multi-modal logic

given a set of labels $I$

for every label $i$ there is a modality $\langle i \rangle$

semantics makes use of a $I$-frame $(W, \{R_i \mid i \in I\})$ with

$W \neq \emptyset$ a set of worlds or states, and $R_i \subseteq W \times W$ for every $i \in I$

$\mathcal{M}, w \models \langle i \rangle \phi$ iff $\mathcal{M}, v \models \phi$ for some $v$ with $R_i v w$

multi-modal logic: bisimulation

we use one index set $I$

let $\mathcal{M} = (W, \{R_i \mid i \in I\}, V)$ and $\mathcal{M}' = (W', \{R'_i \mid i \in I\}, V')$ be $I$-models

$\emptyset \neq Z \subseteq W \times W'$ is a bisimulation if for every $(w, w') \in Z$

$w \in V(p)$ if and only if $w' \in V'(p)$

if $R_i w v$ then there is $v'$ with $R'_i w' v'$ and $(v, v') \in Z$ (same $i$)

if $R'_i w' v'$ then there is $v$ with $R_i w v$ and $(v, v') \in Z$ (same $i$)
bisimulation and modal equivalence

if two worlds are bisimilar, then they are modally equivalent

we work with multi-modal formulas and $\mathcal{I}$-models

modal equivalence and bisimulation

under the restriction that every $R_i$ is finitely branching

(which does not imply that the model is finitely branching)

two worlds are bisimilar if and only if they are modally equivalent

theorem by Hennessy and Milner (1985)

instances of multi-modal logic

we already know various instances of multi-modal logic:

$I = \emptyset$ gives propositional logic

$I = \{0\}$ gives basic modal logic

$I = \{F, P\}$ gives temporal logic with temporal model $R_F = R_P^{-1}$

$I = \text{Prog}(A)$ gives propositional dynamic logic with PDL-model

exercises

until not definable in basic modal logic in temporal frames

until not definable in temporal modal logic in temporal frames

next not definable in basic modal logic in temporal frames

define discrete in temporal logic

use next to define discrete in temporal logic

define right-linearity in temporal logic (similar to ex in exercise class 4)
overview

- towards propositional dynamic logic
- propositional dynamic logic

program verification

prove that a program meets its specification

for example:

input: finite list of integers
program: sort
output: sorted permutation of the input

example: gcd program

\[
\text{while } y \neq 0 \text{ do} \\
\text{begin} \\
\quad z := x \mod y; \\
\quad x := y; \\
\quad y := z; \\
\text{end} \\
\text{return } x
\]

\(x\) is an input variable and an output variable
\(y\) is an input variable
\(z\) is a work variable

example: run of gcd program

a trace or a run is a sequence of states

in the gcd example:

a state is a tuple of values of \(x, y, z\); example of a run:

\[(15, 27, 0) \rightarrow (15, 27, 15) \rightarrow (27, 27, 15) \rightarrow (27, 15, 15) \rightarrow (27, 15, 12) \rightarrow (15, 12, 12) \rightarrow (15, 12, 3) \rightarrow (12, 3, 3) \rightarrow (3, 3, 0) \rightarrow (3, 0, 0)\]
example: correctness of gcd program

pre:
\(x, y \in \{0, 1, 2, \ldots\} \text{ and } y \neq 0 \land c = x \land d = y\)

during:
\(gcd(x, y) = gcd(c, d)\) is an invariant

post:
\(y = 0 \land x = gcd(c, d)\)

program verification: approach by Hoare

prove statements of the form \(\{\text{precondition}\} \text{program} \{\text{postcondition}\}\)
the precondition and postcondition are formulas
the program is a while-program
built from sequential composition, conditional, while, assignment
we have proof rules for showing \(\{\phi\} \alpha \{\psi\}\)

Tony Hoare

Turing Award 1980
quicksort,
Hoare Logic following work by Floyd and Turing,
Communicating Sequential Processes (CSP)

Alan Turing

the work on program verification by Tony Hoare
actually goes back to work by Turing
towards program verification using modal logic

a state of a program execution is a state or world

a program is a regular program which slightly generalizes while program

a statement \{\text{pre}\} \text{program} \{\text{post}\} is a formula \text{pre} \rightarrow [\text{program}]\text{post}

propositional dynamic logic (PDL): starting point

for every program \(\alpha\) we have a modality \(\langle\alpha\rangle\)

\(\langle\alpha\rangle\phi\) intuitively means
it is possible to execute \(\alpha\) starting in the current state,
and halt (successfully) in a state satisfying \(\phi\)

\([\alpha]\phi\) intuitively means
if \(\alpha\) halts (successfully), then it halts in a state satisfying \(\phi\)

set Prog of PDL (or regular) programs: definition

atomic program
\(a\) from a set \(A\) of atomic programs

sequential composition
\(\alpha;\beta\)

non-deterministic choice
\(\alpha\cup\beta\)

iteration
\(\alpha^*\)

test
\(\phi?\) with \(\phi\) a formula, so depends on the grammar for formulas
PDL programs: intuitive meaning

atomic, indecomposable, step

\( \phi \)?
if \( \phi \) then skip else abort, that is,
if \( \phi \) holds then continue without changing state,
if \( \phi \) does not hold then block without halting

\( \alpha; \beta \)
do \( \alpha \), then do \( \beta \)

\( \alpha \cup \beta \)
choose \( \alpha \) or \( \beta \) and execute it

\( \alpha^* \)
choose \( n \geq 0 \) and execute \( \alpha \) \( n \) times

mutual dependency: examples

\([p]?p\]
if \( p \) halts then in a state satisfying \( p \) with \( p \) an atomic proposition

\((p)?p\)
it is possible to execute \( p \) and halt in a state where \( p \) holds

\([\alpha]\bot\)
\( \alpha \) never terminates

\([\alpha]\top\)
is always true

\(\top?\)
is skip

\(\bot?\)
is fail (unsuccessful halt)

PDL formulas: definition

atomic formula

\( p \) from a set \( \text{Var} \) of atomic propositions

true and false

\( \top \) and \( \bot \)

negation

\( \neg \phi \)

conjunction

\( \phi \land \psi \)

diamond

\( \langle \alpha \rangle \phi \), with \( \alpha \) a program, so depends on the grammar for programs

PDL formulas: examples

\([\alpha \cup \beta] \phi\)
always if we execute \( \alpha \) or \( \beta \) we arrive at a state where \( \phi \) holds

\((p)?p\)
there is a sequence of alternating executions of \( \alpha \) and \( \beta \) bringing us to a state where \( \phi \) holds

\((\langle \alpha \rangle \phi)^*\)
\( \phi \) holds after a finite number (\( n \geq 0 \)) of \( \alpha \) steps

\(\phi\leftrightarrow\phi \lor (\langle \alpha \rangle; \alpha^*) \phi\)
if and only if

either \( \phi \) holds here (\( n = 0 \)), or (\( n > 0 \)) we can do an \( \alpha \) step and then more \( \alpha \) steps to reach a state where \( \phi \) holds
PDL formulas: more examples

\[ [\alpha] \phi \land \psi \iff [\alpha]\phi \land [\alpha]\psi \]  
(seems a tautology)

\[ [\alpha; \beta] \phi \iff [\alpha][\beta] \phi \]  
(seems a tautology)

\[ [\alpha] p \iff [\beta] p \]  
(gives an equivalence between \( \alpha \) and \( \beta \))

towards a semantics for PDL formulas

we obtain the semantics as an instance of multi-modal logic

in particular:
\( \mathcal{M}, s \models (\alpha) \phi \) iff there is \( s' \) such that \((s, s') \in R_\alpha \) and \( \mathcal{M}, s' \models \phi \)

however:

an arbitrary model does respect the intended meaning of the programs

therefore we will impose conditions on the relations \( R_\alpha \)

intuitive requirements for a PDL model

consider \( a; b \) and \( R_{a;b} \)

consider \( a \cup b \) and \( R_{a \cup b} \)

consider \( a^* \) and \( R_{a^*} \)

this suggests to start from all the \( R_\alpha \) with \( a \in A \) an atomic program

but what to do with \( R_{\beta^*} \)?

PDL frame: definition

a Prog-frame \( \mathcal{F} = (W, \{ R_\alpha \mid \alpha \in \text{Prog} \}) \) is a PDL-frame if

\[ R_{\alpha \beta} = R_\alpha ; R_\beta, \text{ and} \]

\[ R_{\alpha \cup \beta} = R_\alpha \cup R_\beta, \text{ and} \]

\[ R_{\alpha^*} = (R_\alpha)^* \]

so if we know all \( R_\alpha \) then we know enough!

what are the definitions on the relations?
definitions on relations

the composition of \( R \) and \( S \): \( R; S = \{(x, z) | \exists y : Rxy \land Syz\} \)

the union of \( R \) and \( S \): \( R \cup S = \{(x, y) | Rxy \lor Sxy\} \)

more definitions on relations

the identity relation: \( \text{Id} = \{(x, x)\} \)

the \( n \)-fold composition of \( R \): \( R^0 = \text{Id} \) and \( R^{n+1} = R^n; R \)

the reflexive-transitive closure of \( R \): \( R^* = \bigcup_{n \geq 0} R^n \)

note: if \( xR^*y \), then there exists \( n \geq 0 \) and there exist \( x_1, \ldots, x_{n-1} \) such that \( x = x_0Rx_1R \ldots Rx_n = y \)

note: \( R^* \) is the smallest reflexive and transitive relation containing \( R \)

PDL model: definition

a model \( \mathcal{M} = (W, \{R_\alpha | \alpha \in \text{Prog}\}, V) \) is a PDL-model if

\( (W, \{R_\alpha | \alpha \in \text{Prog}\} \) is a PDL-frame, and

\( R_\phi^\mathcal{M} = \{(w, w) | \mathcal{M}, w \models \phi\} \)

PDL extension: definition

we can get a PDL model as the extension of a model over labels \( A \)

Let \( \mathcal{M} = (W, \{R_a | a \in A\}, V) \) be an \( A \)-model

Its PDL-extension is defined as \( \hat{\mathcal{M}} = (W, \{\hat{R}_\alpha | \alpha \in \text{Prog}\}, V) \) with

\( \hat{R}_a = R_a \)

\( \hat{R}_{\alpha;\beta} = \hat{R}_\alpha; \hat{R}_\beta \)

\( \hat{R}_{\alpha\cup\beta} = \hat{R}_\alpha \cup \hat{R}_\beta \)

\( \hat{R}_\alpha^* = (R_\alpha)^* \)

\( \hat{R}_\phi^\mathcal{M} = \{(x, x) | \mathcal{M}, x \models \phi\} \)