Reactive Systems: Modelling, Specification and Verification

EWSCS'07–Lecture 5

- Weak bisimilarity (reprise) and weak bisimulation games
- Properties of weak bisimilarity
- Example: a communication protocol and its modelling in CCS
- Concurrency workbench (CWB)
- An introduction to Hennessy-Milner logic (HML)
- Syntax and semantics of HML

Weak Transition Relation

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS such that $\tau \in Act$.

Definition of the Weak Transition Relations

Let a be an action or ε:

$$\stackrel{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^* & \text{if } a \neq \varepsilon \\ (\xrightarrow{\tau})^* & \text{if } a = \varepsilon \end{cases}$$

Definition

If a is an observable action, then $\hat{a} = a$. On the other hand, $\hat{\tau} = \varepsilon$.

Weak Bisimilarity

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS such that $\tau \in Act$.

Weak Bisimulation

A binary relation $R \subseteq Proc \times Proc$ is a weak bisimulation iff whenever $(s, t) \in R$ then for each $a \in Act$ (including τ):

- if $s \xrightarrow{a} s'$ then $t \stackrel{\hat{a}}{\Longrightarrow} t'$ for some t' such that $(s', t') \in R$
- if $t \xrightarrow{a} t'$ then $s \stackrel{\hat{a}}{\Longrightarrow} s'$ for some s' such that $(s', t') \in R$.

Weak Bisimilarity

Two processes $p_1, p_2 \in Proc$ are weakly bisimilar ($p_1 \approx p_2$) if and only if there exists a weak bisimulation R such that $(p_1, p_2) \in R$.

$$\approx \; = \bigcup \{R \mid R \text{ is a weak bisimulation}\}$$

Weak Bisimulation Game

Definition

Same as for the strong bisimulation game except that

- defender can now answer using $\stackrel{a}{\Longrightarrow}$ moves.

The attacker is still using only $\stackrel{a}{\longrightarrow}$ moves.

Let's play!

Theorem

- States s and t are weakly bisimilar if and only if the defender has a universal winning strategy starting from the configuration (s, t).
- States s and t are not weakly bisimilar if and only if the attacker has a universal winning strategy starting from the configuration (s, t).

Weak Bisimulation Game

Definition

Same as for the strong bisimulation game except that

- defender can now answer using $\stackrel{a}{\Longrightarrow}$ moves.

The attacker is still using only $\stackrel{a}{\longrightarrow}$ moves.

Let's play!

Theorem

- States s and t are weakly bisimilar if and only if the defender has a universal winning strategy starting from the configuration (s, t).
- States s and t are not weakly bisimilar if and only if the attacker has a universal winning strategy starting from the configuration (s, t).

Weak Bisimilarity – Properties

Properties of $\approx$

- an equivalence relation
- the largest weak bisimulation
- validates lots of natural laws, e.g.
  - $a.\tau.P \approx a.P$
  - $P + \tau.P \approx \tau.P$
  - $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
  - $P + Q \approx Q + P \qquad P|Q \approx Q|P \qquad P + Nil \approx P \qquad \ldots$
- strong bisimilarity is included in weak bisimilarity ($\sim \subseteq \approx$)
- abstracts from τ loops

Case Study: Communication Protocol

$$\begin{aligned} \mathsf{Send} &\stackrel{\mathrm{def}}{=} \mathsf{acc.Sending} & \mathsf{Rec} &\stackrel{\mathrm{def}}{=} \mathsf{trans.Del} \\ \mathsf{Sending} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{send}}.\mathsf{Wait} & \mathsf{Del} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{del}}.\mathsf{Ack} \\ \mathsf{Wait} &\stackrel{\mathrm{def}}{=} \mathsf{ack.Send} + \mathsf{error.Sending} & \mathsf{Ack} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{ack}}.\mathsf{Rec} \end{aligned}$$

$$\begin{aligned} \mathsf{Med} &\stackrel{\mathrm{def}}{=} \mathsf{send.Med}' \\ \mathsf{Med}' &\stackrel{\mathrm{def}}{=} \tau.\mathsf{Err} + \overline{\mathsf{trans}}.\mathsf{Med} \\ \mathsf{Err} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{error}}.\mathsf{Med} \end{aligned}$$

Case Study: Communication Protocol

$$\begin{aligned}
\mathsf{Send} &\stackrel{\mathrm{def}}{=} \mathsf{acc}.\mathsf{Sending} \\
\mathsf{Sending} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{send}}.\mathsf{Wait} \\
\mathsf{Wait} &\stackrel{\mathrm{def}}{=} \mathsf{ack}.\mathsf{Send} + \mathsf{error}.\mathsf{Sending}
\end{aligned}
\qquad
\begin{aligned}
\mathsf{Rec} &\stackrel{\mathrm{def}}{=} \mathsf{trans}.\mathsf{Del} \\
\mathsf{Del} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{del}}.\mathsf{Ack} \\
\mathsf{Ack} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{ack}}.\mathsf{Rec}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Med} &\stackrel{\mathrm{def}}{=} \mathsf{send}.\mathsf{Med}' \\
\mathsf{Med}' &\stackrel{\mathrm{def}}{=} \tau.\mathsf{Err} + \overline{\mathsf{trans}}.\mathsf{Med} \\
\mathsf{Err} &\stackrel{\mathrm{def}}{=} \overline{\mathsf{error}}.\mathsf{Med}
\end{aligned}$$

Verification Question

$$\mathsf{Impl} \stackrel{\mathrm{def}}{=} (\mathsf{Send} \,|\, \mathsf{Med} \,|\, \mathsf{Rec}) \smallsetminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$$

$$\mathsf{Spec} \stackrel{\mathrm{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$$

Question

$$\mathsf{Impl} \stackrel{?}{\approx} \mathsf{Spec}$$

1. Draw the LTS of Impl and Spec and prove (by hand) the equivalence.
2. Use the Concurrency WorkBench (CWB).

Verification Question

$$\mathrm{Impl} \stackrel{\mathrm{def}}{=} (\mathrm{Send} \mid \mathrm{Med} \mid \mathrm{Rec}) \smallsetminus \{\mathrm{send}, \mathrm{trans}, \mathrm{ack}, \mathrm{error}\}$$

$$\mathrm{Spec} \stackrel{\mathrm{def}}{=} \mathrm{acc}.\overline{\mathrm{del}}.\mathrm{Spec}$$

Question

$$\mathrm{Impl} \stackrel{?}{\approx} \mathrm{Spec}$$

1. Draw the LTS of Impl and Spec and prove (by hand) the equivalence.
2. Use the Concurrency WorkBench (CWB).

Verification Question

$$\mathsf{Impl} \stackrel{\mathrm{def}}{=} (\mathsf{Send} \mid \mathsf{Med} \mid \mathsf{Rec}) \smallsetminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$$

$$\mathsf{Spec} \stackrel{\mathrm{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$$

Question

$$\mathsf{Impl} \stackrel{?}{\approx} \mathsf{Spec}$$

1. Draw the LTS of Impl and Spec and prove (by hand) the equivalence.
2. Use the Concurrency WorkBench (CWB).

Verification Question

$$\mathsf{Impl} \stackrel{\text{def}}{=} (\mathsf{Send} \mid \mathsf{Med} \mid \mathsf{Rec}) \smallsetminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$$

$$\mathsf{Spec} \stackrel{\text{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$$

Question

$$\mathsf{Impl} \stackrel{?}{\approx} \mathsf{Spec}$$

1. Draw the LTS of Impl and Spec and prove (by hand) the equivalence.
2. Use the Concurrency WorkBench (CWB).

Verification Question

$$\mathsf{Impl} \stackrel{\text{def}}{=} (\mathsf{Send} \mid \mathsf{Med} \mid \mathsf{Rec}) \setminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$$

$$\mathsf{Spec} \stackrel{\text{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$$

Question

$$\mathsf{Impl} \stackrel{?}{\approx} \mathsf{Spec}$$

1. Draw the LTS of Impl and Spec and prove (by hand) the equivalence.
2. Use the Concurrency WorkBench (CWB).

CCS Expressions in CWB

CCS Definitions

$\mathsf{Med} \stackrel{\mathrm{def}}{=} \mathsf{send}.\mathsf{Med}'$
$\mathsf{Med}' \stackrel{\mathrm{def}}{=} \tau.\mathsf{Err} + \overline{\mathsf{trans}}.\mathsf{Med}$
$\mathsf{Err} \stackrel{\mathrm{def}}{=} \overline{\mathsf{error}}.\mathsf{Med}$
$\vdots$
$\mathsf{Impl} \stackrel{\mathrm{def}}{=} (\mathsf{Send} \mid \mathsf{Med} \mid \mathsf{Rec}) \setminus \{\mathsf{send}, \mathsf{trans}, \mathsf{ack}, \mathsf{error}\}$

$\mathsf{Spec} \stackrel{\mathrm{def}}{=} \mathsf{acc}.\overline{\mathsf{del}}.\mathsf{Spec}$

CWB Program (protocol.cwb)

```
agent Med = send.Med';
agent Med' = (tau.Err + 'trans.Med);
agent Err = 'error.Med;
:
set L = {send, trans, ack, error};
agent Impl = (Send | Med | Rec) \ L;

agent Spec = acc.'del.Spec;
```

CWB Session

```
[luca@vel5638 CWB]$ ./xccscwb.x86-linux

> help;

> input "protocol.cwb";

> vs(5,Impl);

> sim(Spec);

> eq(Spec,Impl);               ** weak bisimilarity **

> strongeq(Spec,Impl);         ** strong bisimilarity **
```

Is Weak Bisimilarity a Congruence for CCS?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $P[f] \approx Q[f]$ *for each relabelling function* f
- $P \setminus L \approx Q \setminus L$ *for each set of labels* L.

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Is Weak Bisimilarity a Congruence for CCS?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process R*
- $P[f] \approx Q[f]$ *for each relabelling function f*
- $P \setminus L \approx Q \setminus L$ *for each set of labels L.*

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Is Weak Bisimilarity a Congruence for CCS?

Theorem

Let P and Q be CCS processes such that $P \approx Q$. Then

- $\alpha.P \approx \alpha.Q$ *for each action* $\alpha \in Act$
- $P \mid R \approx Q \mid R$ *and* $R \mid P \approx R \mid Q$ *for each CCS process* R
- $P[f] \approx Q[f]$ *for each relabelling function* f
- $P \setminus L \approx Q \setminus L$ *for each set of labels* L.

What about choice?

$\tau.a.Nil \approx a.Nil$ but $\tau.a.Nil + b.Nil \not\approx a.Nil + b.Nil$

Conclusion

Weak bisimilarity is not a congruence for CCS.

Verifying Correctness of Reactive Systems

Let *Impl* be an implementation of a system (e.g. in CCS syntax).

Equivalence Checking Approach

$$\mathit{Impl} \equiv \mathit{Spec}$$

- $\equiv$ is an abstract equivalence, e.g. $\sim$ or $\approx$
- *Spec* is often expressed in the same language as *Impl*
- *Spec* provides the full specification of the intended behaviour

Model Checking Approach

$$\mathit{Impl} \models \mathit{Property}$$

- $\models$ is the satisfaction relation
- *Property* is a particular feature, often expressed via a logic
- *Property* is a partial specification of the intended behaviour

Verifying Correctness of Reactive Systems

Let *Impl* be an implementation of a system (e.g. in CCS syntax).

Equivalence Checking Approach

$$Impl \equiv Spec$$

- $\equiv$ is an abstract equivalence, e.g. $\sim$ or $\approx$
- *Spec* is often expressed in the same language as *Impl*
- *Spec* provides the full specification of the intended behaviour

Model Checking Approach

$$Impl \models Property$$

- $\models$ is the satisfaction relation
- *Property* is a particular feature, often expressed via a logic
- *Property* is a partial specification of the intended behaviour

Our Aim

Develop a logic in which we can express interesting properties of reactive systems.

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Logical Properties of Reactive Systems

Modal Properties – what can happen now (possibility, necessity)

- drink a coffee (can drink a coffee now)
- does not drink tea
- drinks both tea and coffee
- drinks tea after coffee

Temporal Properties – behaviour in time

- never drinks any alcohol
  (safety property: nothing bad can happen)
- eventually will have a glass of wine
  (liveness property: something good will happen)

Can these properties be expressed using equivalence checking?

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \;::=\; tt \;\mid\; ff \;\mid\; F \wedge G \;\mid\; F \vee G \;\mid\; \langle a \rangle F \;\mid\; [a]F$$

Intuition:

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ there is at least one a-successor that satisfies F
- $[a]F$ all a-successors have to satisfy F

Remark

Temporal properties like *always/never in the future* or *eventually* are not included.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

Intuition:

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ there is at least one a-successor that satisfies F
- $[a]F$ all a-successors have to satisfy F

Remark

Temporal properties like *always/never in the future* or *eventually* are not included.

Hennessy-Milner Logic – Syntax

Syntax of the Formulae ($a \in Act$)

$$F, G \ ::= \ tt \ | \ ff \ | \ F \wedge G \ | \ F \vee G \ | \ \langle a \rangle F \ | \ [a]F$$

Intuition:

- tt all processes satisfy this property
- ff no process satisfies this property
- $\wedge$, $\vee$ usual logical AND and OR
- $\langle a \rangle F$ there is at least one a-successor that satisfies F
- $[a]F$ all a-successors have to satisfy F

Remark

Temporal properties like *always/never in the future* or *eventually* are not included.

Hennessy-Milner Logic – Semantics

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS.

Validity of the logical triple $p \models F$ ($p \in Proc$, F a HM formula)

$$
\begin{array}{rl}
p \models tt & \text{for each } p \in Proc \\
p \models f\!f & \text{for no } p \text{ (we also write } p \not\models f\!f) \\
p \models F \wedge G & \text{iff } p \models F \text{ and } p \models G \\
p \models F \vee G & \text{iff } p \models F \text{ or } p \models G \\
p \models \langle a \rangle F & \text{iff } p \xrightarrow{a} p' \text{ for some } p' \in Proc \text{ such that } p' \models F \\
p \models [a]F & \text{iff } p' \models F, \text{ for all } p' \in Proc \text{ such that } p \xrightarrow{a} p'
\end{array}
$$

We write $p \not\models F$ whenever p does not satisfy F.

What about Negation?

For every formula F we define the formula F^c as follows:

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$

Theorem (F^c is equivalent to the negation of F)

For any $p \in Proc$ and any HM formula F

1. $p \models F \implies p \not\models F^c$
2. $p \not\models F \implies p \models F^c$

What about Negation?

For every formula F we define the formula F^c as follows:

- $tt^c = ff$
- $ff^c = tt$
- $(F \wedge G)^c = F^c \vee G^c$
- $(F \vee G)^c = F^c \wedge G^c$
- $(\langle a \rangle F)^c = [a]F^c$
- $([a]F)^c = \langle a \rangle F^c$

Theorem (F^c is equivalent to the negation of F)

For any $p \in Proc$ and any HM formula F

1. $p \models F \Longrightarrow p \not\models F^c$
2. $p \not\models F \Longrightarrow p \models F^c$