

Predavanja

ustni izpit: 40%

predstavitel: 20% (20 minut po 3 osebe)

Vaje

Projekt 40%

Logic in computer science

- Correctness of software/hardware/systems
(watch Xavier Leroy: In search of software perfection)
- Developing proof assistants

Propositional logic (natural deduction proof system)

Simple type theory : Curry-Howard correspondance
(propositions as types)

Propositional logic

propositional formulas ϕ, ψ, θ

propositional variables p

atomic formula:

$$\phi ::= p \mid \psi \wedge \theta \mid \psi \vee \theta \mid \psi \rightarrow \theta \mid \neg \theta \mid \perp$$

TAVTOLOGISA - validna formula (vedno res)

Natural deduction: proof system. Using

formal rules we prove a formula ϕ

We can prove formula $\Leftrightarrow$ it is a tautology

Primer: Proof of $p \wedge q \rightarrow p \vee r$

1. Assume $p \wedge q$ true (assumption)
2. p ($\wedge$ elimination)
3. $p \vee r$ ($\vee$ introduction₁)
4. $p \wedge q \rightarrow p \vee r$ ($\rightarrow$ introduction)

Za konstruktivno logiko (Brez LEM)

$\wedge$

$$\frac{\phi \quad \psi}{\phi \wedge \psi} \wedge_i \quad \frac{\phi \wedge \psi}{\phi} \wedge_{e1} \quad \frac{\phi \wedge \psi}{\psi} \wedge_{e2}$$

premises conclusion

$\vee$

$$\frac{\phi}{\phi \vee \psi} \vee_{i1} \quad \frac{\psi}{\phi \vee \psi} \vee_{i2} \quad \frac{\phi \vee \psi \quad \boxed{\begin{smallmatrix} \phi \\ \vdots \\ \bot \end{smallmatrix}} \quad \boxed{\begin{smallmatrix} \psi \\ \vdots \\ \bot \end{smallmatrix}}}{\bot} \vee_e$$

$\rightarrow$

$$\boxed{\begin{smallmatrix} \phi \\ \vdots \\ \psi \end{smallmatrix}} \quad \frac{\phi \rightarrow \psi \quad \phi}{\psi} \rightarrow_e \quad \text{modus ponens}$$

box discharges the assumption ϕ

$\neg$ $\neg \phi \equiv \phi \rightarrow \bot$

$$\boxed{\begin{smallmatrix} \phi \\ \vdots \\ \bot \end{smallmatrix}} \neg_i \quad \frac{\neg \phi \quad \phi}{\bot} \neg_e$$

$\bot$ $\frac{\bot}{\phi} \bot_e$

NO FORMULA IN THE BOX GETS USED OUTSIDE THE BOX.

Primer $p \wedge (q \vee r) \rightarrow (p \wedge q) \vee (p \wedge r)$

1. $\underline{p \wedge (q \vee r)}$ assumption
2. p
3. $q \vee r$
4. q assumption
5. $p \wedge q$
6. $(p \wedge q) \vee (p \wedge r)$
7. r assumption
8. $p \wedge r$
9. $(p \wedge q) \vee (p \wedge r)$ $\vee_e (3)(4-6) (7-9)$
10. $(p \wedge q) \vee (p \wedge r) \rightarrow (1-10)$

In classical logic

$$\boxed{\begin{smallmatrix} \neg \phi \\ \vdots \\ \bot \end{smallmatrix}} \quad \frac{\bot}{\phi} \text{raa} \quad \text{Reductio ad absurdum}$$

DN: prove LEM with raa

$$\frac{}{\vdots} \quad \frac{\neg \phi \quad \frac{\bot}{\phi}}{\phi \vee \neg \phi} \quad \frac{\neg \phi}{\phi \vee \neg \phi}$$

$\phi \vee \neg \phi$

A proof of formula ϕ is given by sequence of formulas following the rules such that all ^{open} assumptions in the proof are from Γ from assumptions Γ
notation: $\vdash \phi$

Theorem: $\vdash \phi$ (including raa) $\overset{\text{soundness}}{\iff} \phi$ is tautology
 $\underset{\text{completeness}}{\iff} \phi$ is true under any assignment of truth values to propositional variables that make all formulas in Γ true

Formulas $\phi ::= p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \perp$

Types $A ::= \alpha \mid A \times B \mid A + B \mid A \rightarrow B \mid$

$t : A$
 $\nearrow$ term $\nwarrow$ type

$\frac{s:A \quad t:B}{(s,t):A \times B}$

$\frac{t:A \times B}{pr_1(t):A} xe_1$

$\frac{t:A \times B}{pr_2(t):B}$

$\frac{\lambda x:A. \overset{\text{variable}}{t:B}}{\lambda x:A. t:A \rightarrow B} \rightarrow_i$
 term $\rightarrow$

$\frac{t:A \rightarrow B \quad u:A}{tu:B} \rightarrow_e$

$\frac{t:A}{in_1(t):A+B}$

$\frac{t:B}{in_2(t):A+B}$

$\frac{s:A+B \quad \boxed{\begin{array}{c} x:A \\ \vdots \\ t:C \end{array}} \quad \boxed{\begin{array}{c} y:B \\ \vdots \\ u:C \end{array}}}{\text{case } s \text{ (} in_1(x:A) \mapsto t \mid in_2(y:B) \mapsto u \text{)} : C}$

Primer:

A derived term of type $A \times B \rightarrow B \times A$

1. $x:A \times B$ { assumption } tež ne potrebujemo
2. $pr_1(x):A$ xe_1 (1)
3. $pr_2(x):B$ xe_2 (1)
4. $(pr_2(x), pr_1(x)):B \times A$ x_i (3) (2)
5. $\lambda x:A \times B. (pr_2(x), pr_1(x)):A \times B \rightarrow B \times A \rightarrow_i$ (1-4)

For every type we have introduction rules,
elimination rules and computation rules

computation rule for rewrite rule (longer arrow)

$$x: \begin{array}{l} pr_1(s, t) \xrightarrow{\quad} s \\ pr_2(s, t) \xrightarrow{\quad} t \end{array}$$

β -reduction
substitute u for
 x in t

$$\rightarrow: (\lambda x:A. t) u \longrightarrow t[x := u]$$

(before doing this first
rename bound variable
in t that are distinct
from variables in u)

$$+: \text{case } (in_1(s)) (in_1(x) \mapsto t \mid in_2(y) \mapsto u) \longrightarrow t[x := s]$$

$$0: \frac{t: 0}{\text{empty}(t): A}$$

DM: Find canonical term of type
 $A \times (B + C) \longrightarrow A \times B + A \times C$

Propositional logic $\sim$ Simple Type theory

Predicate logic (first order logic) $\sim$ Dependent type theory

Propositional logic formulas

$$\Phi ::= p \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid \Phi \rightarrow \Psi \mid \perp \quad \text{arity} \geq 0$$

Predicate logic extends prop. logic with predicates
expressing relations, and constants and function
symbols expressing elements $\hookrightarrow$ arity ≥ 0

Terms express elements

$$t ::= x \mid c \mid f(t_1, \dots, t_n)$$

$\uparrow$ constant $\nwarrow$ function symbol of arity n

Formulas express propositions

$$\Phi ::= P(t_1, \dots, t_n) \mid \Phi \wedge \Psi \mid \Phi \vee \Psi \mid \Phi \rightarrow \Psi \mid \perp \mid \forall x. \Phi \mid \exists x. \Phi$$

Example: a language for expressing property of integers in predicate logic

constants: 0, 1

function symbols $+$, $\cdot$ of arity 2

example terms z , $x+y$, $x \cdot (y + (z+1))$

predicates $=$, $\leq$

example formulas $x \leq y$, $\exists y \exists z \exists w \exists v. x = y \cdot y + z \cdot z + w \cdot w + v \cdot v$
↑
free variable

We could also define $\leq$ as

$$x \leq y \Leftrightarrow \exists x. \exists y. \exists v. \exists u. y = x + z \cdot z + w \cdot w + v \cdot v + u \cdot u$$

(vs naravna števila sa vsote štirih kvadratov)

$$x+1=0 \quad (x=-1)$$

We will interpret predicate logic in type theory
 Prop ... type of propositions

$$\frac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \wedge \Psi : \text{Prop}} \wedge \text{form} \quad \frac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \vee \Psi : \text{Prop}} \vee \text{form}$$

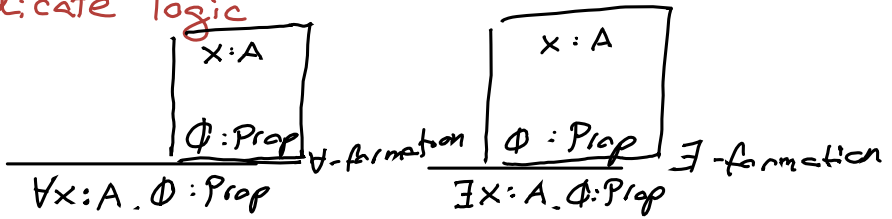
$$\frac{\Phi : \text{Prop} \quad \Psi : \text{Prop}}{\Phi \rightarrow \Psi : \text{Prop}} \rightarrow \text{formation} \quad \frac{}{\perp : \text{Prop}}$$

- (1) $p : \text{Prop}$ assumption
 - (2) $q : \text{Prop}$ assumption
 - (3) $p \wedge q : \text{Prop}$ $\wedge$ formation (1) (2)
 - (4) $q \wedge p : \text{Prop}$ $\wedge$ formation (2) (1)
 - (5) $p \wedge q \rightarrow q \wedge p : \text{Prop}$ $\rightarrow$ formation (3) (4)
- } open assumptions are called a context

$$p : \text{Prop}, q : \text{Prop} \vdash p \wedge q \rightarrow q \wedge p : \text{Prop}$$

For predicate logic we also need quantifiers

Predicate logic



(formation, not introduction)

- ① $p : A \rightarrow \text{Prop}$ assumption
- ② $x : A$ assumption
- ③ $p\ x : \text{Prop}$ $\rightarrow e$
- ④ $\forall x : A. p\ x : \text{Prop}$ $\forall$ -formation (2) (3)

$$p : A \rightarrow \text{Prop} \vdash \forall x : A. p\ x : \text{Prop}$$

(Instead of formation rules, we can include connectives and quantifiers as typed constants

$$\rightarrow, \vee, \wedge : \text{Prop} \rightarrow \text{Prop} \rightarrow \text{Prop}$$

$$\exists, \forall : (A \rightarrow \text{Prop}) \rightarrow \text{Prop}$$

iz Primera: $\forall (x : A. p(x))$

Example proof of

$$p:A \rightarrow \text{Prop}, g:A \rightarrow \text{Prop} \vdash (\forall x:A. p x) \rightarrow (\forall x:A. g x) \rightarrow (\forall x:A. p x \wedge g x)$$

3 judgments forms

$t:A$

$\Phi: \text{Prop}$ "is prop."

ϕ "is true"

① $p:A \rightarrow \text{Prop}$ *assump.*

② $g:A \rightarrow \text{Prop}$ *assump.*

③ $\forall x:A. p x$ *assump*

④ $\forall y:A. g y$ *assump*

⑤ $z:A$ *assum*

z se ne
prej pojaviti
v kontekstu

⑥ $p z$ *VE (3)*

⑦ $g z$ *VE (4)*

⑧ $p z \wedge g z$ *∧I (6) (7)*

⑨ $\forall z:A. p z \wedge g z$ *∧I (5)-(8)*

(10) $\forall y:A. g y \rightarrow \forall z:A. p z \wedge g z$

(11) $\forall x:A. p x \rightarrow (\dots) \rightarrow (\dots) \rightarrow (\dots)$
→ (4) - (3)
→ (9) - (10)

$z:A$
⋮
 Φ

$\forall z:A. \Phi$ *VI*

$\forall x:A. \Phi \quad t:A$
 $\Phi[x:=t]$ *VE*

$t:A \quad \Phi[x:=t]$
 $\exists x:A. \Phi$ *∃i*

$x:A$
 Φ
⋮
 Ψ

$\exists x:A. \Phi$

$\Psi: \text{Prop}$ *∃e*

Tega se ponavadi
ne piše

To pomeni: da je
 Ψ ne vsebuje x .
je prop-orkon brez x

DN: Dokaži

$$p:\text{Prop}, g:A \rightarrow \text{Prop} \vdash \exists x:A (p \wedge g x) \rightarrow p \wedge (\exists x:A. g x)$$

Simply we replace the grammar for every type with a universe Type and formation rules for types

We had: $A ::= \alpha \mid A \times A \mid A + A \mid A \rightarrow A \mid 0$

$$\frac{A : \text{Type} \quad B : \text{Type}}{A + B : \text{Type}}$$
$$A \times B : \text{Type}$$
$$A \rightarrow B : \text{Type}$$

Key idea of dependent type theory is types depend on the context

$\mathbb{N}$ set of natural numbers

List $\mathbb{N}$ set of lists (=finite sequences) of nat. numbers

Vector $\mathbb{N} \ n$ set of lists of nat. numbers of length n

Dependant product

construction $\prod_{n:\mathbb{N}} \text{vector } \mathbb{N}^n$ | function ϕ that maps $n:\mathbb{N}$ and return a value from the set $\text{vector } \mathbb{N}^n$

examples

$$n \mapsto \overbrace{0 \dots 0}^n$$

$$n \mapsto \underbrace{1, 2, \dots, n}_n$$

$x:A$ $\vdots$ $B:\text{Type}$

Π -formation

$\Pi x:A, B:\text{Type}$

type Nat

$\frac{}{\text{nat} : \text{Type}}$ formation rule

$\frac{}{0 : \text{nat}}$ $\frac{t : \text{nat}}{\text{succ } t : \text{nat}}$ introduction rules

alternative

$\text{succ} : \text{nat} \rightarrow \text{nat}$

Elimination and computation rules (after

$\frac{t : \text{nat} \quad P : \text{nat} \rightarrow \text{Prop} \quad - : P_0 \quad \boxed{\begin{array}{c} x : \text{nat} \\ \vdots \\ P(\text{succ } x) \end{array}}}{P t}$ Introduction rule

Instead of rule based formulations we can

assume rules:

introduction: $0 : \text{nat}$ $\text{succ} : \text{nat} \rightarrow \text{nat}$

elimination:

$\forall P : \text{nat} \rightarrow \text{Prop} \left(P 0 \rightarrow (\forall x : \text{nat}. P x \rightarrow P(\text{succ } x)) \rightarrow \forall x : \text{nat}. P x \right)$

$\hookrightarrow$ induction

iz indukcije v matematiki izrazimo *recursion theorem*

if v have A set and $a \in A$ and $A \rightarrow A$
 then there exist unique function $\Gamma: \mathbb{N} \rightarrow A$
 de $\Gamma(0) = a$ $\Gamma(n+1) = f(\Gamma(n))$

Recursion principle

$$R_{\text{nat}} : \prod_{P: \text{nat} \rightarrow \text{Type}} (P_0 \rightarrow (\prod_{x: \text{nat}} P_x \rightarrow P(\text{succ } x)) \rightarrow \prod_{x: \text{nat}} P_x)$$

$\uparrow$
recursor

$$R_{\text{nat}} P b f : \prod_{x: \text{nat}} P_x$$

computation rule:

$$R_{\text{nat}} P b f 0 \rightarrow b$$

$$R_{\text{nat}} P b f (\text{succ } n) \rightarrow f n (R_{\text{nat}} P b f) : P_{\text{succ } n}$$

Example:

addition $\text{add}: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}$

$\text{add} := \lambda m:\text{nat}. \lambda n:\text{nat}.$

$R_{\text{nat}}(\lambda_:\text{nat}. \text{Nat})\ m\ (\lambda_:\text{nat}. \lambda x:\text{nat}\ \text{succ}\ x)\ n$

$\text{add}\ (\overbrace{\text{succ}\ 0}^m)\ (\overbrace{\text{succ}\ 0}^n) \rightarrow^*$

$R_{\text{nat}}(\lambda_:\text{nat}. \text{nat})\ \text{succ}\ 0\ \lambda x:\text{nat}. \lambda x:\text{nat}\ \text{succ}\ x)\ \text{succ}\ n$
 $\rightarrow (\lambda_:\text{nat}. \lambda x:\text{nat}\ \text{succ}\ x)\ 0\ ((R_{\text{nat}}\ \lambda_:\text{nat}. \text{nat})$

$(\text{succ}\ 0)\ (\lambda_:\text{nat}. \lambda x:\text{nat}. \text{succ}\ x)\ 0))$

$\xrightarrow{*} \text{succ}\ (R_{\text{nat}}(\lambda_:\text{nat}. \text{nat})\ (\text{succ}\ 0)$

$(\lambda_:\text{nat}. \lambda x:\text{nat}. \text{succ}\ x)\ 0)$

$\rightarrow \text{succ}\ (\text{succ}\ 0) = 2 \quad \ddot{\circ}$

type List

$\frac{A \text{ type}}{\text{list } A : \text{Type}}$ function

$\frac{}{\text{nil} : \text{List } A} \quad \frac{a : A \quad s : \text{List } A}{\text{cons } s \ a : \text{list } A}$

$\text{nil} : \text{list } A$

$\text{cons} : A \rightarrow \text{List } A \rightarrow \text{List } A$. alternative

elimination, computation latter

Vector type

$$\frac{A: \text{Type} \quad n: \text{nat}}{\text{vector } A \ n \ \text{Type}}$$

$$\frac{}{\text{nil}: \text{vector } A \ 0} \quad \frac{a:A \quad \text{as}: \text{vector } A \ n}{\text{cons } a \ \text{as}: \text{vector } A \ (\text{succ } n)}$$

alternative constants:

$$\text{nil}: \text{vector } A \ 0$$

cons:

$$\prod_{n: \text{nat}} A \rightarrow \text{vector } A \ n \rightarrow \text{vector } A \ (\text{succ } n)$$

Vector is **dependent type**, because it's depends on the value of n

$$\begin{aligned} \text{Rvector}: & \prod_{P: \prod_{n: \text{nat}} \text{vector } A \ n \rightarrow \text{Type}} P \ 0 \ \text{nil} \rightarrow \\ & \prod_{a:A} \prod_{n: \text{nat}} \prod_{\text{as}: \text{vector } A \ n} P \ n \ \text{as} \rightarrow P \ (\text{succ } n) \ (\text{cons } n \ a \ \text{as}) \\ & \rightarrow \prod_{n: \text{nat}} \prod_{\text{as}: \text{vector } A \ n} P \ n \ \text{as} \end{aligned}$$

Computation rules

$$\begin{aligned} P: & \prod_{n: \text{nat}} (\text{vector } A \ n \rightarrow \text{Type}) \\ b: & P \ 0 \ \text{nil} \\ f: & \end{aligned}$$

$$\text{Rec}_{\text{vector } A} \ P \ b \ f: \prod_{n: \text{nat}} \prod_{\text{as}: \text{vec } A \ n} P \ n \ \text{as}$$

$$\text{Rec}_{\text{vec}} \ P \ b \ f \ 0 \ \text{nil} \longrightarrow b$$

$$\text{Rec}_{\text{vec}} \ P \ b \ f \ (\text{succ } n) \ (\text{cons } n \ a \ \text{as})$$

... ??
←
?

net, list, vectors so induktivni tipi

Dependent product type

$$\frac{A \text{ type} \quad \boxed{\begin{array}{c} x:A \\ \vdots \\ B \text{ type} \end{array}}}{\Pi_{x:A} B : \text{Type}}$$

intuitively: $\Pi_{x:A} B$ type of functionst
that maps $x:A$ to type B

introduction

$$\frac{\boxed{\begin{array}{c} x:A \\ \vdots \\ t:B \end{array}}}{\lambda x:A. t : \Pi_{x:A} B}$$

elimination

$$\frac{s : \Pi_{x:A} B \quad t:A}{s t : B[t/x]}$$

Ka imamo odvisni produkt, ne potrebujemo $\rightarrow$
 $A \rightarrow B := \Pi_{x:A} B$, kjer B ne vsebuje x

computation rule: $(\lambda x:A. t) u \rightarrow t[u/x]$
 β -reduction

Dependent sum type

$$\frac{A : \text{Type} \quad \boxed{\begin{array}{c} x : A \\ \vdots \\ B_{\text{Type}} \end{array}}}{\sum_{x:A} B : \text{Type}}$$

$$\frac{s : A \quad t : B[s/x]}{(s, t) : \sum_{x:A} B}$$

$$\frac{t : \sum_{x:A} B}{p_{r1}(t) : A}$$

$$\frac{t : \sum_{x:A} B}{p_{r2}(t) : B[p_{r1}(t)/x]}$$

$$(A \times B := \sum_{x:A} B)$$

Types and their elements

0 no elements

$A \times B$ pairs (a, b) , $a:A$ $b:B$

$A + B$ $\text{in}_1(a); a:A$ or $\text{in}_2(b); b:B$

$A \rightarrow B$ functions from A to B

$\prod_{a:A} B$ functions from $a:A \rightarrow B[a/x]$

$\sum_{a:A} B$ pairs (a, b) where $a:A$ $b:B[a/x]$

The Brouwer-Heyting-Kleene interpretation of logic

$\perp$ has no proof

proof of $\Phi \wedge \Psi$ is pair (a, b) where a is proof of Φ and b is proof of Ψ

propositions as subterminal types

if $\Phi : \text{Prop}$ then $\Phi : \text{Type}$

(Prop is a subuniverse of Type)

Prop is closed under Π -type

$$\left(\frac{A : \text{Type} \quad \left(\frac{x : A}{\Phi \text{ Prop}} \right)}{\Pi a : A. \Phi : \text{Prop}} \right)$$

if Φ is Prop then Φ is subterminal

- if $a, b : \Phi$ then $a = b$ (equality next week)

Also:

$\text{Prop} : \text{Type}$ - impredicativity

We can take Π as the only constructor

Scott-Pradic encoding of intuitionistic logic

$\perp := \prod_{p:\text{prop}} p$
 "for all propositions P , P is true"

$$\frac{a:\prod_{p:\text{prop}} p \quad \bar{\Phi}:\text{prop}}{a \cdot \bar{\Phi} : \Phi} \quad \text{tag} \quad \frac{\perp}{\Phi} \text{prop}$$

$$\bar{\Phi} \wedge \psi \quad \prod_{p:\text{prop}} (\bar{\Phi} \rightarrow \psi \rightarrow p) \rightarrow p$$

$$\bar{\Phi} \vee \psi \quad \prod_{p:\text{prop}} (\bar{\Phi} \rightarrow p) \rightarrow (\psi \rightarrow p) \rightarrow p$$

$$\Phi \rightarrow \psi$$

$$\forall x:A. \bar{\Phi} \quad \prod_{x:A} \bar{\Phi}$$

$$\exists x:A. \Phi \quad \prod_{p:\text{prop}} (\prod_{x:A} (\Phi \rightarrow p)) \rightarrow p$$

Basic properties

$A : \text{Type}$

$t : A$ - means proof that $A : \text{Type}$

$s \equiv t : A$ s and t are judgmentally equal

Derived judgment forms

$\Phi : \text{Prop}$... instance of $t : A$

$t : \Phi$

Φ means $_\Phi$ (abstracted $t : \Phi$)

Propositional equality

$$\text{Axiom: } \frac{A \text{ type} \quad s:A \quad t:A}{s =_A t \text{ . prop}}$$

$$\frac{t:A}{\text{refl}_t: t =_A t} \quad \text{mke}$$

$$\begin{aligned} \text{Elim: } & y:A \prod_{C:(\prod_{x:A} \prod_{y:A} ((x=_A y) \rightarrow \text{Type}))} \\ & (\prod_{x:A} C x x (\text{refl}_{xx})) \rightarrow \\ & \prod_{x:A} \prod_{y:A} \prod_{p:(x=_A y)} C x y p \end{aligned}$$

Computation:

$$y:A \quad C f t \quad t \quad \text{refl}_{A,t} \rightarrow f t$$

Equality type

$$\frac{u : B[s/x] \quad s \equiv t : A}{u : B[t/s]} \text{ substitution rule}$$

↖ so d h e n e e v a l u e s t, n o n e v a l u e

$$\frac{s : A \quad t : A \quad s \xrightarrow{*} u \quad t \xrightarrow{*} u}{s \equiv t : A}$$

SAT - solving (propositional logic)

$\Phi ::= x \mid \neg \Phi \mid \perp \mid \Phi \vee \Psi \mid \top \mid \Phi \wedge \Psi$
other connectives can be defined

We view formula Φ with ^{at most} k propositional variables $x_1, \dots, x_k$ as a function

$$\llbracket \Phi \rrbracket : \{0, 1\}^k \rightarrow \{0, 1\}$$

↑ semantic brackets

- Φ in variables $x_1, \dots, x_k$ is **satisfiable** if exists $\vec{b} \in \{0, 1\}^k$, such that the formula is true on that assignment of bits
 - formula is a **tautology** or **valid** if it's true for $\forall \vec{s} \in \{0, 1\}^k$. $\llbracket \Phi \rrbracket(\vec{s}) = 1$
 - Φ and Ψ are **equivalent** if for $\forall \vec{b} \in \{0, 1\}^k$. $\llbracket \Phi \rrbracket(\vec{b}) = \llbracket \Psi \rrbracket(\vec{b})$
- $\vec{b}$ is called **satisfying assignment**

Connections with the proof system

- Soundness theorem

if we can prove Φ (from no assumptions)
in natural deduction, then Φ is valid.

- completeness theorem

if Φ is valid we can prove Φ in
nat. deduction

(we need raa or law of excluded middle)

Φ is valid $\Leftrightarrow \neg\Phi$ is not satisfiable

Definitions give rise to "truth table"
algorithms, that check Φ for all $\vec{b} \in \{0, 1\}^k$

Proof system prove alternative algorithm for
validity: search for a proof

Negation normal form

Φ is in neg. normal form (NNF)
if the only negations appear directly in front
of propositional variables

$$\Phi ::= x \mid \neg x \mid \perp \mid \Phi \vee \Psi \mid \top \mid \Phi \wedge \Psi$$

Algorithm for conversion to nnf

$$\neg(\Phi \vee \Psi) \longrightarrow (\neg\Phi) \wedge (\neg\Psi)$$

$$\neg(\Phi \wedge \Psi) \longrightarrow \neg\Phi \vee \neg\Psi$$

$$\neg\neg\Phi \longrightarrow \Phi$$

$$\neg\perp \longrightarrow \top$$

$$\neg\top \longrightarrow \perp$$

(nondeterministic rewrite algorithm)

Theorem:

For any formula Φ , the rewrite algorithm terminates with the formula Φ^{nnf} that is equivalent to Φ .

Time complexity is $O(|\Phi|)$ ← size of Φ

also $|\Phi^{nnf}| = O(|\Phi|)$

A **literal** is a prop. variable x or the negation of x

A **clause** is a finite disjunction of literals

$L_1 \vee \dots \vee L_n$ or $\perp$

(A **coclause** is a finite conjunction of literals

$L_1 \wedge \dots \wedge L_n$ or $\top$)

A formula is in **conjunctive normal form** if it is a finite conjunction of clauses

$D_1 \wedge \dots \wedge D_n$ or $\top$ where D_i is a clause

A formula is in **disjunctive normal form**

$C_1 \vee \dots \vee C_n$

One can give an algorithm that converts

any formula Φ to Φ^{cnf} (notes)

or to Φ^{dnf}

The SAT problem

Input: is a prop. formula Φ

Output: yes if Φ is satisfiable and
no if Φ is not satisfiable

The useful SAT problem

Input: is a prop. formula Φ

Output: yes if Φ satisfiable + a satisfying assignment,
no if not satisfiable

Famously SAT is NP-complete

There is no known efficient in theory algorithm

The DNF SAT problem

Input: formula in DNF

Output: yes or no

There is an easy algorithm in $O(n^2)$

The CNF SAT problem (the useful one)

input: formula in CNF

output: yes or no

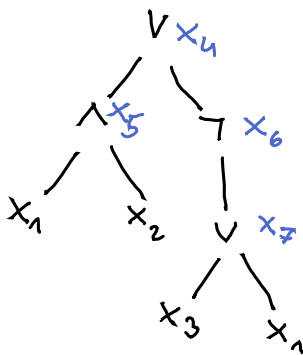
The Tseytin transformation

maps $\Phi(x_1, \dots, x_k)$ to $\Phi^T(x_1, \dots, x_k)$ where $k \geq k$ such that

- Φ^T is 3-CNF (each contains at most 3 literals)
- $(b_1, \dots, b_k) \mapsto (b_1, \dots, b_k): \{0, 1\}^k \rightarrow \{0, 1\}^k$ is a bijection satisfying assignments for Φ and satisfying assignments
- Φ^T is complete in $O(|\Phi|)$ time

Example

$$\Phi: (x_1 \wedge x_2) \vee \neg(x_3 \vee x_4)$$



$$x_4 \Leftrightarrow x_5 \vee x_6$$

$$x_5 \Leftrightarrow x_1 \wedge x_2$$

$$x_6 \Leftrightarrow \neg x_7$$

$$x_7 \Leftrightarrow x_3 \vee x_4$$

implication $\Rightarrow$ due implication $\Leftarrow$

$$1.) (\neg x_4 \vee x_5 \vee x_6) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_6 \vee x_4)$$

$$2.) (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_5)$$

$$3.) (\neg x_6 \vee \neg x_7) \wedge (x_7 \vee x_6)$$

$$4.) (\neg x_7 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_7)$$

$$5.) x_4 = 1) \wedge 2) \wedge 3) \wedge 4)$$

Example: 3-colorability as SAT problem

Let (V, E) be a graph (undirected, irreflexive)

Construct a formula Φ_G , with

prop variables $\{r_v, g_v, b_v \mid v \in V\}$

Φ_G is constituted as CNF formula
as a conjunction of clauses

for every $v \in V$ we include

1) $r_v \vee g_v \vee b_v$

2) $(\neg r_v \vee \neg g_v) \wedge (\neg g_v \vee \neg b_v) \wedge (\neg r_v \vee \neg b_v)$

lv sak v ime notanko eno barvo

for every (u, v) in E we include

3) $\neg r_u \vee \neg r_v \quad \neg g_u \vee \neg g_v \quad \neg b_u \vee \neg b_v$

Satisfying assignment for Φ_G are in 1:1 correspondence
with 3-coloring of G

DPLL(Φ, ρ) ρ assignment of truth variable

Repeat until is applicable

- if Φ is $[]$ return ρ as the satisfying assignment
- if Φ contains $()$
- if Φ contains a unit clause $L: \rho' := \rho, L$, DPLL(simplify(Φ, L), ρ')

if none of the above apply, we choose some literal L in Φ and continue with $\rho' := \rho, L$

$\Phi' = \text{simplify}(\Phi, L)$. continue DPLL(Φ', ρ')

if DPLL returns ρ'' then return ρ''

otherwise DPLL $\rho, \neg L$ and $\Phi' = \text{simplify}(\Phi, \neg L)$

$$[(p \vee \neg) \wedge (\neg p \vee r) \wedge (\neg g \vee \neg r) \wedge (\neg p \vee g \vee \neg r)]$$

choose p

$$[r \wedge (\neg g \vee \neg r) \wedge (g \vee \neg r)] \quad p$$

unit clause r

$$[\neg g \wedge g] \quad p, r$$

unit clause $\neg g$

$$[()] \quad p, r, \neg g$$

consider $\neg p$

$$[g \wedge (\neg g \vee \neg r)] \quad \neg p$$

unit clause g

$$[\neg r] \quad \neg p, g$$

unit clause $\neg r$

$$[] \quad \neg p, g, \neg r$$

Algorithmic problems in predicate logic

In predicate logic we have a signature: a set P of relation symbols or predicates, a set F of function symbols

P has array $i \geq 0$

f has carry ≥ 0

formales

$\Phi ::= \mathcal{P}(t, \dots, t) \mid t = t \mid \neg \Phi \mid \Phi \vee \Psi \mid \Phi \wedge \Psi \mid \forall x. \Phi \mid \exists x. \Phi$

$$t := \times | f(t_1, \dots, t_n) \quad \text{terms}$$

Example signature $(\leq, 0, +)$

$\leq$: relation
 0 : constant
 $+$: array 2

Example formula: $\forall x. 0 \leq x$

is true in natural interpretation of $\mathbb{N}$
it is false in $\mathbb{Z}$ and other interpretations in $\mathbb{N}$

A **sentence** is a formula with no free variables

Structures (Models)

A structure $\mathcal{M} = (M, (P^{\mathcal{M}})_{P \in \mathcal{P}}, (f^{\mathcal{M}})_{f \in \mathcal{F}})$

where M is set (domain or universe of quantification)

for every relation symbol P of arity n

$$P^{\mathcal{M}}: M^n \rightarrow \{\top, \perp\} = \{t, f\}$$

for every function symbol f of arity n

$$f^{\mathcal{M}}: M^n \rightarrow M$$

Example structure

$\mathcal{L}$

$$L := \{a, 1\}^*$$

↑ set of finite sequences

$$0^L := \varepsilon \leftarrow \text{empty sequence}$$

$$+ := w_1, w_2 \mapsto w_1 w_2 \text{ (concatenation)}$$

$$\leq := w_1, w_2 \mapsto \begin{cases} \top & \text{if } w_1 \text{ is prefix of } w_2 \\ \perp & \text{otherwise} \end{cases}$$

The sentence

$\forall x \forall y \ x+y = y+x$ holds in $\mathcal{N}$, but not in $\mathcal{L}$

↙ natural interpretation of $\mathcal{N}$

We write $\mathcal{N} \models \forall x \forall y \ x+y = y+x$

$$\mathcal{L} \not\models \forall x \forall y \ x+y = y+x$$

$\mathcal{M} \models \Phi \dots \mathcal{M} \text{ satisfies } \Phi$

The satisfaction relation $\mathcal{M} \models_p \Phi$
 $\models_p$ environment
 (function mapping variables to elements M)

$$\begin{array}{l} \mathcal{L} \models x \leq y \\ \begin{array}{l} x \mapsto 01 \\ y \mapsto 011 \end{array} \\ \\ \mathcal{L} \not\models x \leq y \\ \begin{array}{l} x \mapsto 000 \\ y \mapsto 0101 \end{array} \end{array}$$

$$\left| \begin{array}{l} x^p := p(x) \\ (f(t_1 \dots t_n))^p := f^{op}(t_1^p \dots t_n^p) \end{array} \right|$$

$$\mathcal{M} \models_p P(t_1 \dots t_n) \Leftrightarrow P^{op}(t_1^p, \dots, t_n^p) = \top$$

$$\mathcal{M} \models_p t_1 = t_2 \Leftrightarrow t_1^p = t_2^p$$

$$\mathcal{M} \models_p \Phi_1 \wedge \Phi_2 \Leftrightarrow (\mathcal{M} \models_p \Phi_1) \wedge (\mathcal{M} \models_p \Phi_2)$$

$$\mathcal{M} \models_p \forall x \Phi \Leftrightarrow \text{for all } a \in M, \mathcal{M} \models_{p[x \mapsto a]} \Phi$$

Huth and Ryan textbook

A sentence Φ is **valid** if for all structures $\mathcal{M}$
 $\mathcal{M} \models \Phi$

Soundness Theorem:

if sentence Φ has a proof in natural deduction, then Φ is valid

Completeness Theorem:

if Φ is valid, then it has proof in ND.

Φ is **satisfiable** if there exist a structure,
such that $\mathcal{M} \models \Phi$

The validity problem:

input: a sentence Φ

output: yes if Φ valid, no otherwise

Theorem (Church/Turing)

The validity problem is undecidable

A problem is decidable if there is an algorithm for it, that is guaranteed to terminate with the correct answer

it follows that the satisfiability problem is also undecidable

A practical algorithmic problem:

input: a sentence Φ and a purported ND proof of Φ ; in ND

output: yes if proof is correct, no otherwise

This is efficiently solvable

(used in proof assistance)

Consider following algorithm for validity

Algorithm (Φ)

Systematically search the space of all proofs
until we find a proof of Φ

if we find a proof, return yes

it never returns no $\therefore$

This algorithm shows that validity is
semidecidable (A yes no is **semidecidable**
if there exist an algorithm that always
correctly answers yes, if the answer is yes
but runs forever otherwise)

Satisfiability is *cosemi: decidable*

cosemi: decidable means that it answers no if false, otherwise runs forever

Modeling a computer network

Signature: Connected / 2 ; server, client / 1

$$\forall x. \neg \text{Connected}(x, x)$$

$$\forall x, y. \text{connected}(x, y) \rightarrow \text{connected}(y, x)$$

$$\forall x. \text{server}(x) \vee \text{client}(x)$$

$$\forall x. \neg \text{server}(x) \vee \neg \text{client}(x)$$

$$\forall x. \text{client}(x) \rightarrow \exists y. \text{server}(y) \wedge \text{connected}(x, y)$$

$$\forall x. \text{server}(x) \rightarrow \exists y. \text{client}(y) \wedge \text{connected}(x, y)$$

$$\forall x, y. \text{connected}(x, y) \rightarrow \neg \text{server}(x) \vee \neg \text{server}(y)$$

It is consistent to have network with ^{at least} 10 ~~services~~ ^{nodes} such that exactly 2 are server

$$\exists x_1, \dots, x_{10} \\ \bigwedge_{1 \leq i < j \leq 10} x_i \neq x_j$$

$$\wedge \exists x \exists y. x \neq y \wedge \forall z. \text{server}(z) \rightarrow z = x \vee z = y \\ \wedge \text{server}(x) \wedge \text{server}(y)$$

Finite satisfiability

Question: sentence Φ

Answer: yes if exists finite structure $\mathcal{M}$, such that $\mathcal{M} \models \Phi$ and no otherwise

Finite satisfiability is semidecidable

Algorithm: for every size $n=1, 2, \dots$ we enumerate all non-isomorphic structures of size n and for each structure $\mathcal{M}$ we look at the model checking problem $\mathcal{M} \models \Phi$, if and when we find such a structure, we return that structure

24.3 wisdom ~~page~~ p. 24

SMT-solving

31.3

Satisfy modulo a theory

SAT	prop. logic	not very expressive but solving is practical
FOL	...	good expressivity but solving is impractical (but not entirely)

SMT the sweet spot in the middle. Expressive enough and satisfiability is still practical

SMT basic setting:

First order signature: function symbols + predicates

Typically we are interested in the question:

Is Φ satisfiable modulo a theory T ?

For our purposes a theory T is determined by a set of structures Mod_T

We are interested in the question

$\exists M \in \text{Mod}_T. \exists \text{environment } \rho. M \models_{\rho} \Phi$?

SMT is useful for program verification.

Construct $\phi(x_1, \dots, x_n)$ saying $x_1, \dots, x_n$ are inputs and with the program behaves incorrectly. A satisfying assignment to ϕ finds a bug in input.

If unsatisfiable, there are no bugs

Interesting theories:

EUF: equality between uninterpreted functions
signature: just function symbols

Flatten formula: no nesting of function symbols,
every term named by a variable

Perform congruence closure

check consistency of the resulting congruence
closure with negated equality

$$x=y \wedge x=g(y) \wedge u=f(x, g(x)) \wedge v=f(y, x) \wedge u \neq v$$

$$x=y \wedge x=g(y) \wedge t=g(x) \wedge u=f(x, t) \wedge v=f(y, x) \wedge u \neq v$$

$[x, y, t, u, v]$... partition of all variables into
separate equivalence classes

$$[x, y, t, u, v]$$

$$[x, y, t, u, v]$$

$$[x, y, t, u, v] \Rightarrow \text{formula is unsatisfiable}$$

• Have algorithm congruence closure decide
satisfiability for clauses (conjunctions of literals)
(practical)

• So we ^{can} decide satisfiability arbitrary quantifier
free formulas by reduction to DNF (impractical)

Anja Petković

Theory of arrays:

Two functions: $\text{write}(a, i, v)$, $\text{read}(a, i)$

Theory: $i=j \rightarrow \text{read}(a, i) = \text{read}(a, j)$

$i=j \rightarrow \text{read}(\text{write}(a, i, v), j) = v$

$i \neq j \rightarrow \text{read}(\text{write}(a, i, v), j) = \text{read}(a, j)$

Reducing array to EUF

$i \neq j \wedge u = \text{read}(\text{write}(a, i, v), j) \wedge v = \text{read}(a, j) \wedge u \neq v$

$i \neq j \wedge u = (i=j ? v : \text{read}(a, j)) \wedge v = \text{read}(a, j) \wedge u \neq v$
 $\quad \quad \quad \uparrow \quad \quad \uparrow \quad \quad \uparrow$
 $\quad \quad \quad \text{cc} \quad \text{ptr} \quad \text{size}$

$i \neq j \wedge u = \text{read}(a, j) \wedge v = \text{read}(a, j) \wedge u \neq v$

$[i | j | u | v]$ Now we perform satisfiability test

$[i | j | u, v]$ unsatisfiable!

What we aim for with SMT?

We have a theory, that supports efficient satisfiability for clauses. We want to bootstrap to arbitrary quantifier free formulas

2 approaches:

- Eager: Translate to a propositional set problem
- Lazy: Modify SAT solving algorithm based on DPLL.
To find contradiction we use clause decision procedure

Start with a flatten formula

$$x = f(y) \wedge y = f(x) \wedge x = y$$

replace terms with variables

$$x = f^y \wedge y = f^x \wedge x = y \wedge (x = y \rightarrow f^x = f^y)$$

We obtain in the end a first order formula, where all atomic formulas are equalities between variables

Let k be the smallest number, such that $2^k \geq$ number of variables.

Bit blasting: replace every variable x with k propositional variables $x_1, \dots, x_k$

Replace every equality $x = y$ with

$$x_1 \leftrightarrow y_1 \wedge x_2 \leftrightarrow y_2 \wedge \dots \wedge x_k \leftrightarrow y_k$$

$$x \neq y \text{ with } \neg(x_1 \leftrightarrow y_1 \wedge \dots \wedge x_k \leftrightarrow y_k)$$

We submit to SAT solver.

This will be satisfiable $\Leftrightarrow$ it's unsatisfiable

Start with flatten formula $x = f(y), y = f(x), x \neq y$

$$x = f^y \wedge y = f^x \wedge x \neq y \wedge (x = y \rightarrow f^x = f^y)$$

$k=2$

$$(x_1 \leftrightarrow f_1^y) \wedge (x_2 \leftrightarrow f_2^y) \wedge (y_1 \leftrightarrow f_1^x) \wedge (y_2 \leftrightarrow f_2^x) \wedge$$

$$\neg(x_1 \leftrightarrow y_1 \wedge x_2 \leftrightarrow y_2) \wedge ((x_1 \leftrightarrow y_1 \wedge x_2 \leftrightarrow y_2) \rightarrow$$

$$f_1^x \leftrightarrow f_1^y \wedge f_2^x \leftrightarrow f_2^y)$$

linear arithmetic

Terms $0 + 1 \ a \cdot x$

$\leftarrow$ integer/natural/real/rational....

$\leq$

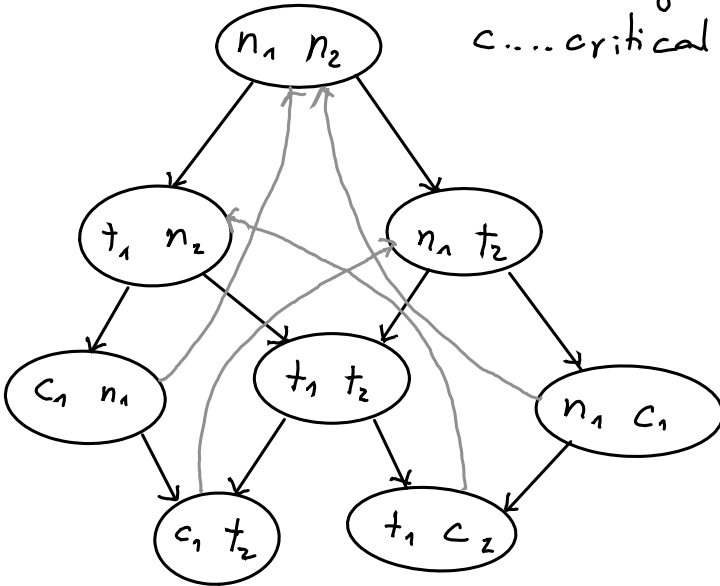
Model Checking

Today: Linear Time Temporal Logic (LTL)

Further reading: HR, chapter 3

Example of transition
system:

n ... noncritical
 t ... trying
 c ... critical



Example **safety property** (the system always stays in safe state)

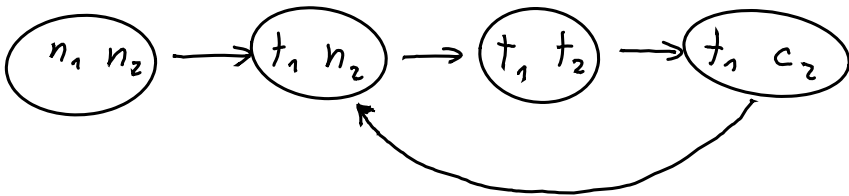
↙ global no ↙ 1 in 2 must critical no enter
 $G(\neg(c_1 \wedge c_2))$ trivially holds

We know also **liveness property** (something desirable will eventually happen)

$$G((t_1 \rightarrow F c_1) \wedge (t_2 \rightarrow F c_2))$$

(œ 1 posküzä dobiti dostop, ga bo enkrat dobile)

This does not hold. counter run:



This is called „lasso” run

A transition system is given by

$$(S, \rightarrow, L, \text{Atoms})$$

where S is finite set (of states)

$\rightarrow \subseteq S \times S$ (the transition relation)

$L: S \rightarrow \mathcal{P}(\text{Atoms})$ maps state to its atoms

The labeling function

powerset of atoms



$$\text{Atoms} = \{n_1, n_2, t_1, t_2, c_1, c_2\}$$

now and in the future Φ holds until ψ holds

Φ holds globally

LTL formulas:

$$\Phi ::= p \mid \neg \Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid X\Phi \mid F\Phi \mid G\Phi \mid \Phi U \Psi$$

$\bigwedge$
Atoms

Φ holds in next
 Φ holds in future

in textbook also $|\Phi R \Psi| \Phi W \Psi$

Requirements for LTL

$$\forall s \in S. \exists s' \in S. s \rightarrow s'$$

The satisfaction relation takes the form

$$\pi \models \phi$$

Here π is infinite run through transition system

$$\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$$

satisfaction relation

$$\pi \models p \iff p \in L(s_1) \quad (p \in L(\pi(1)))$$

$$\pi \models \neg \phi \iff \pi \not\models \phi$$

$$\pi \models \phi \wedge \psi \iff \pi \models \phi \wedge \pi \models \psi$$

$$\pi \models X\phi \iff \pi^2 \models \phi$$

π^2 ϕ holds in run started on second

$$\pi \models G\phi \iff \forall n. \pi^n \models \phi$$

$$\pi \models F\phi \iff \exists n. \pi^n \models \phi$$

$$\pi \models \phi \mathcal{U} \psi \iff \exists n. \pi^n \models \psi \wedge \forall m < n. \pi^m \models \phi$$

$$\pi^n := s_n \rightarrow s_{n+1} \rightarrow \dots$$

Model checking problem

Input: A transition system $\mathcal{M}$

A system state s_0

An LTL formula ϕ

Output:

Yes, if $\forall$ runs π through $\mathcal{M}$ starting at s_0 , we
have $\pi \models \phi$ $m, s_0 \models \phi$

No, since $\exists$ an infinite run π that $\not\models \phi$

The liveness property for mutual exclusion
 $G((t_1 \rightarrow \neg C_1) \wedge t_2 \rightarrow \neg C_2))$

DM: modify the model where this holds

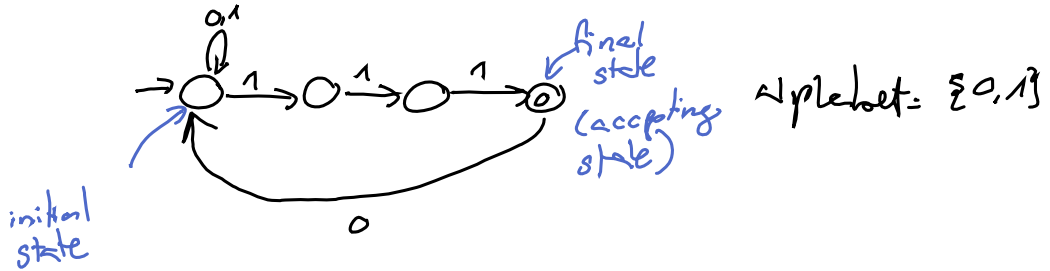
Alternative: Assume fairness property

A fairness assumption:

$$\phi_{\text{fairness}} = GF(t_1 \wedge t_2) \rightarrow GF(C_1) \wedge GF(C_2)$$

Model checking formula: $\phi_{\text{fairness}} \rightarrow \phi_{\text{liveness}}$

Non deterministic Büchi automata



A Büchi automaton defines a language of infinite words

The set of infinite words is written Σ^ω

Examples of ω -words

000000...

010101...

...

infinite word is accepted, if there is some run of the automaton, labeled by the word, that passes accepting state infinitely times

Model checking problem for LTL

input: Transition system $\mathcal{M}$, starting state s , LTL formula ϕ

Output: yes, if $\mathcal{M}, s \models \phi$,
no, otherwise + also run Π such that $\Pi \models \neg \phi$

Language theory alphabet Σ , word is a finite sequence from Σ (Σ^* for the set of words)

A language is a set of words, $L \subseteq \Sigma^*$

Language theory for ω -words

ω -word is an infinite sequence from Σ (Σ^ω for the set of ω -words)

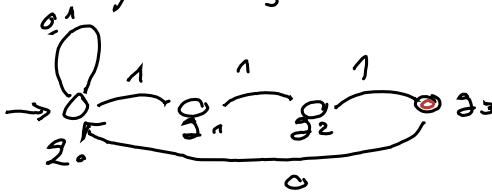
An ω -language is a set of ω -words $L \subseteq \Sigma^\omega$

An automaton on ω -words specifies an ω -language

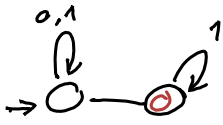
Büchi automata is a particular automata on ω -words

$\Sigma = \{0, 1\}$ A_1 will specify:

$L(A_1) = \{w \in \{0, 1\}^\omega ; w \text{ contains substring } 1110 \text{ infinitely often}\}$



A_2 :



$L(A_2) = \{w \in \{0, 1\}^\omega \mid \text{from some point there are only ones. There are finitely many zeros}\}$

A_3 :



$L(A_3) = \{1\}$

A Büchi automata is formally a tuple

(Q, I, Δ, F) A

$\uparrow$ set of states
 $\uparrow$ set of initial states $I \subseteq Q$
 $\uparrow$ set of final states $F \subseteq Q$

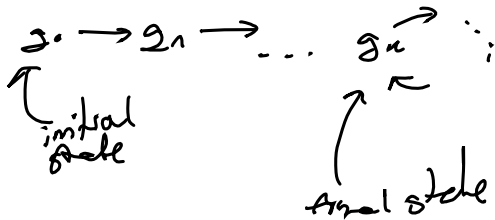
$\Delta \subseteq Q \times \Sigma \times Q$
 transition relation

nondeterministic Büchi
automata

Proposition (Empty testing for an NBA)

Let A be an NBA

$L_w(A) \neq \emptyset \Leftrightarrow$ there exists a path of the form



Such a path is called an accepted loop

Definition: An ω -language $L \subseteq \Sigma^\omega$ is ω -regular
if there exists NBA A such that $L_w(A) = L$

Theorem: ω -regular languages are closed

• under union

• intersection

• complement

(Büchi's theorem)

Consider an CTL formula a and

consider when does it hold, that a run satisfies

$$\pi \models a$$

$$\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots$$

The Trace of a run $\pi = s_1 \rightarrow s_2 \rightarrow \dots$

is the sequence $L(s_1), L(s_2), L(s_3), \dots$ ^{label function}

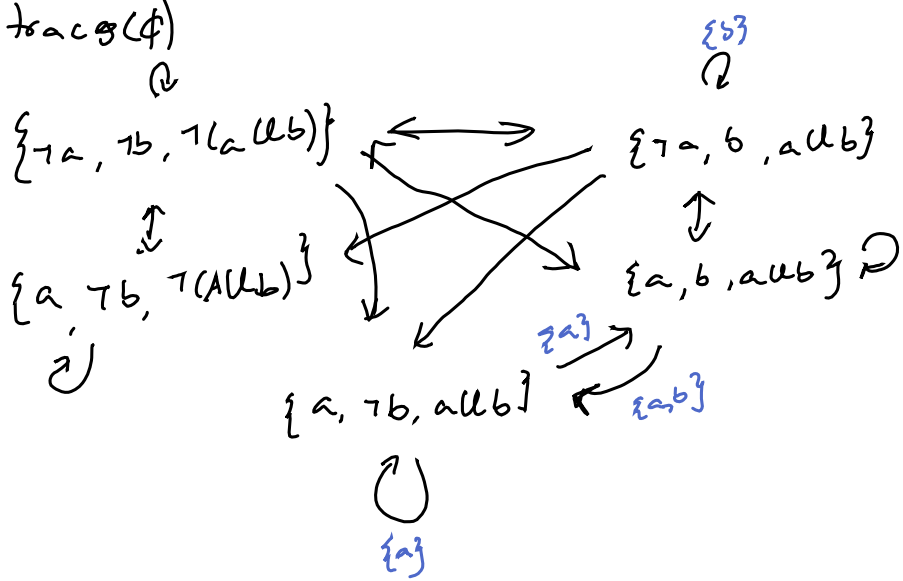
This is an ω -word in $\mathcal{P}(\text{Atoms})^\omega$

The truth of a formula ϕ on a run π is completely determined on the trace

$$\text{Traces}(\phi) := \{w \in \mathcal{P}(\text{Atoms})^\omega; \phi \text{ is true on any run with } w\}$$

Theorem: For any ϕ , $\text{Traces}(\phi)$ is w-regular.

There is an algorithm to construct A_ϕ recognizing $\text{traces}(\phi)$



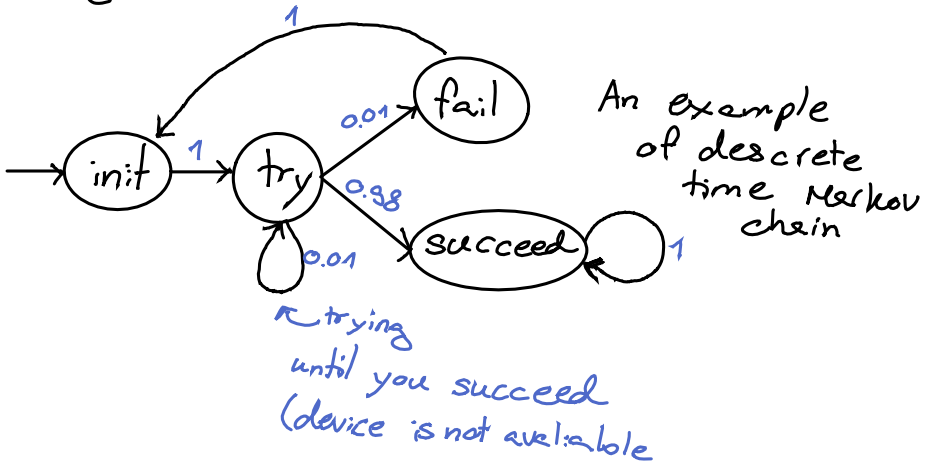
Model checking algorithm

- step 1: construct the automaton $A \models \phi$
- step 2: compute $A \models \phi$

Probabilistic Model Checking

In general:

- many probabilistic models (today we focus on DTMCs ← discrete time Markov chain)
- many probabilistic logics (PCTL, PCTL*)
(we are going to use LTL)
- There is a whole technology of associated algorithms



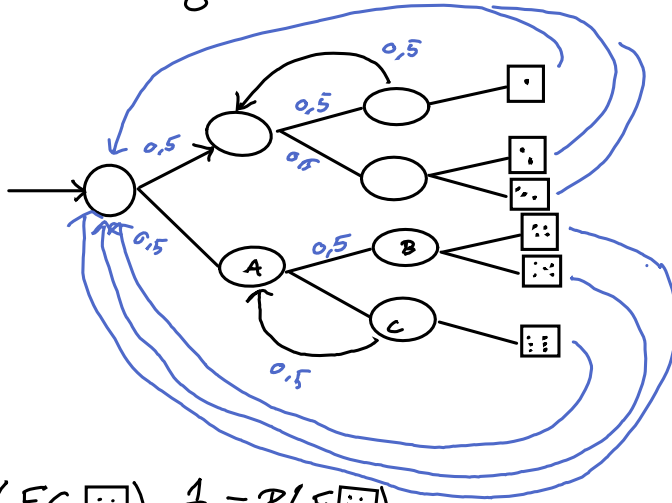
$$P(\text{init}) = 1 \quad P(X \text{ try}) = 1 \quad P(XX \text{ succ}) = 0.98$$

$$P(X(\text{try} \wedge X(\text{try} \wedge X \text{ try}))) =$$

$$= P(XX(\text{try} \wedge X \text{ try})) = P(XXX \text{ try}) = 0.0001$$

$$P(FG \text{ succ}) = 1 - \underbrace{(0.01)^n}_{\tau_{try}} \underbrace{(0.01)^m}_{\tau_{fail}} \xrightarrow{n+m \rightarrow \infty} 1$$

Knuth/Yao-algorithm for a fair dice



$$P(FG \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}) = \frac{1}{6} = P(F \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix})$$

modified: $P(Ou \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}) = \frac{1}{6}$

$$P(Ou \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \wedge X(Ou \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix})) = \frac{1}{36}$$

$$P_B = 0 \quad P_C = \frac{1}{2} + \frac{1}{2} P_A \quad P_A = \frac{1}{2} P_B + \frac{1}{2} P_C =$$

$$= \frac{1}{4} + \frac{1}{4} P_A \quad \Rightarrow P_A = \frac{1}{3}$$

General solution:

We have DTMC and LTL formula Φ

Question: What is the probability that a randomly generated run π satisfies Φ

Two issues:

1. Make sense of the question
2. Compute the answer

A **DTMC** is (S, s_{init}, P, L)

$\uparrow$ set of states $\nwarrow$ transition matrix $\swarrow$ labeling function

- S set of states (in modeling is normally finite)
- $s_{init} \in S$ the initial state
- L labeling function: $S \rightarrow \mathcal{P}(\text{Atoms})$
- $P: S \times S \rightarrow [0, 1]$ **transition probability matrix**
satisfying $\forall s \in S. \sum_{t \in S} P(s, t) = 1$

for a formula Φ we want to calculate

$$\mathcal{P}[\Phi] := \mathcal{P}(\{\pi \in \text{Runs} \mid \pi \models \Phi\})$$

We assign probability to measurable sets of runs

A **cylinder set** is determined by a finite run

$$w = s_1 \xrightarrow{p_1} s_2 \xrightarrow{p_2} \dots \xrightarrow{p_n} s_n \quad s_i = s_{\text{init}} \quad p_i > 0$$

$$\langle w \rangle := \{\pi \in \text{Runs} \mid \pi \text{ extends } w \text{ (} w \text{ is prefix of } \pi)\}$$

$$\mathcal{P}(\langle w \rangle) = \prod_{i=1}^{n-1} p_i$$

The measurable subset of Runs are defined by

- Every cylinder set $\langle w \rangle$ is measurable
- If $R_1, R_2, \dots$ are measurable sets then so are $\bigcup_n R_n$, $\bigcap_n R_n$
- If R is measurable, then so is R^c
- $\emptyset$ and Runs are measurable

A general fact: Let $\mathcal{E}$ be the set of all measurable subset of $\mathbb{R}^n$. There is a unique probability function $\mathbb{P}: \mathcal{E} \rightarrow [0, 1]$ satisfying

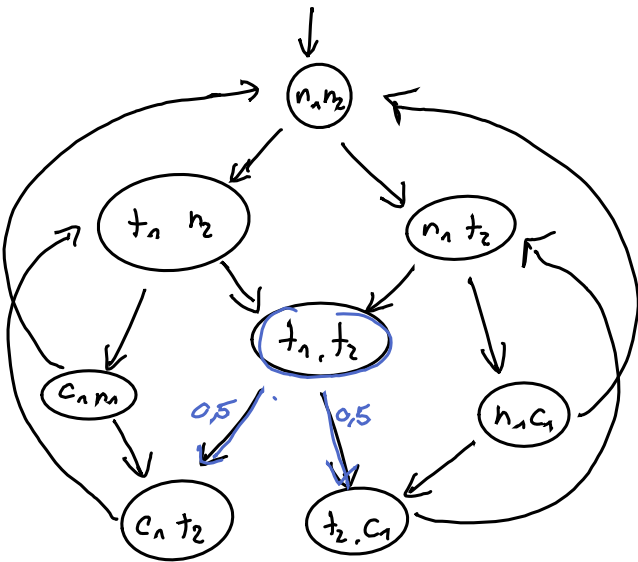
P is a probability measure

- $P \emptyset = 0$
- $P(W)$ is as defined earlier
- If we have $R_1, R_2, \dots$ pairwise disjoint measurable sets then $P(\cup_n R_n) = \sum_n P R_n$
(Countable additivity)

Proposition: For any formula Φ , the set $\{\pi \in \text{Runs} \mid \pi \models \Phi\}$ is measurable.

The proof uses Büchi automata

Model checking: Given (S, s_{init}, P, L) and ϕ
 answer $P_g \phi = \text{is } P(\phi) \geq g$



This is a Markov decision process (MDP)
is combining probability and determinism

Liveness: $G(t_1 \rightarrow FC_1) \wedge (t_2 \rightarrow FC_2)$

A Markov run would be

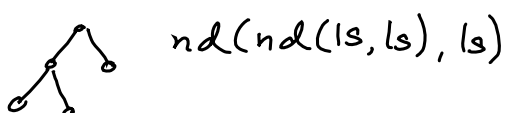
$$(n_1, n_2) \rightarrow n(t_1) \begin{cases} \xrightarrow{0.5} (c_1, t_2) \rightarrow \textcircled{h_1, t_2} \dots \\ \xrightarrow{0.5} (c_2, t_1) \rightarrow \bigcirc \dots \end{cases}$$

lives: $G(t_1 \rightarrow FC_1) \wedge (t_2 \rightarrow FC_2)$ has probability 1 over all marker chains

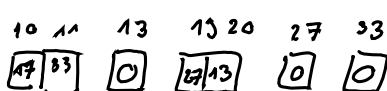
V

mission de sem
manjela 28.4

Binary tries (linked trees in heap memory)



$nd(nd(ls, ls), ls)$



$\hookrightarrow$ o predstavlja list

$\begin{bmatrix} 1 & 2 \\ 5 & 5 \end{bmatrix} \quad \begin{bmatrix} 5 \\ 0 \end{bmatrix} \quad \wedge$ tega ne bomo devolili:

$ltree(t, i)$... linked tree. i is a reference to a linked binary tree in heap memory and the implemented tree is the mathematical tree t

$ltree(nd(nd(lf, lf), lf), 10)$
 $\hookrightarrow$ tu je root

$\neg ltree(nd(lf, lf), 1)$

$ltree$ is defined inductively on the structure of the mathematical tree t and moreover the implemented tree is the only defined part of heap memory

$ltree(lf, i) \iff i \mapsto 0$

$ltree(nd(t_1, t_2), i) \iff \exists j, k. (i \mapsto j) * (i+1 \mapsto k) * ltree(t_1, j) * ltree(t_2, k)$

Program: gre ~~del~~ dravo in ga izbriše

$\{\exists t. ltree(t, i)\}$

def delTree(i)

if $[i] = 0$:

dispose(i)

else:

$j := [i]$

$k := [i+1]$

delTree(j)

delTree(k)

dispose(i)

dispose($i+1$)

$\{\text{emp}\}$

$s, h \models \text{emp} \iff h = \emptyset$

$\{\exists t. ltree(t, i) \wedge [i] = 0\}$

$\{i \mapsto 0\}$ implied

$\{\text{emp}\}$

$\{\exists t. ltree(t, i) \wedge [i] \neq 0\}$

$\{\exists j, k. [i] = j * [i+1] = k * ltree(t_1, j) * ltree(t_2, k)\}$

$ltree(t_1, j) * ltree(t_2, k)$

$\{i \mapsto j * i+1 \mapsto k * ltree(t_1, j) * ltree(t_2, k)\}$

$\{i \mapsto j * i+1 \mapsto k * \text{emp} * ltree(t_2, k)\}$

$\{i \mapsto j * i+1 \mapsto k * \text{emp} * \text{emp}\}$

$\{\text{emp} * \text{emp} * \text{emp} * \text{emp}\}$

$\{\text{emp}\}$

small rule: $\frac{}{\{h \mapsto -\} \text{dispose}(h) \{ \text{end} \}}$

$\{\Phi[E_1/i]\} \text{proc}(E_1) \{\Psi[E_1/i]\} \quad \{\Phi[E_n/i]\} \text{proc}(E_n) \{\Psi[E_n/i]\}$

$\vdots$

$\{\Phi\} \subset \{\Psi\}$

$\{\Phi\} \text{proc}(i) : \subset \{\Psi\}$

$\{\Phi\} \subset \{\Psi\}$

$\{\Phi * \Theta\} \subset \{\Psi * \Theta\}$

free variables of Θ

$FV(\Theta) \cap MV(C) = \emptyset$

$\hookrightarrow$ set of variables that C modifies

Machine configuration: state + heap

function $\nearrow$ finite partial function $\mathbb{N} \rightarrow \mathbb{Z}$

Assertions ϕ state property of configurations
with $s, h \models \phi$ (s, h asserts ϕ)

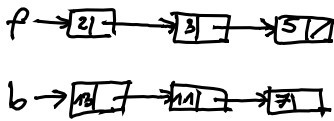
$s, h \models n \mapsto v \Leftrightarrow h = \{(n, v)\}$ $\nwarrow$ partial function (as a graph)

$s, h \models \phi * \psi \Leftrightarrow \exists h_1, h_2. h = h_1 \oplus h_2$ and $s, h_1 \models \phi$ and $s, h_2 \models \psi$ $\nwarrow$ disjoint union

Resource analysis with separation logic

Amortised time analysis:

Functional queues



$\text{make_queue}: O(1)$ $\text{enqueue}(x)$ nejde v seznamu;
 $f := 0$ $i := \text{allocate}(2)$ Damo
 $b := 0$ $[i] := x$ ne konec
 $[i+1] := b$ vzate
 $b := i$

Dequeue

if $f = 0$:
 $\text{then reverse}(b)$ and put it in f $O(n)$
 $b := 0$
 else: skip

if $f \neq 0$: (if it is still zero?):

 then error

 else $v := [f]$;

$k := [f+1]$

 dispose f $\leftarrow \text{debaramo seznamu}$

$f := k$

 return v

$\text{Dequeue}: O(1)$ if $f \neq 0$
 $O(n)$ if $f = 0$

$$\frac{T_{\text{total}}}{N} \leq O(1) \quad \text{for every } N$$

Extended separation logic with resource credits

$\{1\epsilon\} \text{ Makequeue } \{ \text{queue}(f, b) \}$

$\text{queue}(f, b) \Leftrightarrow \text{flist}(f) * \text{blist}(b)$

$\text{flist}(f) \Leftrightarrow ((f=0 \wedge \text{emp} \wedge 0\epsilon) \vee$
 $(f > 0 \wedge \exists v, g [f \mapsto v] * [f+1 \mapsto g] * \text{flist}(g)))$
 $\wedge 0\epsilon$

$\text{blist}(b) \Leftrightarrow (b=0 \wedge \text{emp} \wedge 0\epsilon) \vee$

$(b > 0 \quad b \quad b+1 \quad \text{blist}(g) * 1\epsilon)$

$\{2\epsilon * \text{queue}(f, b)\} \text{ Enqueue } \{ \text{queue}(f, b) \}$

$\{1\epsilon * \text{queue}(f, b) \wedge (f=b=0)\} \text{ Dequeue } \{ \text{queue}(f, b) \}$

$$\{f = 0 \wedge b \text{ list } b\}$$

$$\{f \text{ list } (f) * b \text{ list } (b)\}$$

while $b \neq 0$

$$\{(f \text{ list } (f) * b \text{ list } (b)) \wedge b \neq 0\}$$

$$\{f \text{ list } (f) * \exists v, g. [b \mapsto v] * [b+1 \mapsto g] * 1 \in * b \text{ list } (g)\}$$

tick

$$\{\text{first}(f) * \exists v, g. [b \mapsto v] * [b+1 \mapsto g] * b \text{ list } (g)\}$$

$$k := [b+1]$$

$$\{\exists v. f \text{ list } (f) * [b \mapsto v] * [b+1 \mapsto f] * b \text{ list } (k)\}$$

$$\{f \text{ list } (b) * b \text{ list } (k)\}$$

← implies

$$f := b$$

$$\{\exists v. \text{first}(f) * b \text{ list } (k)\}$$

$$b = k$$

$$\{f \text{ list } (f) * b \text{ list } (b)\}$$

$$\{b = 0 \wedge (f \text{ list } (f) * b \text{ list } (b))\}$$

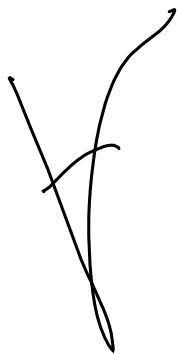
$$\{f \text{ list } (f) \wedge b = 0\}$$

Soundness and completeness of λ -calculus

Hoare triple: $\{\phi\} \subset \{\psi\}$
 $\uparrow$ commands

$$C ::= x := E \mid \text{if } B \text{ then } C \text{ else } C \mid c; c \mid \text{skip} \mid \text{while } B \text{ do } C$$
$$E ::= n \mid x \mid \neg E \mid E + E \mid E * E \mid \dots$$
$$B ::= E = E \mid E < E \mid B \vee B \mid B \wedge B \mid \neg B$$

$\vdash_{\text{par}} \{ \phi \} \subset \{ \psi \} \iff \text{there exists a proof of the specification } \{ \phi \} \subset \{ \psi \}$



For any command C and postcondition ψ ,
we define $wp(C, \psi)$ by:

$wp(C, \psi) \Leftrightarrow$ if the execution of program C terminates, then the resulting state satisfies ψ

Lemma (weakest precondition)

1. $\models_{pr} \{wp(C, \psi)\} C \{ \psi \}$
2. $\models_{pr} \{ \phi \} C \{ \psi \} \Rightarrow \mathbb{Z} \models \phi \rightarrow wp(C, \psi)$

Lemma (expressive completeness)

The property $wp(C, \psi)$ can be explained by a formula in our assertion logic

Very quick outline of proof:

Prove by induction on C that $\models_{pr} \{ \phi \} C \{ \psi \}$

$\Rightarrow \models_{pr} \{ \phi \} C \{ \psi \}$.

Eg, in the case of while B do C .

Suppose $\models \{ \phi \} \text{ while } B \text{ do } C \{ \psi \}$

...

Poslužaj:

- SAT
- Hoare logic

3rd exams

• no type theory

present for 10 minutes
← choose

easy and hard topics

- (10) • LTL specification for DTMCs
- (12-13) • separation logic
- (last) • completeness theorem for
Hoare logic

- SAT solving and
DPLL algorithm
- Predicate logic and bounded model checking
- SMT solving
- (• LTL for transition systems) - excited for me
- Hoare logic

Sign up for exams

8.6 - kon.ec.6

