

2IW05 – Software Specification

Consistency and Modal Logic

Mohammad Mousavi

Formal Systems Analysis (FSA)
Model-Driven Software Engineering
Department of Computer Science
TU/Eindhoven

December 13, 2011

Consistency and Completeness

- All use-cases in the use-case diagram and its description should have a description.

Consistency and Completeness

- All use-cases in the use-case diagram and its description should have a description.
- All methods in the class diagram should have an Alloy specification; relationships should be reflected properly in Alloy.

Consistency and Completeness

- All use-cases in the use-case diagram and its description should have a description.
- All methods in the class diagram should have an Alloy specification; relationships should be reflected properly in Alloy.
- All use-cases should have an SD specification.
- Objects in SDs should be from classes defined in CDs (or actors in the use-case diagram).
- Messages sent to objects in SDs should be methods of the target class in the CDs.

Consistency and Completeness

- All use-cases in the use-case diagram and its description should have a description.
- All methods in the class diagram should have an Alloy specification; relationships should be reflected properly in Alloy.
- All use-cases should have an SD specification.
- Objects in SDs should be from classes defined in CDs (or actors in the use-case diagram).
- Messages sent to objects in SDs should be methods of the target class in the CDs.
- Each class (combination of classes) should have a Statecharts description.
- Each scenario specified by a usecase (an SD) should be among the traces of the SC description.

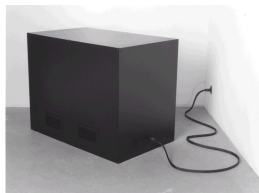
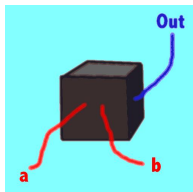
Outline

1 Temporal logic

2 Hennessy-Milner logic

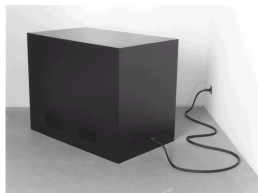
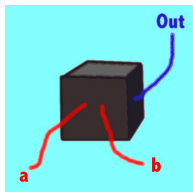
Specification using Temporal logic

- Fix observable events (interactions with external world)



Specification using Temporal logic

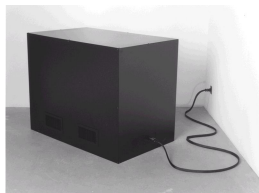
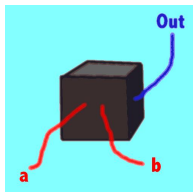
- Fix observable events (interactions with external world)



- Describe temporal properties using these

Specification using Temporal logic

- Fix observable events (interactions with external world)



- Describe temporal properties using these
- Verify the correctness of the properties with respect to some labeled transition system (obtained via MSC, Statechart, process algebra specification, Petri net, etc.)

A computer scientist interacts with its environment

- coffee for taking coffee in
- coin for producing a coin
- pub for producing a publication
- ...

A computer scientist interacts with its environment

- coffee for taking coffee in
- coin for producing a coin
- pub for producing a publication
- ...

Properties of interest

- the computer scientist is not willing to drink coffee now

A computer scientist interacts with its environment

- coffee for taking coffee in
- coin for producing a coin
- pub for producing a publication
- ...

Properties of interest

- the computer scientist is not willing to drink coffee now
- the computer scientist is willing to drink both coffee and tea now

A computer scientist interacts with its environment

- coffee for taking coffee in
- coin for producing a coin
- pub for producing a publication
- ...

Properties of interest

- the computer scientist is not willing to drink coffee now
- the computer scientist is willing to drink both coffee and tea now
- the computer scientist is willing to drink coffee, but not tea, now

A computer scientist interacts with its environment

- coffee for taking coffee in
- coin for producing a coin
- pub for producing a publication
- ...

Properties of interest

- the computer scientist is not willing to drink coffee now
- the computer scientist is willing to drink both coffee and tea now
- the computer scientist is willing to drink coffee, but not tea, now
- always produces a publication after drinking coffee

Outline

1 Temporal logic

2 Hennessy-Milner logic

Hennessy-Milner logic

- Introduced by Hennessy and Milner in 1985



- Matthew Hennessy and Robin Milner, On Observing Nondeterminism and Concurrency, ICALP, LNCS 85, pp. 299-309, 1980.
- Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. Journal of the ACM, 32(1):137-161, 1985.

Syntax of Hennessy-Milner logic

for $a \in Act$

$$F ::= true \mid false \mid F \wedge F \mid F \vee F \mid \langle a \rangle F \mid [a]F$$

Syntax of Hennessy-Milner logic

for $a \in Act$

$$F ::= true \mid false \mid F \wedge F \mid F \vee F \mid \langle a \rangle F \mid [a]F$$

where

- $\langle a \rangle F$ denotes that it is possible to perform action a and thereby (in the next state) satisfy F

Syntax of Hennessy-Milner logic

for $a \in Act$

$$F ::= true \mid false \mid F \wedge F \mid F \vee F \mid \langle a \rangle F \mid [a]F$$

where

- $\langle a \rangle F$ denotes that it is possible to perform action a and thereby (in the next state) satisfy F
- $[a]F$ denotes that no matter how a process performs action a afterwards necessarily F holds

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = \text{false}$

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = false$
- $[A] F$ denotes $[a_1] F \wedge \dots \wedge [a_n] F$ and $[\emptyset] F = true$

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = false$
- $[A]F$ denotes $[a_1]F \wedge \dots \wedge [a_n]F$ and $[\emptyset]F = true$

Negation

- $\neg true$ denotes *false*
- $\neg false$ denotes *true*

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = false$
- $[A]F$ denotes $[a_1]F \wedge \dots \wedge [a_n]F$ and $[\emptyset]F = true$

Negation

- $\neg true$ denotes *false*
- $\neg false$ denotes *true*
- $\neg(F \wedge G)$ denotes $\neg F \vee \neg G$
- $\neg(F \vee G)$ denotes $\neg F \wedge \neg G$

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = false$
- $[A]F$ denotes $[a_1]F \wedge \dots \wedge [a_n]F$ and $[\emptyset]F = true$

Negation

- $\neg true$ denotes *false*
- $\neg false$ denotes *true*
- $\neg(F \wedge G)$ denotes $\neg F \vee \neg G$
- $\neg(F \vee G)$ denotes $\neg F \wedge \neg G$
- $\neg\langle a \rangle F$ denotes $[a]\neg F$
- $\neg[a]F$ denotes $\langle a \rangle\neg F$

Syntax of Hennessy-Milner logic

For $A = \{a_1, \dots, a_n\} \subseteq Act$ with $n \geq 1$

- $\langle A \rangle F$ denotes $\langle a_1 \rangle F \vee \dots \vee \langle a_n \rangle F$ and $\langle \emptyset \rangle F = false$
- $[A]F$ denotes $[a_1]F \wedge \dots \wedge [a_n]F$ and $[\emptyset]F = true$

Negation

- $\neg true$ denotes *false*
- $\neg false$ denotes *true*
- $\neg(F \wedge G)$ denotes $\neg F \vee \neg G$
- $\neg(F \vee G)$ denotes $\neg F \wedge \neg G$
- $\neg\langle a \rangle F$ denotes $[a]\neg F$
- $\neg[a]F$ denotes $\langle a \rangle\neg F$

We will use negation freely!

Examples

- the computer scientist is not willing to drink coffee now

Examples

- the computer scientist is not willing to drink coffee now

$\neg(\text{coffee})$ *true* or $[\text{coffee}]$ *false*

Examples

- the computer scientist is not willing to drink coffee now

$\neg(\text{coffee})$ *true* or $[\text{coffee}]$ *false*

- the computer scientist is willing to drink both coffee and tea now

Examples

- the computer scientist is not willing to drink coffee now

$\neg \langle \text{coffee} \rangle \text{true}$ or $[\text{coffee}] \text{false}$

- the computer scientist is willing to drink both coffee and tea now

$\langle \text{coffee} \rangle \text{true} \wedge \langle \text{tea} \rangle \text{true}$

Examples

- the computer scientist is not willing to drink coffee now

$\neg \langle \text{coffee} \rangle \text{true}$ or $[\text{coffee}] \text{false}$

- the computer scientist is willing to drink both coffee and tea now

$\langle \text{coffee} \rangle \text{true} \wedge \langle \text{tea} \rangle \text{true}$

- the computer scientist is willing to drink coffee, but not tea, now

Examples

- the computer scientist is not willing to drink coffee now

$\neg\langle\text{coffee}\rangle\text{true}$ or $[\text{coffee}]\text{false}$

- the computer scientist is willing to drink both coffee and tea now

$\langle\text{coffee}\rangle\text{true} \wedge \langle\text{tea}\rangle\text{true}$

- the computer scientist is willing to drink coffee, but not tea, now

$\langle\text{coffee}\rangle\text{true} \wedge \neg\langle\text{tea}\rangle\text{true}$

Examples

- the computer scientist can always drink tea immediately after having drunk two coffees in a row

Examples

- the computer scientist can always drink tea immediately after having drunk two coffees in a row

`[coffee][coffee]<tea>true`

Typical formulas

- the process is deadlocked

$[Act]false$

- the process can execute some action

- a must happen next

- F holds after one step

Typical formulas

- the process is deadlocked

$[Act]false$

- the process can execute some action

$\langle Act \rangle true$

- a must happen next

- F holds after one step

Typical formulas

- the process is deadlocked

$[Act]false$

- the process can execute some action

$\langle Act \rangle true$

- a must happen next

$\langle a \rangle true \wedge [Act \setminus \{a\}] false$

$\langle Act \rangle true \wedge [Act \setminus \{a\}] false$

- F holds after one step

Typical formulas

- the process is deadlocked

$[Act]false$

- the process can execute some action

$\langle Act \rangle true$

- a must happen next

$\langle a \rangle true \wedge [Act \setminus \{a\}] false$

$\langle Act \rangle true \wedge [Act \setminus \{a\}] false$

- F holds after one step

$[Act]F \wedge \langle Act \rangle true$

Material

- Chapters 5 and 6 of book 'Reactive Systems – Modelling, Specification and Verification' by L. Aceto, A. Ingólfssdóttir, K. Larsen and J. Srba (chapters are online; the whole book is also recommended)