Reactive Systems: Modelling, Specification and Verification

EWSCS'07–Lecture 6

- Hennessy-Milner logic
- Syntax and semantics
- Correspondence with strong bisimilarity
- Examples in CWB

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

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

Model Checking of Reactive Systems

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 ($a \in Act$)

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

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 – Syntax

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

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

Intuition:

- tt all processes satisfy this property
- $f\!f$ 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.

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)

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

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

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

Hennessy-Milner Logic – Denotational Semantics

For a formula F let $[\![F]\!] \subseteq \mathit{Proc}$ contain all states that satisfy F.

Denotational Semantics: $[\![\_]\!] : \mathit{Formulae} \rightarrow 2^{\mathit{Proc}}$

- $[\![\mathit{tt}]\!] = \mathit{Proc}$
- $[\![\mathit{ff}]\!] = \emptyset$
- $[\![F \vee G]\!] = [\![F]\!] \cup [\![G]\!]$
- $[\![F \wedge G]\!] = [\![F]\!] \cap [\![G]\!]$
- $[\![\langle a \rangle F]\!] = \langle \cdot a \cdot \rangle [\![F]\!]$
- $[\![[a]F]\!] = [\cdot a \cdot][\![F]\!]$

where $\langle \cdot a \cdot \rangle, [\cdot a \cdot] : 2^{(\mathit{Proc})} \rightarrow 2^{(\mathit{Proc})}$ are defined by

$$\langle \cdot a \cdot \rangle S = \{p \in \mathit{Proc} \mid \exists p'.\ p \xrightarrow{a} p' \text{ and } p' \in S\}$$

$$[\cdot a \cdot] S = \{p \in \mathit{Proc} \mid \forall p'.\ p \xrightarrow{a} p' \implies p' \in S\}.$$

The Correspondence Theorem

Theorem

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ *be an LTS,* $p \in Proc$ *and* F *a formula of Hennessy-Milner logic. Then*

$$p \models F \quad \textit{if and only if} \quad p \in [\![F]\!].$$

Proof: by structural induction on the structure of the formula F.

The Correspondence Theorem

Theorem

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS, $p \in Proc$ and F a formula of Hennessy-Milner logic. Then

$$p \models F \quad \textit{if and only if} \quad p \in [\![F]\!].$$

Proof: by structural induction on the structure of the formula F.

Image-Finite Labelled Transition System

Image-Finite System

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an LTS. We call it image-finite iff for every $p \in Proc$ and every $a \in Act$ the set

$$\{p' \in Proc \mid p \xrightarrow{a} p'\}$$

is finite.

Theorem (Hennessy-Milner)

Let $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$ be an image-finite LTS and $p, q \in St$. Then

$$p \sim q$$

if and only if

for every HM formula F: $(p \models F \iff q \models F)$.

CWB Session

hm.cwb

```
agent S = a.S1;
agent S1 = b.0 + c.0;

agent T = a.T1 + a.T2;
agent T1 = b.0;
agent T2 = c.0;
```

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

> input "hm.cwb";
> print;
> help logic;
> checkprop(S,<a>(<b>T & <c>T));
 true
> checkprop(T,<a>(<b>T & <c>T));
 false
> help dfstrong;
> dfstrong(S,T);
 [a]<b>T
> exit;
```