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

A Spatio-Temporal Consistence Language

Yi-Xiang Chen (陈仪香)

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

DAESD Workshop, Nov 7, 2011

Outlines

- Abstractness
- Syntax
- Example
- Transition Systems
- Operational Semantics

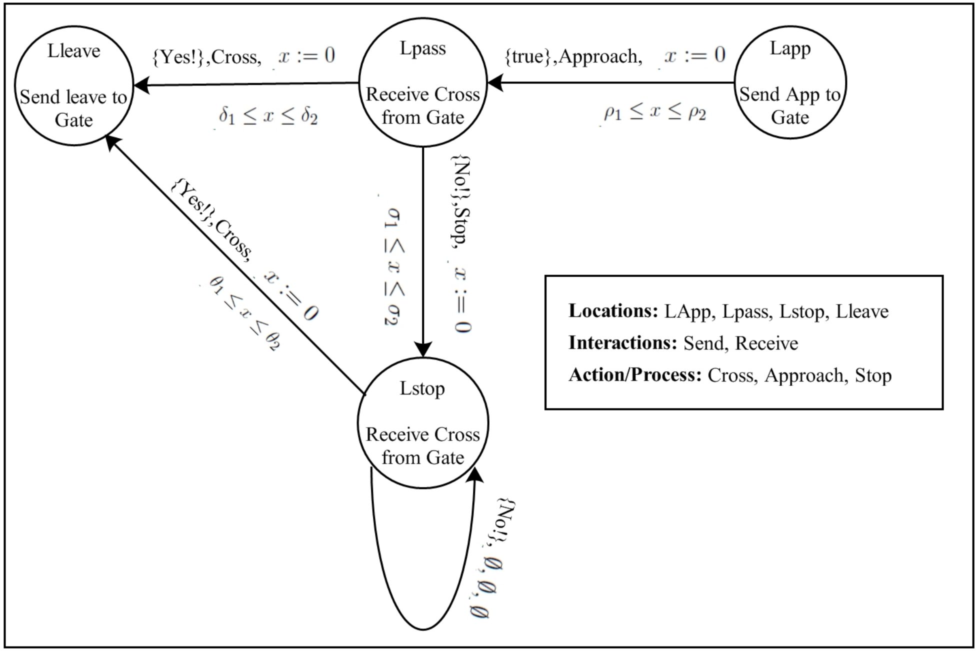
Abstractness

- This presentation introduces a spatio-temporal consistence language (shortly, STeC), based on precess algebra, Hoare's CSP and Milner's CCS.
- STeC describes the consistence between space and time. It means that a process will do its tasks at the required location and time. Therefore, in the syntax, STeC includes some primitive elements which represent the requirement of both time and space.
- This language 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 or cloud computing.
- We will give one case and its formal semantics such as operational semantics.

Physical Situation

Interaction Between gate and train.

- Train:

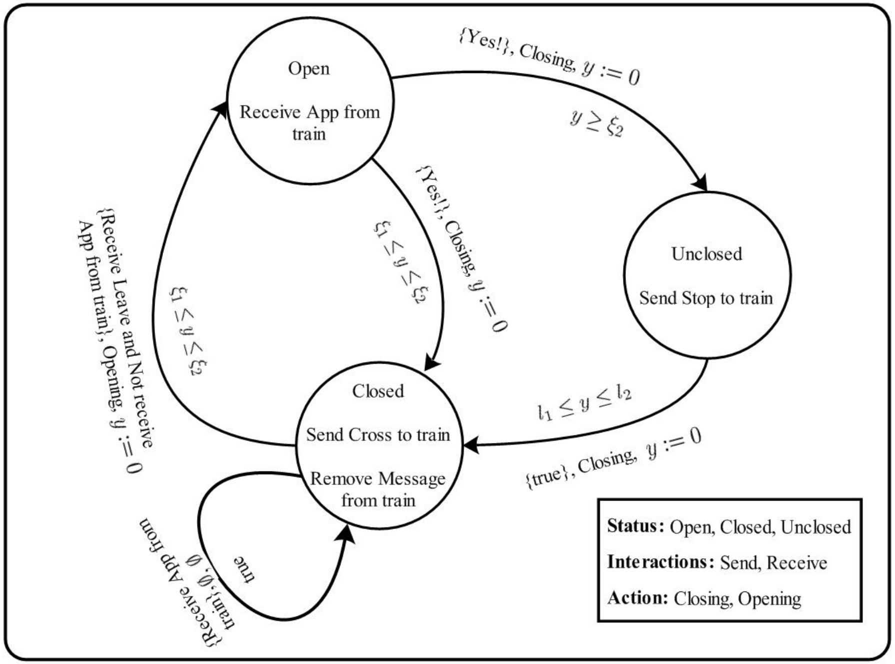

Lleave
Send leave to Gate
{Yes!},Cross, $x := 0$
$\delta_1 \leq x \leq \delta_2$
Lpass
Receive Cross from Gate
{true},Approach, $x := 0$
$\rho_1 \leq x \leq \rho_2$
Lapp
Send App to Gate
{Yes!},Cross, $x := 0$
$\theta_1 \leq x \leq \theta_2$
$\sigma_1 \leq x \leq \sigma_2$
{No!},Stop, $x := 0$
Lstop
Receive Cross from Gate
{No!}, $\emptyset, \emptyset, \emptyset$

Locations: LApp, Lpass, Lstop, Lleave

Interactions: Send, Receive

Action/Process: Cross, Approach, Stop

- Gate:



Open
Receive App from train
Unclosed
Send Stop to train
Closed
Send Cross to train
Remove Message from train
{Yes!}, Closing, $y := 0$
$y \geq \xi_2$
{Yes!}, Closing, $y := 0$
$\xi_1 \leq y \leq \xi_2$
$l_1 \leq y \leq l_2$
{true}, Closing, $y := 0$
{Receive Leave and Not receive App from train}, Opening, $y := 0$
$\xi_1 \leq y \leq \xi_2$
true
{Receive App from train}, $\emptyset$, $\emptyset$
Status: Open, Closed, Unclosed
Interactions: Send, Receive
Action: Closing, Opening

Back
Close

Syntax–Atomic Command A

- We stress two basic concepts: location and time, and a property: consistence between time and location.
- The location is an abstract concept. It may be
  - a channel name like CSP, or
  - an IP address,
  - an agent or
  - an actuator or
  - others.
- The time concept here stresses
  - at time t and
  - duration δ.
- The three atomic commands **Send**, **Get**, and **Ask** involve our language.

Syntax–Atomic Command A

$$A \ ::= \ \textbf{send}_{(l,t,l')}(m) \mid \textbf{get}_{(l,t,l')}(m) \mid \textbf{ask}_{(l,t,l')}(m)$$

- The atomic command **send** $_{(l,t,l')}(m)$ means to send a message m at location l to location l' at time t, this message will be denoted as m_l for its resource l.
- The command **get** $_{(l,t,l')}(m)$ receives a message $m_{l'}$ from the location l' at the location l and time t.
- The command $\textbf{ask}_{(l,t,l')}(m_{l'})$ requires whether or not the message from the location l' exists at the location l and time t.

Syntax–Guard B

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

$$B \ ::= \ A \mid \alpha(\delta) \mid S(\delta) \mid \neg B \mid B \wedge B$$

- $\alpha(\delta)$ is a action statement which shows that the action α will go on the δ unit times. So, the running of action goes on the d unit time. If the action $\alpha(\delta)$ finishes within the d unit time then it is true else false.
- $S(\delta)$ is a state which waits with the duration of d unit time. If the state S keeps the d unit time then it takes the value of truth else the value of false.
- if the atomic statement A holds then it takes the truth else the false.

Syntax–Process P

This language **STeC** is defined as follows in syntax.

$$\begin{aligned} P \;::=\; & \textbf{Stop} \mid \textbf{Skip} \mid A \mid \alpha(\delta) \mid S(\delta) \\ & P;P \mid P \parallel P \mid P+P \mid B \rightarrow P \textbf{ Else } P \\ & \mid P \trianglerighteq_{\delta} P \mid P \trianglerighteq (A \rightarrow P) \end{aligned}$$

- 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 and
- $+$ is the choice operator.
- Statement $B \rightarrow P\,\textbf{Else}\,Q$ is the conditional composition, which means if the guard B is true then it behaves like P else like Q.

Syntax–Process P

$$P \ ::= \ \textbf{Stop} \mid \textbf{Skip} \mid A \mid \alpha(\delta) \mid S(\delta)$$

$$P; P \mid P \parallel P \mid P + P \mid B \to P \ \textbf{Else} \ P$$

$$\mid P \trianglerighteq_\delta P \mid P \trianglerighteq (A \to P)$$

- Timeout Events $P \trianglerighteq_\delta Q$: the statement $P \trianglerighteq_\delta Q$ 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 $P \trianglerighteq (A \to Q)$ initially proceeds like P and is interrupted on occurrence of the atomic command A and then process like Q. Here A is an interrupt command.

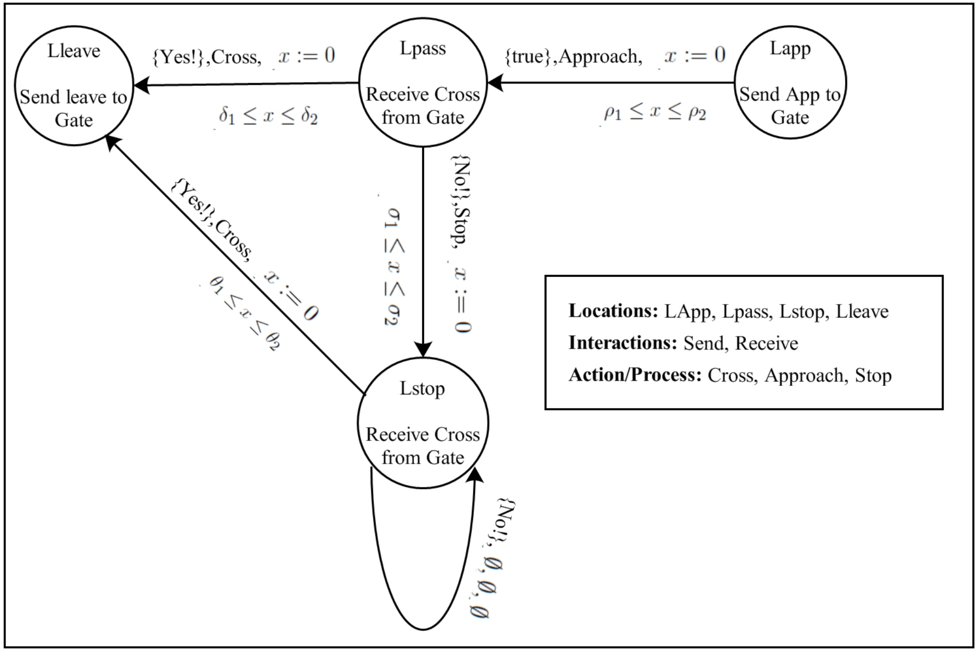
Syntax–Process P

$$\begin{aligned} P \; ::= \; & \mathbf{Stop} \mid \mathbf{Skip} \mid A \mid \alpha(\delta) \mid S(\delta) \\ & P; P \mid P \parallel P \mid P + P \mid B \rightarrow P \;\mathbf{Else}\; P \\ & \mid P \trianglerighteq_\delta P \mid P \trianglerighteq (A \longrightarrow P) \end{aligned}$$

- We can define the sentence $\mathbf{Wait}_{(l,t,l')}(\delta); P$ to be an interrupt sentence: $\mathbf{Stop} \trianglerighteq_\delta P$.
- The command $\mathbf{Wait}_{(l,t,l')}(m_{l'}); P = \mathbf{Stop} \trianglerighteq (\mathbf{Ask}_{(l,t',l')}(m_{l'}) \longrightarrow P)$ with $t \leq t'$, which means to wait for the message coming from l' at location l and time t and then likes P.

Example

Interaction Between gate and train. Train:


Lleave
Send leave to Gate
Lpass
Receive Cross from Gate
Lapp
Send App to Gate
Lstop
Receive Cross from Gate
{Yes!},Cross, $x := 0$
$\delta_1 \leq x \leq \delta_2$
{true},Approach, $x := 0$
$\rho_1 \leq x \leq \rho_2$
{No!},Stop, $x := 0$
$\sigma_1 \leq x \leq \sigma_2$
{Yes!},Cross, $x := 0$
$\theta_1 \leq x \leq \theta_2$
{No!}, $\emptyset, \emptyset, \emptyset$

Locations: LApp, Lpass, Lstop, Lleave

Interactions: Send, Receive

Action/Process: Cross, Approach, Stop

Back

Close

$\mathbf{Send}_{(Lapp,t,Gate)}(App);\ \mathbf{App}(\rho);$

$(\mathbf{Get}_{(Lpass,t+\rho,Gate)}(Pass) \longrightarrow (\mathbf{Passing}(\delta);\ \mathbf{Send}_{(Lleave,t+\rho+\delta,Gate)}(leave))$

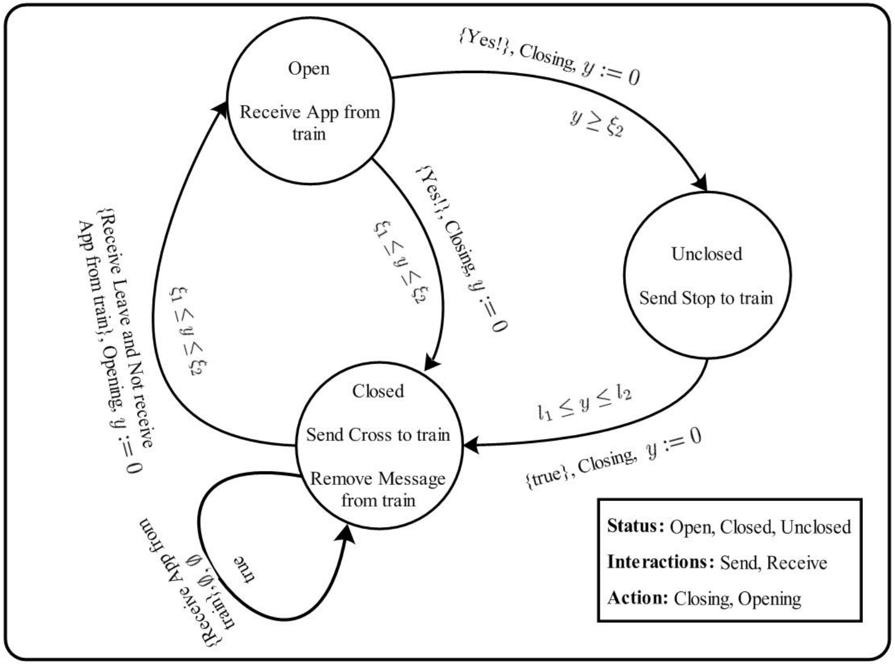
$\mathbf{Else}\ (\mathbf{Stopping}(\tau);\ \mathbf{Wait}_{(Lstop,t+\rho+\tau,Gate)}(Pass);$

$\qquad \mathbf{Pass}(\Delta);\ \mathbf{Send}_{(Lleave,t',Gate)}(leave))$

Example

Interaction Between gate and train.

Gate:



Open
Receive App from train
{Yes!}, Closing, $y := 0$
$y \geq \xi_2$
Unclosed
Send Stop to train
{Yes!}, Closing, $y := 0$
$\xi_1 \leq y < \xi_2$
$l_1 \leq y \leq l_2$
{true}, Closing, $y := 0$
Closed
Send Cross to train
Remove Message from train
{Receive Leave and Not receive App from train}, Opening, $y := 0$
$\xi_1 \leq y < \xi_2$
true
{Receive App from train}, $\emptyset$, $\emptyset$
Status: Open, Closed, Unclosed
Interactions: Send, Receive
Action: Closing, Opening

$$\mathbf{Get}_{(Open,t,train)}(App);\ \mathbf{Closing}(\xi);$$

$$(\mathbf{Closed} \rightarrow (\mathbf{Send}_{(closed,t+\xi,train)}(Cross);\ \mathbf{Closed}(\delta);$$

$$(\mathbf{Wait}_{(closed,t+\xi,train)}(Leave) \wedge \neg\mathbf{Get}_{(closed,t+\xi+\rho+\delta+\epsilon,train)}(App) \rightarrow \mathbf{Open}(\zeta))$$

$$\mathbf{Else}\ (\mathbf{Send}_{(unclosed,t+\xi,train)}(stop);\ \mathbf{Closing};$$

$$(\mathbf{Closed} \rightarrow (\mathbf{Send}_{(closed,d,train)}(Pass);\ \mathbf{Wait}_{(closed,t,train)}(leave)))))$$

Transition System

- Define the set of spatio-temporal stores, **Ststore**, denoted by σ, as the set of multisets of elements of the form m_l, where m_l is a message that comes from location l. We do not care when it is sent.

- Define the set of configurations **Stconf** as **STeC** $\times$ **Ststore** $\times$ **Time**. Configurations are denoted as $(P, \sigma)_u$, where P is a process, σ is a spatio-temporal store of **Ststore** and u is a glob clock.

- We begin by defining the kind of transition system we will use for modelling executions of programs. A spatio-temporal transition system is a subset $\longrightarrow$ of **Stconf**$\times$**Stconf**. An element $((P, \sigma)_u, (P', \sigma')_{u'}) \in\longrightarrow$ is denoted by the notation $(P, \sigma)_u \longrightarrow (P', \sigma')_{u'}$.

Back

Close

Operational Semantics for **STeC**

- The computations in **STeC** may be modeled by a transition system written in Plotkin's style.
- Following the explanation given above, the configurations to be considered consist of an agent together with a multi-set of spatio-temporal store denoting the messages currently available for the computation together with their location and time.
- An operational semantics may be defined directly from them by 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 Action and State

$$\frac{}{(\alpha(\delta), \sigma)_u \longrightarrow (E, \sigma)_{u+\delta}}$$

$$\frac{}{(S(\delta), \sigma)_u \longrightarrow (E, \sigma)_{u+\delta}}$$

This semantics shows that the execution of action consumes δ unit times and state $S(\delta)$ keeps δ unit times.

Back

Close

Operational Semantics for Atomic Commands

$$\frac{a = l, b = t}{(\mathbf{Send}_{(l,t,l')}(m), \sigma)_u \longrightarrow (E, \sigma \cup \{m_l\})_u}$$

$$\frac{a = l, b = t}{(\mathbf{Get}_{(l,t,l')}(m), \sigma \cup \{m_{l'}\})_u \longrightarrow (E, \sigma)_u}$$

$$\frac{a = l, b = t}{(\mathbf{Ask}_{(l,t,l')}(m), \sigma \cup \{m_{l'}\})_u \longrightarrow (E, \sigma \cup \{m_{l'}\})_u}$$

Operational Semantics for **STeC**

$$\frac{}{(\mathbf{Stop}, \sigma)_u \longrightarrow (E, \sigma)_{u'}}$$

$$\frac{}{(\mathbf{Skip}, \sigma)_u \longrightarrow (E, \sigma)_u}$$

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

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

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

Operational Semantics for **STeC**

$$\frac{(P,\sigma)_u \longrightarrow (P',\sigma')_{u'}}{(B \rightarrow P\ \mathbf{Else}\ Q,\sigma)_u \longrightarrow (P',\sigma')_{u'}} \text{ (if } B \text{ is true)}$$

$$\frac{(Q,\sigma)_u \longrightarrow (Q',\sigma')_{u'}}{(B \rightarrow P\ \mathbf{Else}\ Q,\sigma)_u \longrightarrow (Q',\sigma')_{u'}} \text{ (if } B \text{ is false)}$$

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

$$\frac{(P,\sigma)_u \longrightarrow (P',\sigma')_{u'} \wedge (A,\sigma')_{u'} \longrightarrow (E,\sigma'')_{u'}}{(P \trianglerighteq (A \rightarrow Q),\sigma)_u \longrightarrow (Q,\sigma'')_{u'}}$$

Back
Close

Conclusion

- This presentation just gives a premature version of introdution of Spatio-Temporal Consistence language **STeC**.
- There at least exist two lines for a further research:
  - Theoretical Aspect: the denotational semantics and full abstractness
  - Practise Aspect: description of protocol as what Yanwen did.