Design and Analysis of Distributed Interacting Systems

Organizational Matters

Prof. Dr. Joel Greenyer

April 25, 2013
Tutorial today: discussion of the first assignment sheet (which corresponds to the second lecture).
Design and Analysis of Distributed Interacting Systems

Lecture 4 – Linear Temporal Logic

Prof. Dr. Joel Greenyer

April 25, 2013
Last Time
• Environment Analysis
Last Time

• Scenarios help us...
  – **elicit system requirements:**
    • properties whose satisfaction will fully satisfy the customer
    • may not be directly implementable solely by the software
    • if not directly implementable, the software will need to cooperate with the non-software part of the system
  – **capture domain knowledge (or environment assumptions):**
    • properties of the context or the non-software parts of the system
    • often required to help the software fulfill the system requirements
  – **and specify software requirements (or software specification):**
    • properties implementable by the software of the system
    • can be given to a software development team

One more thing

- System requirements: Expressed in terms in actions in the context or shared by the context and the system
One more thing

- There may be different (poss. distributed) software controllers
  - Their interaction is designed later
  - Together they must satisfy the software specification
A system may consist of many hierarchical hardware/software components

- then the same process is repeated on each hierarchy level

• The software specification ($S$) and the domain knowledge ($K$) must be sufficient to guarantee that the system requirements ($R$) are satisfied:

$$S, K \models R$$

• The physical movement of ArmA

K3: If ArmA is ordered to move to the press, it will not arrive at the table (unless it is ordered to move to the table midway).

K4: If ArmA is ordered to move to the table, it will not arrive at the press (unless it is ordered to move to the press midway).
Most properties are either safety or liveness properties.

Safety: Something bad must never happen
- “If ArmA is ordered to move to the press, it will not arrive at the table (unless it is ordered to move to the table midway).”
- “If a blank arrives at the table, ArmA picks up the blank, and moves to the press, the next blank must not arrive before ArmA has returned to the table.”

Liveness: Something good must eventually happen
- “When ArmA releases the blank into the press, the press must press.”
- “If ArmB is ordered to move to the press, it must eventually arrive at the press”

Invariants express that some condition must be true always.—They can be expressed as safety properties: Some condition must never be false.
What to do with System Requirements, Domain Knowledge, Software Specification

• Check that $S, K \models R$ holds
  – if not, we have to rething $S, K$ or $R$
  – checking this manually is error-prone
  – there are tools which allow us to check this automatically
    • properties must be specified formally
    • can take very long or is too complex to compute

• Alternative
  – Create a state-based model of the software, the non-software part of the system, and the context.
  – check whether $S, K$ hold, and ultimately $R$ holds
    • change model or rething $S, K$ or $R$ if not
  – there are tools which allow us to check this automatically
    • again: properties must be specified formally
    • can also take very long or is too complex to compute

model-checking

means: create an operational model (implementation) of the software
Model-Checking

Model Checking + counter example (how the specification can be violated)

false

ture

modify

Model Checking

Model

Specification
Today

• What we didn't finish last time:
  – checking simple properties with LTSA

• Linear Temporal Logic (LTL)
  – you will learn how to express linear-time properties formally
    • syntax
    • interpretation on Kripke Structures
    • equivalences
    • examples

• Next time:
  – Model-Checking LTL formulae
  – Introduction to the model checker SPIN
Safety Properties in the LTSA Tool

• They can be specified as processes that for bad sequences of actions end up in an **ERROR** state

• *Idea*: If composed with other processes, we can check whether a bad sequence of actions is possible or avoided

• The LTSA tool can check this automatically

\[
\text{NO\_SUBSEQ\_A} = (a \rightarrow \text{AOCC}), \\
\text{AOCC} = (b \rightarrow \text{NO\_SUBSEQ\_A} | a \rightarrow \text{ERROR}).
\]

\[
\text{P1} = (a \rightarrow b \rightarrow \text{P1}).
\]

\[
||\text{P1\_NO\_SUBSEQ\_A} = (\text{P1} || \text{NO\_SUBSEQ\_A}).
\]
Safety Properties in the LTSA Tool

- They can be specified as processes that for bad sequences of actions end up in an **ERROR** state
- **Idea**: If composed with other processes, we can check whether a bad sequence of actions is possible or avoided
- The LTSA tool can check this automatically

\[
\text{NO\_SUBSEQ\_A} = (a\rightarrow\text{AOCC}), \\
\text{AOCC} = (b\rightarrow\text{NO\_SUBSEQ\_A}|a\rightarrow\text{ERROR}).
\]

\[
P2 = (a\rightarrow Q), \\
Q = (a\rightarrow Q|b\rightarrow P2).
\]

\[
\text{||P2\_NO\_SUBSEQ\_A} = (P2 \text{|| NO\_SUBSEQ\_A}).
\]

(error-state reachable)
Problem: Composing a process $S$ with another process $P$ that was defined to model a safety property may influence the behavior of that process $S$.

- **transparency**: $P$ should not influence the correct behavior of $S$
- how to ensure this: your homework assignment
Safety Properties in the LTSA Tool

• Instead of specifying what must not happen, it is sometimes more convenient to specify what is required.
• LTSA supports safety properties

    property ARRIVED_AT_TABLE_BEFORE_BLANK_ARRIVES =
    (blankArrives->aMoveToPress->aArrivedAtTable
     ->ARRIVED_AT_TABLE_BEFORE_BLANK_ARRIVES).

• A safety property $\mathcal{P}$ defines a deterministic process that asserts that any trace including actions of the alphabet of $\mathcal{P}$ is accepted by $\mathcal{P}$. 
Safety Properties in the LTSA Tool

• If a safety property $P$ is composed with a process $S$, then all traces of $S$, if removing all actions not in the alphabet of $P$, must also be valid traces of $S$.

• Transparency of safety properties: safety properties do not influence the correct behavior of other processes

$$\text{property } \text{ARRIVED AT TABLE BEFORE BLANK ARRIVES } = \left(\text{blankArrives} \rightarrow \text{aMoveToPress} \rightarrow \text{aArrivedAtTable} \rightarrow \text{ARRIVED AT TABLE BEFORE BLANK ARRIVES}\right).$$
But wait a minute... transparency?

- that is because the property says:
  - “if any of the events blankArrives, aMoveToPress, aArrivedAtTable occurs, then only in the order blankArrives->aMoveToPress->aArrivedAtTable”.

- It does not say “if <something> then <some-other-thing> must not happen”
  - the expressive power of the property construct is limited.
Checking Liveness with LTSA

- LTSA supports simple *progress* properties
  - a certain action must always eventually happen again

- Example: it must be possible that heads can always occur again

\[
\text{progress \ HEADS} = \{\text{heads}\}
\]

LTSA assumes a *fair choice*: if a set of transitions is enabled inf. often, each will be executed inf. often.
Summary: Checking Safety and Liveness with LTSA

• We learned some basic concepts for checking safety and liveness with LTSA
  – they are limited
  – we have to ensure transparency when modeling safety properties “manually”

• Today, we will learn about LTL
  – a logic that allow us to formulate many interesting properties

• LTSA also supports the checking of LTL formulae, but later, we will introduce another model-checking tool.
Model Checking

Model (Kripke structure)

modify

+ counter example (how the specification can be violated)

false

today: LTL

Model Checking

true

Specification (temporal logic)
Linear Temporal Logic (LTL)

• First proposed by Amir Pnueli (1941-2009)
  – Turing Award 1996
    • “for seminal work introducing temporal logic into computing science and for outstanding contributions to program and systems verification.”

• What we will learn about
  – Syntax of LTL
  – Semantics of LTL, interpretation on Kripke Structures
  – Examples and Equivalences
Linear Temporal Logic (LTL)

• also called *Linear-time Temporal Logic*

• *Temporal*... means:
  – the logic allows us to refer to a relative order of events
  – it does not mean that we talk about a precise notion of time
    • not: “the robot arm reaches the press after 2.25 seconds”
  – time can be captured by introducing discrete “ticks” of a clock
  – but continuous time (real-time) cannot be captured

• *Linear*... means:
  – the entities that we consider are whole executions of systems
  – we do not consider *branching*
    • at a point in time, consider different possible futures
    • later we consider the Computational Tree Logic, CTL
LTL and Kripke Structures

- We interpret LTL formulae over Kripke structures
- Remember: A Kripke structure over atomic propositions $AP$ is a tuple $KS = (S, S_0, R, L)$ in which
  - $S$ is a finite, non-empty set of states
  - $S_0 \subseteq S$ is a set of initial states
  - $R \subseteq S \times S$ is a transition relation that must be total
  - $L: S \rightarrow 2^{AP}$ is a function that labels each state with a set of atomic propositions that are true in this state

\[ AP = \{\text{lightOn, savesEnergy} \} \]
From LTS to Kripke Structures

- Atomic propositions represent fundamental properties of states that we are interested in.
- In a Kripke structure, we do not consider what causes the state change.
  - we do not consider actions/events

- What do we do if actions/events are important?
  - we encode them as atomic propositions
    - can be done systematically/automatically; idea:

```
s0 \rightarrow a \rightarrow s1
```

```
s0 \rightarrow s0,a,s1 \rightarrow s1
```

see:
LTL Syntax

• **Def. 1**: Be $AP$ a set of atomic propositions. The set of *LTL formulea over $AP$* is inductively defined as follows
  
  – $p \in AP$ is an LTL formula
  – if $\varphi$ is an LTL formula, so is $\neg \varphi$
  – if $\varphi$ and $\psi$ are LTL formulae, so is $\varphi \lor \psi$
  – if $\varphi$ is an LTL formula, so is $X \varphi$, $G \varphi$, and $F \varphi$
  – if $\varphi$ and $\psi$ are LTL formulae, so are $\varphi U \psi$

• A formula without $X$, $G$, $F$, or $U$ is a *state formula*
Derived Boolean Operators

• We can derive the following Boolean operators:
  
  – $true = p \lor \neg p$
  – $false = \neg true$
  – $\varphi \land \psi = \neg (\neg \varphi \lor \neg \psi)$
  – $\varphi \Rightarrow \psi = \neg \varphi \lor \psi$
  – $\varphi \Leftrightarrow \psi = (\varphi \Rightarrow \psi) \land (\psi \Rightarrow \varphi)$
Semantics (Informally)

• **X** (next)
  \[X \varphi: \varphi \text{ holds in the next state}\]

• **G** (always globally)
  \[G \varphi: \varphi \text{ holds always}\]

• **F** (eventually, in the future, finally)
  \[F \varphi: \varphi \text{ holds at some point in the future}\]

• **U** (until)
  \[\varphi U \psi: \varphi \text{ holds until } \psi \text{ holds (and } \psi \text{ will hold at some point in the future)}\]
Semantics (Informally)

- LTL Formulae are interpreted on the runs of a Kripke structure

- $\models X p$
- $\models G p$
- $\models F p$
- $\models p U q$

an initial state of a Kripke structure
means: $p$ is element of the label of the state
“satisfies”
Semantics: more formally...

- **Def. 2**: Let $M$ be a Kripke Structure and $\varphi$ an LTL formula. Then $M \models \varphi$ iff $\pi \models \varphi$ for all runs $\pi$ of $M$.

- remember: a run is an infinite path starting in a start state
  - $\pi = s_0, s_1, s_2, ...$

- we call $\pi^i$ the $i$-suffix of $\pi$
  - $\pi^i = s_i, s_{i+1}, s_{i+2}, ...$
Semantics: more formally...

- **Def. 3**: Let \( \pi \) be a run. Then \( \pi \models \varphi \) is defined as follows
  - \( \pi \models p, p \in \text{AP}, \text{iff} \ p \in L(s_\varphi) \), i.e., \( p \) holds in the first state of \( \pi \)
  - \( \pi \models \neg \varphi \text{ iff not } \pi \models \varphi \)
  - \( \pi \models \varphi \lor \psi \text{ iff } \pi \models \varphi \text{ or } \pi \models \psi \)
  - \( \pi \models X \varphi \text{ iff } \pi^1 \models \varphi \)
  - \( \pi \models G \varphi \text{ iff } \forall i \geq 0 : \pi^i \models \varphi \)
  - \( \pi \models F \varphi \text{ iff } \exists i \geq 0 : \pi^i \models \varphi \)
  - \( \pi \models \varphi U \psi \text{ iff } \exists k \geq 0 : \pi^k \models \psi \)
    and \( \forall i, 0 \leq i < k : \pi^i \models \varphi \).
Typical properties in LTL

• $p$ is always eventually followed by $q$
  - $G (p \Rightarrow F q)$

• $p$ is always directly followed by $q$
  - $G (p \Rightarrow X q)$

• $p$ will eventually be true forever
  - $F G p$

• $p$ will always be true
  - $G p$

• $p$ will be true infinitely often
  - $G F p$  ($p$ will always eventually be true)
Simple Example: Coffee Machine

- \( AP = \{\text{coffee\_chosen, tea\_chosen, money\_inserted, coffee\_delivered, tea\_delivered}\} \)

- Once in a while someone chooses tea or coffee
  - \( G \ F (\text{coffee\_chosen} \lor \text{tea\_chosen}) \)

- If coffee is chosen and next money is inserted, coffee will be delivered
  - \( G ((\text{coffee\_chosen} \land \ X \text{money\_inserted}) \Rightarrow F \text{coffee\_delivered}) \)

- when coffee has been chosen, tea will not be delivered until tea is chosen
  - \( G (\text{coffee\_chosen} \Rightarrow (\neg \text{tea\_delivered} \ U \text{tea\_chosen})) \)