Non-classical logics

Lecture 8: Many-valued logics (5)

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Exam

Best possibilities:

Tuesday, 24.02.2015 (7)

Wednesday, 18.02.2015 (6)

[Sunday, 22.02.2015 (6)]

Wednesday, 25.02.2015 (6)

Thursday, 12.03.2015 (6)

Suggested date: Thursday, 12.03.2015, 10:00-12:00

Until now

- Many-valued logics (finitely-valued; infinitely-valued)

  History and Motivation
  Syntax
  Semantics

- Finitely-valued logics

  Functional completeness

  Automated reasoning:

    Tableaux

    Resolution

- Infinitely-valued logics

  Łukasiewicz logics, comparison

  Fuzzy logics

"Fuzzy" logics

$W = [0, 1]$

Question: How to define conjunction?

Answer: Desired conditions

$f : [0, 1]^2 \to [0, 1]$ such that:

- f associative and commutative
- for all $0 \leq A \leq B \leq 1$ and all $0 \leq C \leq 1$ we have $f(A, C) \leq f(B, C)$
- for all $0 \leq C \leq 1$ we have $f(C, 1) = C$.

Definition A function with the properties above is called a t-norm.

Examples of t-norms

Gödel t-norm	$f_G(x, y) = \min(x, y)$
Łukasiewicz t-norm	$f_{\text{Ł}}(x, y) = \max(0, x + y - 1)$
Product t-norm	$f_P(x, y) = x \cdot y$

Left-continuous t-norm

Definition. A t-norm f is left-continuous if for every $x, y \in [0,1]$ and every sequence $\{x_n\}_{n\in\mathbb{N}}$ with $0 \leq x_n \leq x$ and $lim_{n\to\infty} x_n = x$ we have $lim_{n\to\infty} f(x_n, y) = f(x, y)$.

Left-continuous t-norm

Definition. A t-norm f is left-continuous if for every $x, y \in [0,1]$ and every sequence $\{x_n\}_{n \in \mathbb{N}}$ with $0 \leq x_n \leq x$ and $lim_{n\to\infty} x_n = x$ we have $lim_{n\to\infty} f(x_n, y) = f(x, y)$.

The following t-norms are left continuous:

Gödel t-norm	$f_G(x, y) = \min(x, y)$
Łukasiewicz t-norm	$f_Ł(x, y) = \max(0, x + y - 1)$
Product t-norm	$f_P(x, y) = x \cdot y$

Left continuous t-norms

With every left continuous t-norm f we can associate the following operations:

- $x \circ_f y = f(x, y)$
- $x \oplus_f y = 1 - f(1 - x, 1 - y)$
- $x \Rightarrow_f y = \max\{z \mid f(x, z) \leq y\}$
- $\neg_f x = x \Rightarrow_f 0$

Remark: Left continuity ensures that $\max\{z \mid f(x, z) \leq y\}$ exists.

Validity: $D = \{1\}$

Left continuous t-norms

With every left continuous t-norm f we can associate the following operations:

- $x \circ_f y = f(x, y)$
- $x \oplus_f y = 1 - f(1 - x, 1 - y)$
- $x \Rightarrow_f y = \max\{z \mid f(x, z) \leq y\}$
- $\neg_f x = x \Rightarrow_f 0$

Łukasiewicz t-norm

$$x \circ_{\text{Ł}} y = \max(0, x + y - 1)$$

$$x \oplus_{\text{Ł}} y = 1 - \max(0, 1 - x - y)$$

$$x \Rightarrow_f y = \min(1, 1 - x + y)$$

$$\neg x = \min(1, 1 - x) = 1 - x$$

Left continuous t-norms

With every left continuous t-norm f we can associate the following operations:

- $x \circ_f y = f(x, y)$
- $x \oplus_f y = 1 - f(1 - x, 1 - y)$
- $x \Rightarrow_f y = \max\{z \mid f(x, z) \leq y\}$
- $\neg_f x = x \Rightarrow_f 0$

Łukasiewicz t-norm

$$x \circ_{Ł} y = \max(0, x + y - 1) \qquad x \wedge_{Ł} y = x \circ_{Ł} (x \Rightarrow y)$$

$$x \oplus_{Ł} y = 1 - \max(0, 1 - x - y) \qquad x \vee_{Ł} y = \neg_{Ł}((\neg_{Ł} x) \wedge_{Ł} (\neg_{Ł} y))$$

$$x \Rightarrow_f y = \min(1, 1 - x + y)$$

$$\neg x = \min(1, 1 - x) = 1 - x$$

Left continuous t-norms

With every left continuous t-norm f we can associate the following operations:

- $x \circ_f y = f(x, y)$
- $x \oplus_f y = 1 - f(1 - x, 1 - y)$
- $x \Rightarrow_f y = \max\{z \mid f(x, z) \leq y\}$
- $\neg_f x = x \Rightarrow_f 0$

Gödel t-norm

$$x \circ_G y = \min(x, y)$$

$$x \oplus_G y = \max(x, y)$$

$$x \Rightarrow_G y = \max\{z \mid x \wedge z \leq y\} = \begin{cases} 1 & \text{if } x \leq y \\ y & \text{if } x > y \end{cases}$$

$$\neg_G x = \max\{z \mid x \wedge z = 0) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x > 0 \end{cases}$$

Checking validity of formulae in fuzzy logics

Given: F formula in a t-norm based fuzzy logic formed with the operations $\{\circ, \oplus, \neg, \Rightarrow\}$ (and also $\vee, \wedge$ if definable)

Task: Check whether F is valid (a tautology)
i.e. whether for all $\mathcal{A} : X \to [0, 1]$, $\mathcal{A}(F) = 1$

Idea:

Assume that there exists $\mathcal{A} : X \to [0, 1]$ such that $\mathcal{A}(F) \neq 1$.
Derive a contradiction.

Let $P_1, \dots, P_n$ be the propositional variables which occur in F.

Check whether $\exists x_1, \dots, x_n F(x_1, \dots, x_m) \neq 1$ is satisfiable in
$\mathcal{A} = ([0, 1], \{\circ_f, \oplus_f, \neg_f, \rightarrow_f, \leftrightarrow_f\})$.

Example 1: Łukasiewicz logic Ł $= \mathcal{L}_{\alpha_1}$

F $\mathcal{F}$-formula, where $\mathcal{F} = \{\vee, \wedge, \circ, \neg, \rightarrow, \leftrightarrow\}$.

Let $P_1, \dots, P_n$ be the propositional variables which occur in F.

Check whether $\exists x_1, \dots, x_n F(x_1, \dots, x_m) \neq 1$ is satisfiable in

$$[0,1]_{Ł} = ([0,1], \{\vee, \wedge, \circ, \neg, \rightarrow\})$$

where $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$ are the operations induced by the t-norm $f_{Ł}(x,y) = \max(0, x+y-1)$, i.e.:

$(\mathrm{Def}_{\circ_{Ł}})$	$x+y<1 \rightarrow x\circ y=0$	$x+y\geq 1 \rightarrow x\circ y=x+y-1$
$(\mathrm{Def}_{\vee})$	$x\leq y \rightarrow x\vee y=y$	$x>y \rightarrow x\vee y=x$
$(\mathrm{Def}_{\wedge})$	$x\leq y \rightarrow x\wedge y = x$	$x>y \rightarrow x\wedge y = y$
$(\mathrm{Def}_{\Rightarrow_{Ł}})$	$x\leq y \rightarrow x\Rightarrow y=1$	$x>y \rightarrow x\Rightarrow y = 1-x+y$
$(\mathrm{Def}_{\neg_{Ł}})$	$\neg x = 1 - x$	

Example 1: Łukasiewicz logic Ł $= \mathcal{L}_{\alpha_1}$

F $\mathcal{F}$-formula, where $\mathcal{F} = \{\vee, \wedge, \circ, \neg, \rightarrow, \leftrightarrow\}$.

Remark: The following are equivalent:

(1) $F(x_1, \dots, x_m) \neq 1$ is satisfiable in $[0,1]_{Ł} = ([0,1], \{\vee, \wedge, \circ, \neg, \rightarrow\})$, where $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$ are the operations induced by the t-norm $f_{Ł}$

(2) $\mathrm{Def}_{Ł} \wedge F(x_1, \dots, x_m) \neq 1$ satisfiable in $[0,1]$.

$(\mathrm{Def}_{\circ_{Ł}})$	$x+y<1 \rightarrow x\circ y=0$	$x+y\geq 1 \rightarrow x\circ y=x+y-1$
$(\mathrm{Def}_{\vee})$	$x\leq y \rightarrow x\vee y=y$	$x>y \rightarrow x\vee y=x$
$(\mathrm{Def}_{\wedge})$	$x\leq y \rightarrow x\wedge y = x$	$x>y \rightarrow x\wedge y = y$
$(\mathrm{Def}_{\Rightarrow_{Ł}})$	$x\leq y \rightarrow x\Rightarrow y=1$	$x>y \rightarrow x\Rightarrow y = 1-x+y$
$(\mathrm{Def}_{\neg_{Ł}})$	$\neg x = 1 - x$	

Example

To show: $((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y)$ is a tautology

New task: $\mathrm{Def}_{Ł} \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ unsatisfiable

where	($\mathrm{Def}_\vee$)	$x \leq y \rightarrow x \vee y = y$	$x > y \rightarrow x \vee y = x$
	($\mathrm{Def}_\wedge$)	$x \leq y \rightarrow x \wedge y = x$	$x > y \rightarrow x \wedge y = y$
	($\mathrm{Def}_{\circ_Ł}$)	$x + y < 1 \rightarrow x \circ y = 0$	$x + y \geq 1 \rightarrow x \circ y = x + y - 1$
	($\mathrm{Def}_{\Rightarrow_Ł}$)	$x \leq y \rightarrow x \Rightarrow y = 1$	$x > y \rightarrow x \Rightarrow y = 1 - x + y$

Example

New task: $\text{Def}_{Ł} \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ unsatisfiable

where

$$
\begin{array}{lll}
(\text{Def}_\vee) & x{\leq}y \rightarrow x{\vee}y{=}y & x{>}y \rightarrow x{\vee}y{=}x \\
(\text{Def}_\wedge) & x{\leq}y \rightarrow x{\wedge}y = x & x{>}y \rightarrow x{\wedge}y = y \\
(\text{Def}_{\circ_Ł}) & x{+}y{<}1 \rightarrow x{\circ}y{=}0 & x{+}y{\geq}1 \rightarrow x{\circ}y{=}x{+}y{-}1 \\
(\text{Def}_{\Rightarrow_Ł}) & x{\leq}y \rightarrow x{\Rightarrow}y{=}1 & x{>}y \rightarrow x{\Rightarrow}y = 1{-}x{+}y
\end{array}
$$

1. Rename subterms starting with Ł-operators and expand definitions:

$$
\begin{array}{l|llll}
p = x \Rightarrow 0 & s \neq 1 & x \leq 0 \rightarrow x \Rightarrow 0 = 1 & x > 0 \rightarrow x \Rightarrow 0 = 1 - x + 0 \\
q = p \Rightarrow 0 & & p \leq 0 \rightarrow p \Rightarrow 0 = 1 & p > 0 \rightarrow p \Rightarrow 0 = 1 - p + 0 \\
r = x \vee y & & q \leq r \rightarrow q \Rightarrow r = 1 & q > r \rightarrow q \Rightarrow r = 1 - q + r \\
s = q \Rightarrow r & & x \leq y \rightarrow x \vee y = y & x > y \rightarrow x \vee y = x
\end{array}
$$

Example

New task: $\mathrm{Def}_{\text{Ł}} \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ unsatisfiable

where

	$(\mathrm{Def}_{\vee})$	$x \leq y \rightarrow x \vee y = y$	$x > y \rightarrow x \vee y = x$
	$(\mathrm{Def}_{\wedge})$	$x \leq y \rightarrow x \wedge y = x$	$x > y \rightarrow x \wedge y = y$
	$(\mathrm{Def}_{\circ_{\text{Ł}}})$	$x + y < 1 \rightarrow x \circ y = 0$	$x + y \geq 1 \rightarrow x \circ y = x + y - 1$
	$(\mathrm{Def}_{\Rightarrow_{\text{Ł}}})$	$x \leq y \rightarrow x \Rightarrow y = 1$	$x > y \rightarrow x \Rightarrow y = 1 - x + y$

2. Replace terms starting with Ł-operations; SAT checking in [0, 1]

$p = x \Rightarrow 0$	$s \neq 1$	$x \leq 0 \rightarrow p = 1$	$x > 0 \rightarrow p = 1 - x + 0$
$q = p \Rightarrow 0$		$p \leq 0 \rightarrow q = 1$	$p > 0 \rightarrow q = 1 - p + 0$
$r = x \vee y$		$q \leq r \rightarrow s = 1$	$q > r \rightarrow s = 1 - q + r$
$s = q \Rightarrow r$		$x \leq y \rightarrow r = y$	$x > y \rightarrow r = x$

Reduction to checking constraints over [0, 1]

Reduction to checking satisfiability in [0, 1] of constraints in linear arithmetic (implications of LA expressions).

NP complete [Sonntag'85]

Similar techniques can be used also for Gödel logics (with the Gödel t-norm).

This method was first described (in a slightly more general context) in:

Viorica Sofronie-Stokkermans and Carsten Ihlemann,
"Automated reasoning in some local extensions of ordered structures."
Proceedings of ISMVL'07, IEEE Press, paper 1, 2007.

and (with full proofs) in

Viorica Sofronie-Stokkermans and Carsten Ihlemann,
"Automated reasoning in some local extensions of ordered structures."
Journal of Multiple-Valued Logics and Soft Computing
(Special issue dedicated to ISMVL'07) 13 (4-6), 397-414, 2007.

Example 1: Gödel logic

F $\mathcal{F}$-formula, where $\mathcal{F} = \{\vee, \wedge, \neg, \rightarrow, \leftrightarrow\}$.

Let $P_1, \dots, P_n$ be the propositional variables which occur in F.

Check whether $\exists x_1, \dots, x_n F(x_1, \dots, x_m) \neq 1$ is satisfiable in

$$[0,1]_G = ([0,1], \{\vee, \wedge, \circ, \neg, \rightarrow\})$$

where $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$ are the operations induced by the t-norm $f_{\mathsf{G}}(x, y) = \min(x, y)$, i.e.:

$$\begin{array}{rll}
(\mathsf{Def}_\circ) = (\mathsf{Def}_\wedge) & x{\leq}y \rightarrow x{\wedge}y = x & x{>}y \rightarrow x{\wedge}y = y \\
(\mathsf{Def}_\vee) & x{\leq}y \rightarrow x{\vee}y{=}y & x{>}y \rightarrow x{\vee}y{=}x \\
(\mathsf{Def}_\Rightarrow) & x{\leq}y \rightarrow x{\Rightarrow}y{=}1 & x{>}y \rightarrow x{\Rightarrow}y = y \\
(\mathsf{Def}_\neg) & x = 0 \rightarrow \neg x = 1 & x > 0 \rightarrow \neg x = 0
\end{array}$$

Example

Check whether $((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y)$ is a tautology in the Gödel logic.
New task: $\mathrm{Def}_G \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ satisfiable?

where

$$
\begin{array}{rlll}
(\mathrm{Def}_\circ) = (\mathrm{Def}_\wedge) & x \leq y \rightarrow x \wedge y = x & & x > y \rightarrow x \wedge y = y \\
(\mathrm{Def}_\vee) & x \leq y \rightarrow x \vee y = y & & x > y \rightarrow x \vee y = x \\
(\mathrm{Def}_\Rightarrow) & x \leq y \rightarrow x \Rightarrow y = 1 & & x > y \rightarrow x \Rightarrow y = y \\
(\mathrm{Def}_\neg) & x = 0 \rightarrow \neg x = 1 & & x > 0 \rightarrow \neg x = 0
\end{array}
$$

Example

New task: $\text{Def}_G \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ satisfiable?

where $(\text{Def}_\circ) = (\text{Def}_\wedge)$ $x \leq y \rightarrow x \wedge y = x$ $\quad x > y \rightarrow x \wedge y = y$

$(\text{Def}_\vee)$ $x \leq y \rightarrow x \vee y = y$ $\quad x > y \rightarrow x \vee y = x$

$(\text{Def}_\Rightarrow)$ $x \leq y \rightarrow x \Rightarrow y = 1$ $\quad x > y \rightarrow x \Rightarrow y = y$

$(\text{Def}_\neg)$ $x = 0 \rightarrow \neg x = 1$ $\quad x > 0 \rightarrow \neg x = 0$

1. Rename subterms starting with Ł-operators and expand definitions:

$$
\begin{array}{l|llll}
p = x \Rightarrow 0 & s \neq 1 & x \leq 0 \rightarrow x \Rightarrow 0 = 1 & x > 0 \rightarrow x \Rightarrow 0 = 0 \\
q = p \Rightarrow 0 & & p \leq 0 \rightarrow p \Rightarrow 0 = 1 & p > 0 \rightarrow p \Rightarrow 0 = 0 \\
r = x \vee y & & q \leq r \rightarrow q \Rightarrow r = 1 & q > r \rightarrow q \Rightarrow r = r \\
s = q \Rightarrow r & & x \leq y \rightarrow x \vee y = y & x > y \rightarrow x \vee y = x
\end{array}
$$

Example

New task: $\text{Def}_G \wedge \underbrace{((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y) \neq 1}_{G_1}$ satisfiable?

where

$$
\begin{array}{rlll}
(\text{Def}_\circ) = (\text{Def}_\wedge) & x{\leq}y \rightarrow x{\wedge}y = x & & x{>}y \rightarrow x{\wedge}y = y \\
(\text{Def}_\vee) & x{\leq}y \rightarrow x{\vee}y{=}y & & x{>}y \rightarrow x{\vee}y{=}x \\
(\text{Def}_\Rightarrow) & x{\leq}y \rightarrow x{\Rightarrow}y{=}1 & & x{>}y \rightarrow x{\Rightarrow}y = y \\
(\text{Def}_\neg) & x = 0 \rightarrow \neg x = 1 & & x > 0 \rightarrow \neg x = 0
\end{array}
$$

2. Replace terms starting with Ł-operations; SAT checking in [0, 1]

$$
\begin{array}{l|llll}
p = x \Rightarrow 0 & s \neq 1 & x \leq 0 \rightarrow p = 1 & x > 0 \rightarrow p = 0 \\
q = p \Rightarrow 0 & & p \leq 0 \rightarrow q = 1 & p > 0 \rightarrow q = 0 \\
r = x \vee y & & q \leq r \rightarrow s = 1 & q > r \rightarrow s = r \\
s = q \Rightarrow r & & x \leq y \rightarrow r = y & x > y \rightarrow r = x
\end{array}
$$

Satisfiable (e.g. by $\beta(x){=}\beta(y){=}\frac{1}{2}$), so $((x \Rightarrow 0) \Rightarrow 0) \Rightarrow (x \vee y)$ not tautology in Gödel logic.

Product logic

Similar techniques can be used also for the product logic
(with the product t-norm)

$\mapsto$ non-linearity (hence higher complexity)

Many-Valued Logics

- Many-valued logics (finitely-valued)

  History and Motivation
  Syntax /Semantics

  Functional completeness

  Automated reasoning: Tableaux, Resolution

- Infinitely-valued logics

  Examples: Łukasiewics logics $\mathcal{L}_{\aleph_0}, \mathcal{L}_{\aleph_1}$
  description of the tautologies

  Fuzzy logics:
  - t-norms, Łukasiewics, Gödel, Product t-norm
  - Łukasiewics logic, Gödel logic, Product logic
  - Automated methods for checking validity

Applications of many-valued logic

- independence proofs
- modeling undefined function and predicate values (program verification)
- semantic of natural languages
- theory of logic programming: declarative description of operational semantics of negation
- modeling of electronic circuits
- modeling vagueness and uncertainly
- shape analysis (program verification)

Applications of many-valued logic

- independence proofs
- modeling undefined function and predicate values (program verification)
- semantic of natural languages
- theory of logic programming: declarative description of operational semantics of negation
- modeling of electronic circuits
- modeling vagueness and uncertainly
- shape analysis (program verification)

Independence proofs

Task: Check independence of axioms in axiom systems [Bernays 1926]

Here: Example: Axiom system for propositional logic K_1

Ax1 $p_1 \Rightarrow (p_2 \Rightarrow p_1)$

Ax2 $((p_1 \Rightarrow p_2) \Rightarrow p_1) \Rightarrow p_1$

Ax3 $(p_1 \Rightarrow p_2) \Rightarrow ((p_2 \Rightarrow p_3) \Rightarrow (p_1 \Rightarrow p_3))$

Ax4 $(p_1 \wedge p_2) \Rightarrow p_1$

Ax5 $(p_1 \wedge p_2) \Rightarrow p_2$

Ax6 $(p_1 \Rightarrow p_2) \Rightarrow ((p_1 \Rightarrow p_3) \Rightarrow p_1 \Rightarrow p_2 \wedge p_3))$

Ax7 $p_1 \Rightarrow (p_1 \vee p_2)$

Ax8 $p_2 \Rightarrow (p_1 \vee p_2)$

Axiom system: K_1

Ax9 $(p_1 \Rightarrow p_3) \Rightarrow ((p_2 \Rightarrow p_3) \Rightarrow p_1 \vee p_2 \Rightarrow p_3))$

Ax10 $(p_1 \approx p_2) \Rightarrow (p_1 \Rightarrow p_2)$

Ax11 $(p_1 \approx p_2) \Rightarrow (p_2 \Rightarrow p_1)$

Ax12 $(p_1 \Rightarrow p_2) \Rightarrow ((p_2 \Rightarrow p_1) \Rightarrow p_1 \approx p_2))$

Ax13 $(p_1 \Rightarrow p_2) \Rightarrow (\neg p_2 \Rightarrow \neg p_1)$

Ax14 $p_1 \Rightarrow \neg\neg p_1$

Ax15 $\neg\neg p_1 \Rightarrow p_1$

Inference rule: Modus Ponens: $\dfrac{H \quad H \Rightarrow G}{G}$

Independence

Definition: An axiom system K is independent iff for every axiom $A \in K$, A is not provable from $K \setminus \{A\}$.

We will show that Ax2 is independent

Independence

Definition: An axiom system K is independent iff for every axiom $A \in K$, A is not provable from $K \backslash \{A\}$.

We will show that Ax2 is independent

Idea: We introduce a 3-valued logic L_{K_1} with truth values $\{0, u, 1\}$, $D = \{1\}$ and operations $\neg, \Rightarrow, \wedge, \vee, \approx$ as defined for $\mathcal{L}_3$ in the lecture.

To show:

1. Every axiom in K_1 except for $Ax2$ is a L_{K_1}-tautology.
2. Modus Ponens leads from L_{K_1} tautologies to a L_{K_1}-tautology.
3. $Ax2$ is not a L_{K_1}-tautology.

Independence

From 1,2,3 it follows that every formula which can be proved from $K_1 \backslash Ax2$ is a tautology.

Hence – since $Ax2$ is not a tautology – $K_1 \backslash \{Ax2\} \not\models Ax2$.

Proof

We introduce a 3-valued logic L_{K_1} with truth values $\{0, u, 1\}$, $D = \{1\}$ and operations $\neg, \Rightarrow, \wedge, \vee, \approx$ as defined for $calL_3$ in the lecture.

To show:

1. Every axiom in K_1 except for $Ax2$ is a L_{K_1}-tautology.
2. Modus Ponens leads from L_{K_1} tautologies to a L_{K_1}-tautology.
3. $Ax2$ is not a L_{K_1}-tautology.

Proof

We introduce a 3-valued logic L_{K_1} with truth values $\{0, u, 1\}$, $D = \{1\}$ and operations $\neg, \Rightarrow, \wedge, \vee, \approx$ as defined in the lecture.

To show:

1. Every axiom in K_1 except for $Ax2$ is a L_{K_1}-tautology.
2. Modus Ponens leads from L_{K_1} tautologies to a L_{K_1}-tautology.
3. $Ax2$ is not a L_{K_1}-tautology.

1. Routine (check all axioms in $K_1 \setminus \{Ax2\}$).

Proof

We introduce a 3-valued logic L_{K_1} with truth values $\{0, u, 1\}$, $D = \{1\}$ and operations $\neg, \Rightarrow, \wedge, \vee, \approx$ as defined in the lecture.

To show:

1. Every axiom in K_1 except for $Ax2$ is a L_{K_1}-tautology.
2. Modus Ponens leads from L_{K_1} tautologies to a L_{K_1}-tautology.
3. $Ax2$ is not a L_{K_1}-tautology.

2. Analyze the truth table of $\Rightarrow$.

Assume H is a tautology and $H \Rightarrow G$ is a tautology.

Let $\mathcal{A} : \Pi \to \{0, u, 1\}$.

Then $\mathcal{A}(H) = 1$ and $\mathcal{A}(H \Rightarrow G) = 1$, so $\mathcal{A}(G) = 1$.

Proof

We introduce a 3-valued logic L_{K_1} with truth values $\{0, u, 1\}$, $D = \{1\}$ and operations $\neg, \Rightarrow, \wedge, \vee, \approx$ as defined in the lecture.

To show:

1. Every axiom in K_1 except for $Ax2$ is a L_{K_1}-tautology.
2. Modus Ponens leads from L_{K_1} tautologies to a L_{K_1}-tautology.
3. $Ax2$ is not a L_{K_1}-tautology.

3. Let $\mathcal{A} : \Pi \to \{0, u, 1\}$ with $\mathcal{A}(p_1) = u$ and $\mathcal{A}(p_2) = 0$.

Then

$$\begin{aligned} \mathcal{A}(((p_1 \Rightarrow p_2) \Rightarrow p_1) \Rightarrow p_1) &= ((u \Rightarrow 0) \Rightarrow u) \Rightarrow u \\ &= (u \Rightarrow u) \Rightarrow u = u. \end{aligned}$$

Applications of many-valued logic

- independence proofs
- modeling undefined function and predicate values (program verification)
- semantic of natural languages
- theory of logic programming: declarative description of operational semantics of negation
- modeling of electronic circuits
- modeling vagueness and uncertainly
- shape analysis (program verification)

Shape analysis

Shape Analysis is an important and well covered part of static program analysis.

The central role in shape analysis is played by the set U of abstract stores.

U is perceived as the abstraction of the locations program variables can point to.

In an object-oriented context U can be viewed as an abstraction of the set of all objects existing at a snapshot during program execution

Shape analysis

U set of abstract stores.

X set of program variables.

Abstract state of a program at a given snapshot:

- Structure $\mathcal{S} = (U, \{x : U \to \{0,1\}\}_{x \in X} \cup$ Additional predicates)

  $x(v) = 1$ (also denoted $\mathcal{S} \models x[v]$) iff variable x points to store v.

For any abstract state $\mathcal{S}$ and any program variable x we require that the unary predicate x holds true of at most one store, i.e. we require

$$\mathcal{S} \models \forall s_1 \forall s_2 ((x(s_1) \wedge x(s_2)) \to s_1 = s_2).$$

It is possible that x does not point to any store, i.e. $\mathcal{S} \models \forall s(\neg x(s))$.

Shape analysis

Additional predicates on $\mathcal{S}$ depend on the specific program/task

Example: next : $U^2 \to \{0, 1\}$

Examples of properties:

$\exists s\ x(s)$	x does not point to null
$\forall s(\neg(x(s) \wedge t(s)))$	x and t do not point to the same store
$\exists s$ is(s)	the list defined by next contains a shared node

We have used the abbreviation

$$\text{is}(s) = \exists s_1 \exists s_2 (next(s_1, s) \wedge next(s_2, s) \wedge s_1 \neq s_2)$$

Goal: prove for a given program, or a given program part, that a certain property holds at every program state, or every stable program state.

Example: List reversing

Goal: Cycle-freeness of a list pointer structure is preserved by the algorithm reversing the list.

Describing cycle-freeness

1. $\neg\exists v(next(v, n)$ n is the store representing the head of the list
2. $\forall v \forall w(next(m, v) \wedge next(m, w) \rightarrow v = w)$ for all stores m reachable from n,
3. $\neg \mathrm{is}(m)$ for all stores m reachable from n.

Remark:

If conditions 1.–3. hold then the list with entry point n cannot be cyclic.

We concentrate here on showing the preservation of the formula $\mathrm{is}(s)$.

Example: List reversing

Algorithm for list reversing:

```
class ReverseList {
   int value;
   ReverseList next;


public ReverseList reverse() {
       ReverseList t, y= null, x = this;
       while (x != null) {
       st1: t=y;
       st2: y=x;
       st3: x=x.next;
       st4: y.next = t;}
       return y;}}
```

Example: List reversing

Task:

Assume that at the beginning of the while loop $\mathcal{S} \models \neg is(n)$ is true for all stores n in the list.

Show that in the state $\mathcal{S}_e$ after execution of the while loop again $\mathcal{S}_e \models \neg is(n)$ holds true for all n.

Problem: Since we cannot make any assumptions on the set of stores U at the start of the while-loop we need to investigate infinitely many structures, which obviously is not possible.

Shape analysis

Idea [Mooly Sagiv, Thomas Reps and Reinhard Wilhelm]

Use of three-valued structures to approximate two-valued structures.

More precisely, we try to find finitely many three-valued structures $\mathcal{S}_1^3, ..., \mathcal{S}_k^3$ such that for an arbitrary two-valued abstract state $\mathcal{S}$ that may be possible before the while-loop starts there is a surjective mapping F from S onto one of the $\mathcal{S}_i^3$ for $1 \leq i \leq k$ with $\mathcal{S} \sqsubseteq^F \mathcal{S}_i^3$, i.e.

- for all n-ary predicate symbols p and all $b_1, \ldots, b_n \in U_{\mathcal{S}}$ we have:

$$p_{\mathcal{S}_i^3}(F(b_1), \ldots, F(b_n)) \leq_i p_{\mathcal{S}}(b_1, \ldots, b_n)$$

bb where $a \leq_i b$ iff $a = b$ or $a = \frac{1}{2}$

(every possible initial state has an abstraction among $\mathcal{S}_1^3, ..., \mathcal{S}_k^3$)

Shape analysis

Plan:

Step 1:

For every three-valued structure $\mathcal{S}_i^3$ we will define an algorithm to compute a three-valued structure $\mathcal{S}_{i,e}^3$.

We think of $\mathcal{S}_{i,e}^3$ as the three-valued state reached after execution of α_r (the body of the while-loop) when started in $\mathcal{S}_i^3$.

If $\mathcal{S}$ is a two-valued state it is fairly straight forward to compute the two-valued state $\mathcal{S}_e$ that is reached after executing α_r starting with $\mathcal{S}$, since the commands in α_r are so simple.

The construction of $\mathcal{S}_{i,e}^3$ will be done such that $\mathcal{S} \sqsubseteq^F \mathcal{S}_i^3$ implies $\mathcal{S}_e \sqsubseteq^F \mathcal{S}_{i,e}^3$.

Shape analysis

Plan:

Step 2:

Determine a set $\mathcal{M}_0$ of abstract three-valued states to start with.

Shape analysis

Plan:

Step 3:

At iteration $k (k \geq 1)$ we are dealing with a set $\mathcal{M}_{k-1}$ of abstract three-valued states.

We try to prove for every $\mathcal{S}^3 \in \mathcal{M}_{k-1}$ that if $\mathcal{S}^3 \models \forall s(\neg \mathsf{is}(s)))$ then $\mathcal{S}^3_e \models (\forall s(\neg \mathsf{is}(s)))$.

It will then follow that for any two-valued state $\mathcal{S}$ that is reachable with $k-1$ iterations of α_r:

$$\mathcal{S} \models \forall \neg \mathsf{is}(s) \Rightarrow \mathcal{S}_e \models \forall s \neg \mathsf{is}(s)$$

If we succeed we set

$$\mathcal{M}_k = \{\mathcal{S}^3_e | \mathcal{S}^3 \in \mathcal{M}_{k-1}\}$$

Shape analysis

Plan:

Step 3 (continued)

If $\mathcal{M}_k \subseteq \mathcal{M}_{k-1}$ we are finished and the claim is positively established.

Otherwise we repeat step 3 with $\mathcal{M}_k$.

If for one $\mathcal{S}^3 \in \mathcal{M}_{k-1}$, $\forall s(\neg \mathrm{is}(s)))$ evaluated to 0 then our conjecture was false.

If for one $\mathcal{S}^3 \in \mathcal{M}_{k-1}$, $\forall s(\neg \mathrm{is}(s)))$ evaluated to $\frac{1}{2}$ then this result is inconclusive. Should this happen we need to iterate the procedure with a larger set $\mathcal{M}'_{k-1}$.

There is, unfortunately, no guarantee that this iteration will come to a conclusive end in the general case.

Shape analysis

[Example on the blackboard]

cf. also P.H. Schmidt's lecture notes, Section 2.4.4 (pages 91-100).