Linear State Machines Formal Model

Legalese.com

October 19, 2017

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

Contents

1 Events, Time, Traces, Finite State Contracts

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

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 multigraph[1] which we might call its skeleton; the nodes are the states, and each directed edge, which we will call an edge, 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-edge is an edge whose role is r.
- For a an action, an a-event (a-edge) is an event (edge) whose action is a.
- For s a state, the incoming s-edges (outgoing s-edges) are the edges coming into (going out of) s.

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

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

- A may-next edge defines permitted events.
- A relievable must-next edge 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 edge defines the strongest kind of obligated events.

Note that the events defined by relievable must-next edge and must-next edge are also considered permitted events.

We say that an edge e 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-edge can be a must-next edge or a relievable must-next edge. That completes the definition of the finite directed graph skeleton of a contract.

Each edge e is also associated with a edge guard $\mathsf{edgeGuard}_e(\cdot)$ relation. For simple contracts, it is just a relation on time stamps, and an edge e is enabled upon entering a global state with time stamp t iff $\mathsf{edgeGuard}_e(t)$ is true.[2]

Each edge e is also associated with a deadline function $\mathsf{deadline}_e(\cdot)$, which yields a deadline. $\mathsf{deadline}_e(t)$ is either a time stamp after t, or the special element $\underline{\infty}$. The deadline for an edge is when:

- an enabled may-next edge (a kind of permission) expires[3].
- an enabled must-next edge (the strong form of obligation) causes a breach by role_e[4] if a compatible event has not been performed by the deadline.
- an enabled relievable must-next edge (the weak form of obligation) causes a possibly-joint breach by role_e if a compatible event has not been performed by the deadline **and** no other permitted event is performed by the deadline.

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

[2]Currently, LSM examples are written assuming the edge guards of a state s's edges 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.

[3]Todo: expires should probably be a defined term.

[4]Which recall, in this formal model means a transition to the state $\mathsf{breached}(\{\mathsf{role}_e\})$

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

The edge guards must satisfy the following conditions, which would be statically verified in a contract-definition language. 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 edge guard of a must-next edge evaluates to true (at t) then every other edge 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 edge's edge guards evaluates to true (at t) then any other relievable must-next edge for r evaluate to false (at t).

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

1.1 Execution for simple contracts

A simple contract of course starts in its start state. Let $E_1, E_2, \ldots$ 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 $\langle \mathsf{startstate}, 0 \rangle$.

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

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

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

[5]Note that t' is E's time stamp

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

- If E is compatible with e and E happens within e's deadline, then the next state must be target_e.[7] This means E fulfilled the obligation created by e.
- Otherwise, E must be $\mathsf{breachEvent}(\mathsf{role}_e, \mathsf{deadline}_e(t) + 1)$.

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

- An event compatible with T_p.
- $\mathsf{breachEvent}(R, t^*)$.[9] This means that all of the roles whose enabled relievable must-next edge 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 with Global Variables

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 $\mathsf{GVarTypes} \in \mathbb{T}^*$.

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. A particular global vars assignment initvals for the values of the global vars in the unique start state is required for a contract. Alternatively, one may omit some initial values, which results in a contract template; the meaning should be obvious.

The event definition receives the following generalizations:

[7] i.e. if $t' \leq \mathsf{deadline}_e(t)$ then $s' = \mathsf{target}_e$.

[8] i.e. discard e if $\mathsf{deadline}_e(t) > t'$.

[9] Obviously not possible if R is empty

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

The conditions on edge 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 edge evaluates to true (on G) then every other edge 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 edge with role r evaluates to true (on G) then every other of s's relievable must-next edge with role r evaluates to false.

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

Note (probably to move to some other section or document): it will often be the case in a contract-definition language 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-edges are exactly the set of a-edges. As a convenience, a contract-definition language will likely allow the outgoing JH_a-edges to depend directly on a's parameters (that is, for the edge guard 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 $\mathsf{transform}_a$ (recall, its global vars transform) to save its parameter values to those new global vars, so that the outgoing JH_a-edges can then refer to them.

2.1 Execution

3 Event Parameters and Schema

Add to the definition of contract 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 $\mathsf{paramtypes}_a$.

Each a-edge gets assigned an event schema called $\mathsf{eventschema}_a$. This is a function from gvartypes $\times$ timestamps to a set of a-events. Equivalently, it is a relation on gvartypes $\times$ timestamps $\times$ a-events, and that is likely how it will be represented in a contract-definition language. We call a set of a-events an event schema. In many cases, an event schema will be a singleton set. It is only for Env-events that singleton event schema allow us to express things that we couldn't otherwise; in all other cases, singleton event schema can be simulated efficiently using globalvars. Nonetheless, we will recommend using singleton event schema for events other than Env-events. Check for yourself that the definitions from the previous section do not need significant changes for singleton eventschema.

Non-singleton event schema are most useful for an infinite or large choice of actions (and, in the case of Env-events, for infinite or large nondeterminism).

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 contract-definition language.

We have so far been noncommittal about what types are available for global vars.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).