华东师范大学(East China Normal University)
URL—http://faxulty.ecnu.edu.cn/chenyixiang
yxchen@sei.ecnu.edu.cn

STeC: A Location-Triggered Specification Language

Yi-Xiang Chen (陈仪香)

Brose Team @ Software Engineering Institute(SEI)
East China Normal University(ECNU)
Shanghai, China

Second DAESD Workshop, March 28, 2012, Sophia-Antipolis, France

Outlines

- Background and related work
- Agents and Processes
- Syntax of STeC
- Transition Systems
- Operational Semantics
- The Railroad Crossing Issue
- Conclusion and Future Work

Background and Related Work

- Distributed embedded system: Internet of Things (IoT), Cyber-Physical Systems (CPS), and Cloud/Grid Computing.
- Formal Method is important to improve the safety and security of real-time systems, (Heitmeyer 1996)
- To give system developers and customers greater confidence that real-time systems satisfy their requirements, especially their critical requirements.
- Adding the time fact or time operator into specification languages is a typical method for developing the specification of real-time systems.

Background and Related Work

- Timed CSP (Reed and Roscoe, 1986): a real-time extension of the process language CSP by adding a single primitive WAITt.
- Schneider (1995): the operational semantics of timed CSP.
- Davies and Scheneider(1995): gave a brief history of timed CSP.
- Ouaknine and Schneider(2006): reviewed the development of the process algebra timed CSP.

Background and Related Work

- Wang Yi(1992): an interleaving model for real-time systems by adding the temporal, primitive action prefix, $\mu@t.P$ to Milner's CCS, where t is a time variable.
- Hennessy and Regan(1995): a timed process language for timed system by introducing a new action δ which is meant to denote idling until the next clock cycle.
- Rounds and Song (2003): the Φ-calculus—a language for distributed control of reconfigurable embedded systems by adding active environments which flow continuously over time to Milner's π-calculus.
- Jacquet and Linden (2009) introduced a timed coordination language by adding the notion of duration for which units of time a token will stay in the dataspace.

Background and Related Work

- Timed automata (Alur and Dill 1994): to model the behavior of real time systems over time.
- Fersman et al (2006): used timed automata to describe task arrival patterns for relaxing the stringent constraints on task arrival times and to analyze the schedulability of fixed-priority systems.
- Abdulla et al(2010): gave a sampled semantics of times automata for dense time behavior.

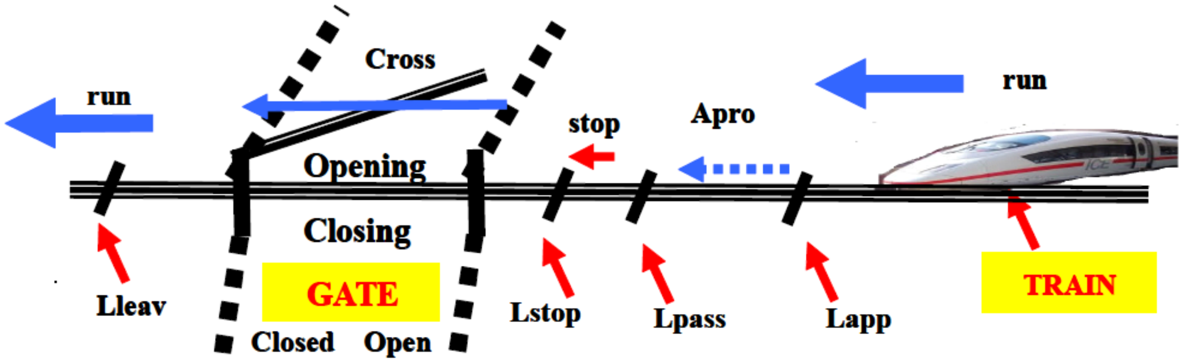
Background and Related Work

- The railroad crossing: a point at which a railway and a road intersect on the same level.
- In order to avoid collision, a gate is built at the crossing point.
- Safety issue: The gate must be closed when a train passes the crossing point.
- Formal Methods to specified this issue in terms of timed CSP and timed automata.

Background and Related Work

- Roscoe (1985): a CSP solution to the “trains” problem.
- Heitmeyer and Lynch (1994): a solution to the generalized railroad crossing in terms of timed automata.
- Archer and Heitmeyer(1996): applied the mechanical proof system PVS to a solution of the Generalized Railroad Crossing (GRC) problem based on the timed automata.
- Beauquier and Slissenko (1997): a formal analysis of the railroad crossing problem, to describe semantics of algorithms with continuous time.
- Damm et al (2007): a verification methodology for cooperating traffic agents and illustrated this approach with a variant of the European Train Control System (ETCS).
- Mostafa et al(2010): an intelligent railway crossing control system for multiple tracks that features a controller which receives messages from incoming and outgoing trains by sensors based on radio.

The Railway Crossing Issue



We address that both trains and gates are intelligent agents.

- A train has three components: four locations, three actions and two interactions communicating with gate.
- Gate is also an intelligent agent. It is autonomous (e.g., it opens and closes it by itself), and communicates with train.

Our Contribution

- This presentation introduces a spatial-temporal consistence language for real-time systems (Shortly, STeC).
  - The consistence requires that an agent do its tasks at the required location or time.
  - This language is a location-triggered specification language for real-time systems.
  - The interaction between real-time agents deals with communications.

Our Contribution

- STeC looks like an **extension** of process precess algebra, Hoare's CSP and Milner's CCS in syntax.
  - The execution time of actions and states of agents are stressed.
  - Two kinds of interrupts- time and interaction interrupts are considered.
  - Following the Dijkstra's guard style, nondeterministic choice phase guarded by actions and states is introduced.
  - The operational semantics is introduced in the style of Plotkin's version.
- Agents as smart objects will specify in terms of the language STeC.
- The railroad crossing problem is specified in terms of this language STeC, since gate and train are considered as agents.

Agents and Processes

- Oxford Dictionary of Computing: an agent is an autonomous system that receives information from its environment, processes it, and performs actions on that environment.
- A robot: an example of an agent that perceives its physical environment through sensors and acts through effectors.
- Networked Agents: Vehicle network and wireless sensor network are classical examples of networked agents.
  - Cars and sensors are examples of agents that receive information from other agents by communication mechanism and from its environments through sensors, processes it and performs actions by themselves on that environment.

Agents and Processes

- The networked environment: distributed systems.
  - *Organization:* The process of identifying and managing the relationships between agents.
  - *Communication mechanism:* The method of describing the communication protocol between agents.
  - *Storage method:* The approach of how a message is stored into and extracted from storage space as well as what forms of messages are stored at. This storage will be denoted as the notation Σ.

Agents and Processes

- Agents are smart objects.
  - communicating each other and reacting with environment
  - processing/computing the information
  - taking action and keeping status
- Processes: used to define the behaviours of agents.
- What we do is to develop a specification language to specify and analyze the behaviour of an agent in a networked environment.
- This language, called **STeC**, is provided with the triggered location at which agents stay and with the spatial-temporal consistence for real-time systems.
  - the syntax to specify processes
  - the semantics to analyze the meaning of each component and
  - the reasoning mechanism to show the evolution of behaviors of agents.

Agents and Processes

- Each agent: an intelligent body/system including such parts as communication, actions and states.
- The communication between agents is provided through operators of Send and Get which are triggered by locations at which an agent stays.
- An action: a process of which that agent takes a task. α stands for actions. The action is considered to have a duration, e.g., **running**(3s) means that that agent will run with 3 seconds.
- State of an agent is a form of what that agent appears. β stands for states. We consider how long a state will keep, e.g., Closed(4s) means that the state Closed will last 4 seconds.

Syntax of STeC

we stress two basic concepts: location and time, and a property: consistence between time and location.

- A location: a physical place or a state of agent for which l stands.
- The time concept: at what time t and with how long it maintains.
- Communication between agents: these commands **Send** and **Get**, triggered by locations
- The notation $m_{G \Rsh G'}$: to denote the message m sent to G' by agent G.
- The storage Σ: recording those messages of the form $m_{G \Rsh G'}$. Those messages disappear as long as they are extracted. In other words, the messages are used only one time.

Back
Close

Syntax of STeC

Atomic Commands of **Send** and **Get**:

$$A \ ::= \ \mathsf{Send}^{G}_{(l,t,G')}(m) \mid \mathsf{Get}^{G}_{(l,t,G')}(m)$$

- The atomic command $\mathsf{Send}^{G}_{(l,t,G')}(m)$ defines the process that the agent G sends a message m (denoted as $m_{G \Rsh G'}$) to the agent G' at location l and at time t.
- The command $\mathsf{Get}^{G}_{(l,t,G')}(m)$ means that agent G receives a message $m_{G' \Rsh G}$ from G' at location l and at time t.

Syntax of STeC

Each agent may have some actions and states. α stands for actions and β stands for states.

- The notation $\alpha_{(l,t)}(\delta)$ to represent that the action α is executed at location l and at time t and go on δ unit times
  - action $\textsf{closing}_{(\textsf{Open},12\text{am})}(3s)$: the action **closing** is executed at location **Open** and at time 12am and keep 3 seconds.
- The notation $\beta_{(l,t)}(\delta)$ to represent that at location t and at time t, the state β lasts δ unit times.
  - The state $\textsf{closed}_{(\textsf{Closed},13\text{pm})}(5s)$: at location **Closed** and at time 13pm, the states **closed** keep 5 seconds.

The duration time δ takes the non-negative real numbers as well as the infinite ∞.

- Statement $\alpha(\infty)$: the action α will go on and not stop.
- Statement $\alpha(0)$: the action α does not consume the time.

Syntax of STeC

This language STeC is of the Dijkstra's guarded language. The guards is, denoted as B, defined as follows:

$$B \ ::= \ \alpha^G_{(l,t)}(\delta) \mid \beta^G_{(l,t)}(\delta) \mid \neg B \mid B \wedge B$$

The guard B takes the Boolean/truth values: true T and false F.

- If the action $\alpha(\delta)$ finishes within the δ unit time then it is true else false.
- $\beta(\delta)$ is a state which waits with duration of δ unit time. If the status β keeps δ unit time then it takes the value of truth else the value of false.

Syntax of STeC

The syntax of this language STeC is defined in the Backus-Naur Form as follows. We ignore those superscripts in atomic commands and actions as well as states without confusion

$$\begin{aligned} P \ ::= \ & \mathsf{Stop}_{(l,t)} \mid \mathsf{Skip}_{(l,t)} \mid \mathsf{Send}_{(l,t,G')}(m) \mid \mathsf{Get}_{(l,t,G')}(m) \mid \\ & \alpha_{(l,t)}(\delta) \mid \beta_{(l,t)}(\delta) \mid P; P \mid P \parallel P \mid \\ & [\!]_{i \in I} B_i \longrightarrow P_i \mid P \trianglerighteq_\delta P \mid P \trianglerighteq ([\!]_{i \in I} A_i \longrightarrow P_i) \mid \end{aligned}$$

- **Stop** is a non-terminating idle process.
- **Skip** terminates immediately with no effect on the process.
- ; is the sequent composition.
- $\parallel$ is the parallel operator. Process P and Q of process $P \parallel Q$ always execute independently and interact asynchronously through **Send-Get** communication.

Syntax of STeC

$$
\begin{aligned}
P \quad ::= \quad & \mathsf{Stop}_{(l,t)} \mid \mathsf{Skip}_{(l,t)} \mid \mathsf{Send}_{(l,t,G')}(m) \mid \mathsf{Get}_{(l,t,G')}(m) \mid \alpha_{(l,t)}(\delta) \mid \beta_{(l,t)}(\delta) \mid \\
& P; P \mid P \parallel P \mid [\!]_{i\in I} B_i \to P_i \mid P \trianglerighteq_\delta P \mid P \trianglerighteq ([\!]_{i\in I} A_i \to P_i) \mid
\end{aligned}
$$

- Statement $[\!]_{(i\in I)}(B_i \longrightarrow P_i)$ is a nondeterminstic choice, B_i is a guard. Only one of these guards B_i is true. Thus, only one of those processes P_i executes. Its special case is the conditional statement $B \longrightarrow P\,\mathsf{Else}\,Q$, which means if the guard B is true then it behaves like P else like Q.
- Timeout Events $P \trianglerighteq_\delta Q$: initially behaves as P for up to δ time units and then as Q after δ time units. So, it is interrupted by the duration time δ.
- The sentence $P \trianglerighteq ([\!]_{i\in I} A_i \longrightarrow P_i)$ initially proceeds like P and is interrupted on occurrence of the atomic command A_i and then process like P_i.

Syntax of STeC

- Define the phase $\mathsf{Wait}_{(l,t)}(\delta)$: $\mathsf{Stop}_{(l,t)}(\infty) \trianglerighteq_\delta \mathsf{Skip}_{(l,t+\delta)}$.
- The command $\mathsf{Wait}_{(l,t,G)}(m) = \mathsf{Stop}_{(l,t)}(\infty) \trianglerighteq (\mathsf{Get}_{(l,t',G)}(m) \rightarrow \mathsf{Skip}_{(l,t'+\tau(\mathsf{Get}))})$ with $t \leq t'$, waiting for the message coming from agent G at location l and time t.
- The notation $\mathcal{P}$ to denote the set of all processes.
- To easily express successful termination, we shall introduce a special terminating symbol E for the case that no executable statements exist.
- To meet the intuition, we shall always rewrite precesses of the form $E; P, P; E$ and $P \parallel E, E \parallel P$ as P.
- Define the extended set of processes as $\mathcal{P}^e = \mathcal{P} \cup \{E\}$.
  - The structure $(\mathcal{P}^e, E, ;, \parallel)$ is a bimonoid, and more, $(\mathcal{P}^e, E, \parallel)$ is a commutative monoid.
  - $E; P = P; E = P$ and $E; E = E$, as well as, $E \parallel P = P \parallel E = P$ and $E \parallel E = E$. Moreover, $P \parallel Q = Q \parallel P$.

Execution Time of Processes

The execution time of programs is a basic notion for real-time systems.We define the concept of execution time, denoted as $\tau(P)$, of process P.

- The execution time of atomic commands Send and Get, denoted as $\tau(\mathsf{Send})$ and $\tau(\mathsf{Get})$, are preassigned depended on the communicative machine.
- Define the execution time $\tau(B)$ of a guard B in the following table.

$\tau(\alpha(\delta)) = \delta$	$\tau(\beta(\delta)) = \delta$
$\tau(B_1 \wedge B_2) = \tau(B_1) + \tau(B_2)$	$\tau(\neg B) = \tau(B)$

Execution Time of Processes

The execution time $\tau(P)$ of process P is defined in the following table.

$\tau(\mathsf{Stop}) = \infty$	$\tau(\mathsf{Skip}) = 0$
$\tau(\mathsf{Send}_{(l,t,G)}(m)) = \tau(\mathsf{Send})$	$\tau(\mathsf{Get}_{(l,t,G)}(m)) = \tau(\mathsf{Get})$
$\tau(\alpha_{(l,t)}(\delta)) = \delta$	$\tau(\beta_{(l,t)}(\delta)) = \delta$
$\tau(P_1; P_2) = \tau(P_1) + (P_2)$	$\tau(P_1 \parallel P_2) = \mathrm{Max}(\tau(P_1), \tau(P_2))$
$\tau([\!]_{i\in I}(B_i \rightarrow P_i) = \sum_{i\in I} \tau(B_i) + \mathrm{Max}_{i\in I}\tau(P_i)$	$\tau(P \rhd_\delta Q) = \delta + \tau(Q)$
$\tau(P \rhd ([\!]_{i\in I} A_i \rightarrow Q_i)) = \tau(P) + \mathrm{Max}_{i\in}\tau(Q_i)$	$\tau(E) = 0$

Back
Close

Execution Time of Processes

- $\tau(\mathsf{Stop}) = \infty$: the process Stop will execute for ever
- $\tau(\mathsf{Skip}) = 0$: The execution of Skip does not consume the time.
- $\tau(\mathsf{Send}_{(l,t,G)}(m)) = \tau(\mathsf{Send})$ and $\tau(\mathsf{Get}_{(l,t,G)}(m)) = \tau(\mathsf{Get})$.
- $\tau(\alpha_{(l,t)}(\delta)) = \delta$ and $\tau(\beta_{(l,t)}(\delta)) = \delta$: Meeting the intuition.
- $\tau(P_1; P_2) = \tau(P_1) + (P_2)$: Their executions in a sequence.
- $\tau(P_1 \parallel P_2) = \mathrm{Max}(\tau(P_1), \tau(P_2))$: The parallel statement terminates only if all sub-statements terminate. So, its time is the maximal one.
- $\tau([]_{i\in I}(B_i \rightarrow P_i) = \sum_{i\in I} \tau(B_i) + \mathrm{Max}_{i\in I}\tau(P_i)$: the maximal execution time and computing truth values of guards needs times.
- $\tau(P \rhd_\delta Q) = \delta + \tau(Q)$: P is interrupted by the delay time δ.
- $P \rhd ([]_{i\in I} A_i \rightarrow Q_i)$: We ignore the execution time of the interrupt command A_i, since the execution time of P covers one of the interrupt A_i due to their parallel processing.
- $\tau(E) = 0$: E just is a symbol and never executed.

Execution Time of Processes

We may compute the time of statement Wait. Indeed, we have the following results.

- $\tau(\mathsf{Wait}_{(l,t)}(\delta)) = \tau(\mathsf{Stop}_{(l,t)} \trianglerighteq_\delta \mathsf{Skip}_{(l,t+\delta)}) = \delta$;
- $\tau(\mathsf{Wait}_{(l,t,G)}(m)) = \tau(\mathsf{Stop}_{(l,t)} \trianglerighteq (\mathsf{Get}_{(l,t',G)}(m) \longrightarrow \mathsf{Skip}_{(l,t'+\tau(\mathsf{Get}))})) = \tau(\mathsf{Stop}) + \tau(\mathsf{Get})$. This time is not known, since we do not know how long we need wait for the message coming.

Transition System

- The computation and reasoning in **STeC** may be modeled by a transition system written in Plotkin's style.
- Configurations to be considered consist of a precess or E together with a multi-set of spatial stores denoting the messages currently available.
- Define a storage, denoted by Σ, as the set of elements of the form $m_{G \Rsh G'}$, where $m_{G \Rsh G'}$ is a message that is sent by agent G to agent G'.
- Notation Σ^* to denote the family of all finite subsets of Σ which σ will stand for and be called as a store.
- All agents will be concurrent on that storage.
- Only statements Send and Get change the storage. Indeed, Send will add messages into and the Get removes messages from the storage.

Transition System

- Define the concept of environment as the binary (σ, u), where $\sigma \in \Sigma^*$ and u is a glob clock of that distributed agent system
  - taking its value from the Time-the set of the non-negative real numbers $[o, +\infty)$ and the infinite ∞.
  - The value ∞ denotes the infinite duration.
- The notation $\mathcal{E}$ will denote the set of all environments. That is, $\mathcal{E} = \Sigma^* \times \mathsf{Time}$.

Transition System

Based on the environment (σ, u), we define the truth value, $\mathcal{T}(B)$, of a guard B as a function from the environment set to the truth value $\{T, F\}$ as follows.

- $\mathcal{T}(\alpha^{G}_{(l,t)}(\delta)(\sigma, u)) = T$ if and only if the agent G, at location l and time t, completes its action α in δ unit times.
- $\mathcal{T}(\beta^{G}_{(l,t)}(\delta)(\sigma, u)) = T$ if and only if the agent G, at location l and time t, maintains its state β with δ unit times.
- $\mathcal{T}(\neg B)(\sigma, u) = T$ if and only if $\mathcal{T}(B)(\sigma, u) = F$;
- $\mathcal{T}(B_1 \wedge B_2)(\sigma, u) = T$ if and only if $\mathcal{T}(B_1)(\sigma, u) = \mathcal{T}(B_2)(\sigma, u) = T$.

Transition System

- Define the set of configurations **Stconf** as $\mathcal{P}^e \times \mathcal{E}$.
- Configuration, $(P, (\sigma, u))$, is denoted as $(P, \sigma)_u$, where P is a process or E.
- The $(E, \sigma)_u$ is a terminating mark, since no processes execute.

We begin by defining the kind of transition system we will use for modelling executions of processes.

- A spatial-temporal transition system: a subset $\longrightarrow$ of **Stconf**$\times$**Stconf**.
- An element $((P, \sigma)_u, (P', \sigma')_{u'}) \in\longrightarrow$: denoted by the notation $(P, \sigma)_u \longrightarrow (P', \sigma')_{u'}$.

Operational Semantics for **STeC**

An operational semantics for a computer programming language defines the meaning of programs written in that language in terms of

- how a machine is intended to execute them step by step
- reporting the traces of all the computation steps made during the executions both in terms of the contents of the shared space and the moments of execution.

Operational Semantics for **STeC**

- Operational Semantics for Stop and Skip
- Since Stop is a non-terminating idle process except for it is interrupted and
- Skip terminates immediately with no effect on the process.

$$\frac{a = l, u = t}{(\mathsf{Stop}_{(l,t)}, \sigma)_u \longrightarrow (\mathsf{Stop}_{(l,t+1)}, \sigma)_{t+1}}$$

$$\frac{a = l, u = t}{(\mathsf{Skip}_{(l,t)}, \sigma)_u \longrightarrow (E, \sigma)_u}$$

Operational Semantics for **STeC**

- Operational Semantics for Action and State
- The execution of action $\alpha(\delta)$ consumes δ unit times and status $\beta(\delta)$ keeps δ unit times.
- After δ unit times, the action α will finish and the status β disappears.

$$\frac{a = l, u = t}{(\alpha_{(l,t)}(\delta), \sigma)_u \longrightarrow (E, \sigma)_{u+\delta}}$$

$$\frac{a = l, u = t}{(\beta(\delta), \sigma)_u \longrightarrow (E, \sigma)_{u+\delta}}$$

Operational Semantics for **STeC**

- Operational Semantics for Communication Commands

Interaction commands **Send** and **Get** carry out the interaction between messages. The **Send** command adds the new message into the store, Whilst the command **Get** takes away the message from the store. Communication commands consumes time when they execute. The below are their operational semantics.

$$\frac{a = l, u = t}{(\mathsf{Send}^{G}_{(l,t,G')}(m), \sigma)_u \longrightarrow (E, \sigma \cup \{m_{G \Rsh G'}\})_{u+\tau(\mathsf{Send})}}$$

$$\frac{a = l, u = t}{(\mathsf{Get}^{G}_{(l,t,G')}(m), \sigma \cup \{m_{G' \Rsh G}\})_u \longrightarrow (E, \sigma)_{u+\tau(\mathsf{Get})}}$$

Back
Close

Operational Semantics for **STeC**

- Operational Semantics for Sequent Statement

The sequent statement $P;Q$ is executed in order. Its semantics is as follows.

$$\frac{(P,\sigma)_u \longrightarrow (E,\sigma')_{u'}}{(P;Q,\sigma)_u \longrightarrow (Q,\sigma')_{u'}}$$

We can define a more general case to be

$$\frac{(P,\sigma)_u \longrightarrow (P',\sigma')_{u'}}{(P;Q,\sigma)_u \longrightarrow (P';Q,\sigma')_{u'}}$$

Operational Semantics for **STeC**

- Operational Semantics for Parallel Statement

Parallel statement $P \parallel Q$ at the environment (σ, u) and during the duration from u to u' has two cases:

- case 1. P and Q execute individually without the interaction of each other.
  - the parallel statement may change the store.
  - the store is the union of stores after executions of P and Q at the time u'
- Case 2. P and Q has an interaction of message, i.e., one executes the $\mathsf{Send}(m)$ command and one another does the $\mathsf{Get}(m)$ command.
  - the parallel does not change the store at the time u'.

In order to record the evolution of stores after the execution of parallel statement, we define an union operator $\uplus$ of stores as follows.

$$\sigma_1 \uplus \sigma_2 = \begin{cases} \sigma, & \text{if } \exists \sigma \text{and} m \text{such that} \sigma_1 = \sigma \backslash \{m\}, \sigma_2 = \sigma \cup \{m\} \\ \sigma_1 \cup \sigma_2, & \text{otherwise.} \end{cases}$$

Therefore, we define the operational semantics of parallel statement as

$$\frac{(P,\sigma)_u \longrightarrow (P',\sigma_1)_{u'} \wedge (Q,\sigma)_u \longrightarrow (Q',\sigma_2)_{u'}}{(P \parallel Q,\sigma)_u \longrightarrow (P' \parallel Q',\sigma_1 \uplus \sigma_2)_{u'}}$$

Operational Semantics for **STeC**

- Operational Semantics for Nondeterministic Choice
- Nondeterministic choice statement $[]_{i\in I}(B_i \longrightarrow P_i)$ likes only one P_i if the corresponding guard B_i is true.
- Notice that computing the truth value of guards consumes time and the truth value of guards is dependent on the environment of store and time. The store does not change.
- Computing the truth value of B_i will depend directly on the time of $u + \tau(B_1) + \cdots + \tau(B_{i-1})$.
- Notation $\oplus_{j=1}^{i}\tau(B_j)$ to denote the sum $\tau(B_1) + \cdots + \tau(B_i)$.
- $\mathcal{T}(B_i)(\sigma, u + \oplus_{j=1}^{i-1}\tau(B_j)) = T$ may deduce $\mathcal{T}(B_1)(\sigma, u) = \cdots = \mathcal{T}(B_{i-1})(\sigma, u + \oplus_{j=1}^{i-2}\tau(B_j)) = F$.

$$\frac{\mathcal{T}(B_i)(\sigma, u + \oplus_{j=1}^{i-1}\tau(B_j)) = T}{([]_{i\in I}(B_i \longrightarrow P_i), \sigma)_u \longrightarrow (P_i, \sigma)_{u+\oplus_{j=1}^{i-1}\tau(B_j)}}$$

Operational Semantics for **STeC**

- Operational Semantics for Timeout Event
- Timeout event statement $P \rhd_\delta Q$ will like P's behavior before δ time and then like Q's behavior after δ time. The δ is an interrupt command.
- If the execution time of P is less than δ then it immediately likes Q regardless of the duration δ.
- There exist two cases:
  - (1) the execution time $\tau(P)$ is more than or equal to δ, and
  - (2) $\tau(P)$ is less than, but not equal to δ.

$$\frac{(P,\sigma)_u \rightarrow (P',\sigma')_{u'}(u' \geq \delta)}{(P \trianglerighteq_\delta Q,\sigma)_u \rightarrow (Q,\sigma')_{u+\delta}} \quad \frac{(P,\sigma)_u \rightarrow (E,\sigma')_{u'}(u' < \delta)}{(P \trianglerighteq_\delta Q,\sigma)_u \rightarrow (Q,\sigma')_{u'}}$$

Operational Semantics for **STeC**

- Operational Semantics of Interrupt Statement
- The interrupt statement $P \trianglerighteq ([]_{i\in I} A_i \rightarrow Q_i)(P \neq A)$ likes P's behavior before the occurrence of interrupt command A_i and then likes Q_i's behavior after the execution of A_i.
- Since P is not any atomic communication command A, the execution of P does not change the store σ at any time.
- The execution of A_i depends on the environment (σ, u') due to the evolution of the time caused by the execution of P at the initial environment (σ, u).
- P is still executed when the A_i executes.

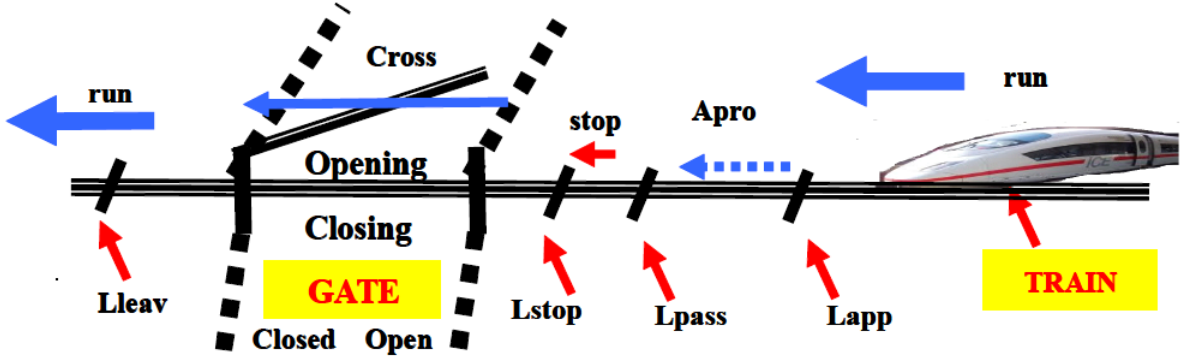
$$\boxed{\frac{(P,\sigma)_u \longrightarrow (P',\sigma)_{u'} \wedge (A_i,\sigma)_u \longrightarrow (E,\sigma')_{u'} \text{ for some } i}{(P \trianglerighteq ([]_{i\in I} A_i \rightarrow Q_i),\sigma)_u \longrightarrow (Q_i,\sigma'')_{u'}} \; (P \neq A)}$$

Operational Semantics for **STeC**

Up to now, we have established the operational semantics of the language **STeC**.

Railroad Crossing Issue

The railroad crossing problem is classical. It states how a train passes a gate at the railroad crossing safely. So, one requires that the gate is closed when a train passes.



Railroad Crossing Issue

A train has three components: four locations, three actions and two interactions communicating with gate.

- These four locations are denoted as Lappr, Lpass, Lstop, and Lleav.
  - Location Lappr is the place that train sends message Appr to gate, showing that train is coming and going to pass the crossing point.
  - Location Lpass is the place that train receives the message Cross from gate. If the train receives this message successfully then it will pass the crossing, else it will immediately stop.
  - Location Lstop is the place for train to stop when it does not receive the message Cross from gate at location Lpass for safety.
  - Location Lleav is the place that the train sends message Leav to gate. After getting this message, the gate may start closing.

- Those three actions of a train are Approach, Pass and Stop.
  - Action Approach(τ(Approach)) states the train continues running after passing the location Lappr and before reaching the location Lstop, lasting the τ(Approach) unit times.
  - Action Pass(τ(Pass)) states the train will pass the crossing after receiving the message Cross from gate at location Lpass until arriving at location Lleav, lasting the τ(Cross) unit times.
  - Action Stop(τ(Stop)) commands the train to stop quickly and stop at location Stop when it fails to get the message Cross from gate at location Lpass.
- Two interactions are Send and Get for train to communication with with gate.

Railroad Crossing Issue

Gate is an intelligent agent. A gate has three states/locations Open, Closed and Unclosed, two actions Opening and Closing as well as two interaction commands Send and Get.

- Three States:
  - State Open(τ(Open)) says that the gate is open. It may last the τ(Open) unit times.
    - The initial state of gate is Open.
    - After getting the message Appr sent by train at location Lappr, the gate starts action Closing.
  - State Closed(τ(Closed)) says that the gate is closed. It may last the τ(Closed) unit times.
    - The gate must be in the state Closed during that train passes the crossing.
    - When finishing the action closing in the required unit times, the gate sends message Cross to train before the train arrives at location Lpass.

- State Unclosed(τ(Unclosed) says that the gate is not closed. It may last τ(Uclosed) unit times.
  - ∗ It is dangerous that the gate is at the status Unclosed when the train passes the crossing.
  - ∗ When failing the action Closing in the required unit time, the gate is at the state Unclosed and must send message Noncross to train before the train arrives at the location Lpass so that the train stops at location Lstop.

- Two actions are Opening and Closing of gate, defined as follows.
  - Action Opening(δ) indicates that the gate opens autonomously and reaches the state Open within the δ unit times.
  - Action Closing(δ) indicates that the gate closed autonomously and reaches the state Closed within the δ unit times.

- Two interactions of a gate are Send and Get too.

Railroad Crossing Issue

We specify these two intelligent agents, **Train** and **Gate** as two series of processes written in terms of the language **STeC**, respectively.

The agent **Train** is specified as

$$
\begin{array}{l}
\mathsf{Run}(\infty);(\mathsf{Send}_{(\mathsf{Lapp},t,Gate)}(\mathsf{Appr}) \parallel \mathsf{Approach}_{(\mathsf{Lapp},t)}(\Gamma))\triangleright \\
\left(\begin{array}{l}
\mathsf{Get}_{(\mathsf{Lpass},t+\Gamma,Gate)}(\mathsf{Cross}) \rightarrow (\mathsf{Pass}_{(\mathsf{Lpass},t+\Gamma)}(\Delta); \\
\qquad\qquad (\mathsf{Send}_{(\mathsf{Lleav},t+\Gamma+\Delta,Gate)}(\mathsf{Leav}) \parallel \mathsf{Run}_{(\mathsf{Lleav},t+\Gamma+\Delta)}(\infty))) \\
\square \\
\mathbf{Get}_{(\mathsf{Lpass},t+\Gamma,Gate)}(\mathsf{Noncross}) \rightarrow (\mathsf{Stop}_{(\mathsf{Lpass},t+\Gamma)}(\Upsilon);\mathsf{Wait}_{(\mathsf{Lstop},t+\Gamma+\Upsilon,Gate)}(\mathsf{Cross}); \\
\qquad\qquad \mathsf{Pass}_{(\mathsf{Lstop},t')}(\Omega);(\mathsf{Send}_{(\mathsf{Lleav},t'+\Omega,Gate)}(\mathsf{Leav}) \parallel \mathsf{Run}_{(\mathsf{Lleav},t'+\Omega)}(\infty)))
\end{array}\right)
\end{array}
$$

Railroad Crossing Issue

The agent **Gate** is defined as

$$\mathsf{Open}(\infty) \rhd (\mathsf{Get}_{(\mathsf{Open},t+\tau(\mathsf{Send}_T),train)}(\mathsf{Appr}) \longrightarrow \mathsf{Closing}_{(\mathsf{Open},t+\theta_0)}(\Pi));$$

$$\left(\begin{array}{l}
(\mathsf{Closed}_{(\mathsf{Closed},t+\theta_0+\Pi)}(1) \longrightarrow (\mathsf{Send}_{(\mathsf{closed},t+\theta_1,train)}(\mathsf{Cross}); (\mathsf{Closed}_{(\mathsf{Closed},t+\theta_2)}(\infty)\rhd \\
\qquad (\mathsf{Get}_{(\mathsf{closed},t+\theta_3),train)}(\mathsf{Leav}) \longrightarrow \mathsf{Opening}_{(\mathsf{Closed},t+\theta_4)}(\zeta)))); \mathsf{Open}(\infty) \\
\square \\
(\mathsf{Unclosed}_{(\mathsf{Unclosed},t+\theta_0+\Pi)}(0) \longrightarrow (\mathsf{Send}_{(\mathsf{Unclosed},t+\theta_0+\Pi,train)}(\mathsf{Noncross}) \parallel \mathsf{Closing}_{(\mathsf{Unclosed},t+\theta_0+\Pi)}(\pi)) \\
\qquad ; (\mathsf{Closed}_{(\mathsf{Closed},t+\theta_5)}(1) \longrightarrow (\mathsf{Send}_{(\mathsf{closed},t+\theta_5+1,train)}(\mathsf{Cross}); \\
\qquad\qquad (\mathsf{Closed}_{(\mathsf{Closed},t+\theta_5+1+\tau(\mathsf{Send}_T))}(\infty) \rhd (\mathsf{Get}_{(\mathsf{Closed},t'+\Omega+\tau(\mathsf{Send}_T))}(\mathsf{Leav}) \longrightarrow \\
\qquad\qquad\qquad \mathsf{Opening}_{(\mathsf{Closed},t'+\Omega+\tau(\mathsf{Send}_T)+\tau(\mathsf{Get}_G))}(\zeta)))))); \mathsf{Open}(\infty)
\end{array}\right)$$

where $\theta_0 = \tau(\mathsf{Send}_T)) + \tau(\mathsf{Get}_G)$, $\theta_1 = \theta_0 + \Pi + 1, \theta_2 = \theta_1 + \tau(\mathsf{Send}_G), \theta_3 = \Gamma + \Delta + \tau(\mathsf{Send}_T), \theta_4 = \theta_3 + \tau(\mathsf{Get}_G), \theta_5 = \theta_0 + \Pi + \pi$, and t' is not known before Train receives the message **Cross** from Gate, since the execution time of **Wait** is not known before it finishes.

Railroad Crossing Issue

The railroad crossing problem is specified as the parallel agents: **Train** $\|$ **Gate**.

Proposition 1: If $\tau(\mathsf{Send}_T) + \tau(\mathsf{Get}_G) + \tau(\mathsf{Closing}) + 1 \leq \tau(\mathsf{Approach})$, then the crossing issue is safety, i.e., the gate is closed when the train passes the cross.

Proposition 2: We are supposed that $\tau(\mathsf{Get}_G) = n \times \tau(\mathsf{Send}_T)$. If $\tau(\mathsf{Send}) \leq \dfrac{\tau(\mathsf{Approach}) - \tau(\mathsf{Closing}) - 1}{n+1}$, then the crossing issue is safety.

Conclusion and Future Work

- This presentation introduced the syntax and operational semantics of the language STeC stressing the spatial-temporal consistence for real-time systems based on environments recording the stores of messages and the time.
- STeC is a kind of location-triggered specification language. It can be used to describe real-time systems with stressing of locations such as aircraft and bullet train and such computing system as distributive system.
- We described the railroad crossing problem as an interaction of two intelligent agents–train and gate and specify it in terms of this language STeC.
- We will pay attention on STeC's denotational semantics and its application for real-time systems.

Thanks!

Merci!

谢谢