** strictly increasing, we would need to be working at a level of granularity for time that is at least one level smaller than the smallest unit of time that would appear in an informal version of the contract (at least when **time unit** = days, since contracts that use days as their minimum unit generally do not require that all **events** happen on different days).

A **contract** has a fixed finite number of **states**, one of which is designated the **start state**, and which includes at least the following:

- **fulfilled**
- **breached**(X) for each nonempty subset X of the **roles**. There is also an **action** **breaches**(X) for each such X , and **breachEvent**(X, t) is defined as the **event** $\langle \text{breaches}(X), \text{Env}, t \rangle$

Between any two events in a **trace**, the **contract** is in some **global state** which consists of at least a **time stamp** for the current time and a **state** (in Section 2, global variables will be added).

A **contract** has a finite directed edge-labeled multigraph¹ which we might call its **skeleton**; the nodes are the **states**, and each directed edge, which we will call a **transition**, is labeled with an **action**. The **skeleton** is the part of the **contract** that is easy to visualize. Some notation:

- For r a **role**, an **r -transition** is a **transition** whose **role** is r .
- For a an **action**, an **a -event** (**a -transition**) is an **event** (**transition**) whose **action** is a .
- For s a **state**, the **incoming s -transitions** (**outgoing s -transitions**) are the edges coming into (going out of) s .

Every **transition** is one of the following three types. They will be explained in more detail in the next section.

¹By this I mean there may be multiple edges from one node to another, but they must have different labels.

- A may next transition defines permitted **events**.
- A relievable must next transition defines the most-used kind of obligated **events**. These are obligations that are relieved by the performance of a permitted **event** *by some other* agent.
- A must next transition defines the strongest kind of obligated **events**.

Note that the events defined by **relievable must next transitions** and **must next transitions** are also considered permitted **events**.

We say that a **transition** τ and an **event** $E = \langle a, r, t \rangle$ are compatible iff they have the same **action** a and the same **role** r . This definition will be modified in Section 2 when we add **event** parameters.

Since the environment **Env** cannot breach a contract or be *obligated* to do anything, no **Env**-transition can be a **must next transition** or a **relievable must next transition**. That completes the definition of the finite directed graph **skeleton** of a **contract**.

Each **transition** τ is also associated with a transition guard $\text{transGuard}_\tau(\cdot)$ relation. For **simple contracts**, it is just a relation on **time stamps**, and a **transition** τ is enabled upon entering a **global state** with **time stamp** t iff $\text{transGuard}_\tau(t)$ is true.²

Each **transition** τ is also associated with a deadline function $\text{deadline}_\tau(\cdot)$, which yields a deadline. $\text{deadline}_\tau(t)$ is either a **time stamp** after t , or the special element ∞ . The **deadline** for a **transition** is when:

- an **enabled may next transition** (a kind of permission) expires³.
- an **enabled must next transition** (the strong form of obligation) causes a breach by role_τ ⁴ if a **compatible event** has not been performed by the deadline.
- an **enabled relievable must next transition** (the weak form of obligation) causes a possibly-joint breach by role_τ if a **compatible event** has not been performed by the deadline **and** no other permitted **event** is performed by the deadline.

²Currently, LSM examples are written assuming the **transition guards** of a **state** s 's **transitions** get evaluated only once upon entering the **state**. It would also be reasonable to guess that they get evaluated once per **time unit** while the **contract** is in that state. This is not ideal.

³Todo: expires should probably be a defined term.

⁴Which recall, in this formal model means a transition to the state $\text{breached}(\{\text{role}_\tau\})$

For **simple contracts**, a **deadline function** is just a function from **time stamps** to $\text{timeunit} \cup \text{timestamps}$. If d is such a function, and a state is entered at **time stamp** t , then:

- If $d(t) \in \text{timestamps}$, the deadline is $d(t)$.
- If $d(t) \in \text{timeunit}$, the deadline is $t + d(t)$.

The **transition guards** must satisfy the following conditions, which would be statically verified in a language for **contracts**. We give the **simple contracts** definitions here, but these conditions will be used in Section 2 as well.

unambiguous absolute obligation condition: For every **time stamp** t , if some **transition guard** of a **must next transition** evaluates to true (at t) then every other **transition guard** evaluates to false (at t).

choiceless relievable obligations condition: For every **role** r and **time stamp** t , if one of r 's **relievable must next transitions**'s **transition guards** evaluates to true (at t) then any other **relievable must next transitions** for r evaluate to false (at t).

breach or somewhere to go condition: If it is possible for all the **enabled non-Env transitions** to expire simultaneously, without causing a breach (which entails that there are no enabled **must next transitions** or **relievable must next transitions**) then there must be an **Env-transition** with **deadline** ∞ .

1.1 Execution for simple contracts

A **simple contract** of course starts in its **start state**. Let E_1, E_2, \dots be a finite or infinite **trace** (recall: a sequence of **events**), as defined in Section 1. Let G_i be the **global state** that follows E_i for each i .

G_0 is $\langle \text{startstate}, 0 \rangle$.

Let $i \geq 0$, and assume execution is defined up to entering G_i . To reduce notational clutter, let us use the aliases:

$$G = \langle s, t \rangle = G_i \quad E = E_i \quad G' = \langle s', t'^5 \rangle = G_{i+1}$$

Case 1: There is some **enabled must next transition** τ in G . If there is any other **enabled transition**, then this **contract** (not just this **trace**) violates the **unambiguous absolute obligation condition**, and so is invalid.⁶

⁵Note that t' is E 's **time stamp**

⁶Recall that a language (tool) for **simple contracts** will verify that such a thing can't happen.

- If E is **compatible** with τ and E happens within τ 's deadline, then the next state must be target_τ .⁷ This means E fulfilled the obligation created by τ .
- Otherwise, E must be $\text{breachEvent}(\text{role}_\tau, \text{deadline}_\tau(t) + 1)$.

Case 2: There is no **enabled must next transition** in G . From the set of **enabled may next transitions** of s and the set of **enabled relievable must next transitions** in G , compute the deadline for each, and discard the **transitions** whose deadline has passed by the time E happens;⁸ let T_p be the resulting set of **transitions**. From the set of **enabled relievable must next transitions** in G , compute the deadline for each, and discard the **transitions** whose deadline is not the unique minimal **time stamp** t^* within that set; let T_o be the resulting set, and let R be $\{\text{role}_\tau \mid \tau \in T_o\}$. Then E is either:

- An event compatible with T_p .
- $\text{breachEvent}(R, t^*)$.⁹ This means that all of the **roles** whose **enabled relievable must next transition** expire earliest (at t^*) are jointly responsible for the breach.

The **breach or somewhere to go condition** ensures that one of those two cases will apply. In particular, it implies that at least one of T_p or R is nonempty.

2 Infinite state: Global Variables and Event Parameters

Todo?: Move Event Parameters to its own section. A lot can be done with infinite state systems without them.

We introduce a set of basic datatypes \mathbb{T} , which includes at least \mathbb{B} , \mathbb{N} , and \mathbb{Z} . Add to the definition of **contract**:

- A fixed finite set of typed **global vars**. The **global vars** are ordered, so we may describe their collective types as a single tuple $\text{GVarTypes} \in \mathbb{T}^*$. Add to the definition of **global state** an assignment of values to the **global vars**. We'll call such an assignment a global vars assignment.

⁷i.e. if $t' \leq \text{deadline}_\tau(t)$ then $s' = \text{target}_\tau$.

⁸i.e. discard τ if $\text{deadline}_\tau(t) > t'$.

⁹Obviously not possible if R is empty

- An assignment of types (\mathbb{T} -tuples) to the **actions**. This allows **events** to have parameters. We refer to such a type as an action-parameters domain, and the specific **action-parameters domain** for **action** a is paramtypes_a .

The **event** definition receives the following generalizations:

- Each **action** a additionally has a **global vars transform**, denoted transform_a , which is a function from $\text{gvartypes} \times \text{paramtypes}_a \times \text{timestamps}$ to gvartypes .
- The definition of the **transition guard** of a a -transition is generalized: it may now depend on the values of the **global vars**; i.e. it is now a relation on $\text{timestamps} \times \text{gvartypes}$.

The conditions on **transition guards** are updated in unsurprising ways. For every **state** s :

unique absolute obligation condition: For every **global state** G whose (local) **state** is s , if one of s 's **must next transitions** evaluates to true (on G) then every other **transition guard** of s evaluates to false.

role-unique relievable obligations condition: For every **role** r and **global state** G whose (local) **state** is s , if one of s 's **relievable must next transitions** with **role** r evaluates to true (on G) then every other of s 's **relievable must next transitions** with **role** r evaluates to false.

breach or somewhere to go condition: If it is possible for all the **enabled non-Env transitions** to expire simultaneously, without causing a breach (which entails that there are no enabled **must next transitions** or **relievable must next transitions**) then there must be an **Env-transition** with **deadline** ∞ .

Note (probably to move to some other section or document): it will often be the case in a language for **contracts** that we simultaneously define an **action** a and a **state** JH_a (for “ a Just Happened”, to fit its literal meaning). In this case, the incoming JH_a -transitions are exactly the set of a -transitions. As a convenience, a language for **contracts** will allow the outgoing JH_a -transitions to depend directly on a 's parameters (that is, for the **transition guard** and **event-schema constructor** to depend on a 's parameters). This is merely a convenience because, as we will see when we define execution, one can achieve the same effect by introducing new **global vars** that are only used by a and JH_a ; a uses transform_a (recall, its **global vars transform**) to save its parameter values to those new **global vars**, so that the outgoing JH_a -transitions can then refer to them.

2.1 Execution

3 Event Schema

4 May-Later and Must-Later

This section does not actually change the definition of **contract**. Instead, it defines an often-useful **contract** structure that is likely to be supported with custom syntax in a language for **contracts**.

We have so far been noncommittal about what types are available for **global vars** and **action-parameters domains**. We will see later that the types strongly affect expressivity. As a special case, the reader should convince themselves that any **contract** that uses only boolean (or other finite domain) types can be simulated by a **simple contract** (using a much larger number of **states**).