SNT — Simulační nástroje a techniky

Petr Peringer
peringer AT fit.vutbr.cz

Vysoké učení technické v Brně,
Fakulta informačních technologií,
Božetěchova 1, 612 66 Brno

(Verze 2025-02)
Přeloženo 7. února 2025

Úvod

Text je určen pro studenty FIT. Obsahuje přehled problematiky simulačních nástrojů, technik, metod, algoritmů a teoretických přístupů vhodný pro studenty magisterského studia, kteří zvládli základy modelování a simulace v bakalářském předmětu IMS, některý z vyšších programovacích jazyků (C, C++, ...) a mají odpovídající matematické znalosti.

Obsah slajdů je velmi stručný, podrobnější informace jsou součástí výkladu. Předpokládá se též samostatná práce studentů s literaturou.

Pravidla

- přednášky
- konzultace
- samostatná práce — projekt (a různé domácí úkoly)

Hodnocení celkem 100b:

- 30b projekt
- Zápočet: alespoň polovina z výše uvedených bodů
- 70b zkouška (minimum=30b)

Zdroje informací

- Literatura
- WWW odkazy
- Oficiální stránka:
  `https://www.fit.vut.cz/study/course/SNT/`
- Aktuální informace pro studenty:
  `..../SNT/public/`
- Vlastní simulační experimenty
- ...

Literatura

- Cellier F., Kofman E.: *Continuous System Simulation*, Springer, 2006
- Nutaro J.: *Building Software for Simulation*, Willey, 2011
- Rábová a kol.: *Modelování a simulace*, skriptum VUT, Brno, 1992
- Ross S.: *Simulation*, 3rd edition, Academic Press, 2002
- Zeigler B., Praehofer H., Kim T.: *Theory of Modelling and Simulation*, 3rd edition, Academic Press, 2019

Modelování systémů na počítačích

Opakování – viz materiály k předmětu IMS:

- Základní pojmy a principy
- Souvislosti a použití
- Výhody, problémy
- Alternativy

SNT:

- Teorie systémů — DEVS
- Algoritmy řízení simulace
- Přehled simulačních nástrojů
- Případové studie

Základní pojmy — opakování

- Systém, Model, Modelování, Simulace
- Princip modelování a simulace:
  Realita → *Znalosti* → *Abstraktní model* → *Simulační model*
- *Experimentování s modelem* → *Analýza výsledků*
- Cílem simulace je získat nové znalosti o modelovaném systému.
- Základní etapy modelování a simulace.
- Klasifikace systémů a modelů.
- Základní algoritmy: *Next-event*, numerické integrační metody, řazení funkčních bloků, metoda Monte Carlo, ...

Teoretické základy modelování

Definice základních pojmů:

- Časová množina
  - *Reálný/fyzikální čas*
  - *Modelový čas*
  - *Strojový čas*
- Systém, prvek systému
- Okolí systému
- Chování systému
- Model systému
- Simulátor
- Simulace

Čas – definice

Čas je chápán jako nezávislá veličina

$$time = (T, <)$$

kde

- T je množina
- $<$ je *lineární uspořádání* nad T. To znamená že pro každou dvojici (t, t') buď $t < t'$ nebo $t' < t$ nebo $t = t'$.

Relace uspořádání dovoluje používat termíny jako *minulost, budoucnost* a *přítomnost*.

Časové intervaly

$(t_1, t_2) = \{\tau | \tau \in T, t_1 < \tau < t_2\}$
$[t_1, t_2] = \{\tau | \tau \in T, t_1 \leq \tau \leq t_2\}$
$(t_1, t_2] = \{\tau | \tau \in T, t_1 < \tau \leq t_2\}$
Zápis $\langle t_1, t_2 \rangle$ znamená libovolný z uvedených intervalů.
Někdy se pro časové intervaly používá označení $T_{\langle t_1, t_2 \rangle}$.

Poznámka: Pro účely modelování je možné omezit časovou základnu na podmnožinu reálných čísel $\mathcal{R}$. Tyto časové základny mají definovánu operaci $+$, která zachovává uspořádání a $(T, +)$ má vlastnosti abelovské grupy. Časová základna $\mathcal{R}$ reprezentuje *spojitou časovou základnu*, všechny časové základny isomorfní s množinou celých čísel $\mathcal{I}$ reprezentují *diskrétní časovou základnu*.

Časové funkce, signály

Je-li T časová základna, potom všechny funkce

$$f : T \to A$$

kde A je libovolná množina, nazýváme *časové funkce* nebo *signály*.

Hodnotu funkce f v čase t označíme $f(t)$ případně f_t.
Restrikci funkce f na množinu $T' \subset T$ definujeme jako funkci f/T':

$$f/T' : T' \to A, \quad f/T'(t) := f(t) \forall t \in T'$$

která je také časovou funkcí.

Segmenty

Zvláštní význam mají restrikce nad intervaly.
Restrikce f na intervalu $\langle t_1, t_2 \rangle$ se nazývá *segment* nebo *trajektorie* a obvykle se zapisuje:

$$\omega : \langle t_1, t_2 \rangle \rightarrow A$$

nebo

$$\omega_{\langle t_1, t_2 \rangle}$$

Dva segmenty ω_1, ω_2 jsou *sousedící (contiguous)*, když jejich domény na sebe navazují.

Segmenty — konkatenace

Pro spojité segmenty je definována operace *konkatenace* $\bullet$:

$$\omega_1 \bullet \omega_2 : \langle t_1, t_3 \rangle \to A$$

kde $\omega_1 \bullet \omega_2(t) = \omega_1(t)$ pro $t \in \langle t_1, t_2 \rangle$ a $\omega_1 \bullet \omega_2(t) = \omega_2(t)$ pro $t \in \langle t_2, t_3 \rangle$.

Množina segmentů Ω nad A a T je uzavřená vzhledem ke konkatenaci jestliže pro každou dvojici $\omega_1, \omega_2 \in \Omega$ také $\omega_1 \bullet \omega_2 \in \Omega$.

Segmentace

Segment $\omega_{t\rangle}$ nebo $\omega_{\langle t_1,t\rangle}$ nazveme *levý segment* ω.

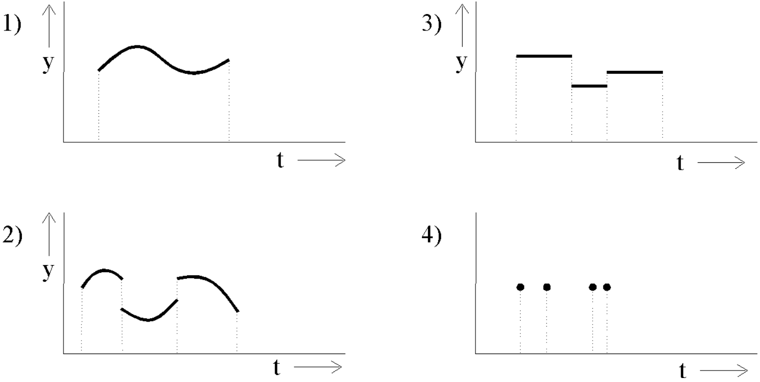
Poznámka: levý segment = minulost

Množina segmentů Ω nad A a T je uzavřená vzhledem k levé segmentaci jestliže pro každé $\omega : \langle t_1, t_2\rangle \to A \in \Omega$ a $t \in \langle t_1, t_2\rangle$ také $\omega_{t\rangle} \in \Omega$.
Podobně lze definovat pravou segmentaci.

Klasifikace segmentů

1. Spojitý segment $\omega : \langle t_1, t_2 \rangle \rightarrow \mathcal{R}^n$ nad spojitou časovou základnou musí být spojitý ve všech bodech $t \in (t_1, t_2)$
2. Po částech spojitý segment musí být spojitý ve všech bodech t, kromě konečného počtu bodů $t' \in \langle t_1, t_2 \rangle$
3. Po částech konstantní segment je speciální případ po částech spojitého segmentu.
4. DEVS segment (event segment) $\omega : \langle t_0, t_n \rangle \rightarrow A \cup \{\emptyset\}$ je segment nad spojitou časovou základnou a libovolnou množinou $A \cup \{\emptyset\}$, kde $\emptyset$ označuje žádnou událost (non-event), která je hodnotou segmentu mezi výskyty jednotlivých událostí z množiny A v časech $t_i, i = 1, \ldots, n-1$.
5. Prázdný segment $\omega : \langle t_1, t_1 \rangle \rightarrow A$ označujeme symbolem $\emptyset$

Příklady segmentů



Dynamický systém

Pod pojmem *systém* rozumíme množinu prvků, které spolu navzájem spolupracují (komunikují) tak, aby vyhovovaly našim požadavkům a vykonávaly specifikovanou činnost.
Systém můžeme chápat buď jako samostatnou jednotku, která není ovlivňována z *okolí systému* — tzv. *uzavřený systém*, nebo systém má vstupy a výstupy a jde o *otevřený systém*.

Poznámka: Hierarchie specifikací systému (*"blackbox"*, modulární systémy)

Obecný dynamický systém

Obecný dynamický systém *DS* lze definovat jako strukturu:

$$DS = (T, X, Y, \Omega, Q, \Delta, \Lambda)$$

kde

- T je časová základna,
- X je množina vstupních hodnot,
- Y je množina výstupních hodnot,
- Ω je množina přípustných vstupních segmentů $\omega : \langle t_1, t_2\rangle \rightarrow X$ nad T a platí, že Ω je uzavřená vzhledem ke konkatenaci a levé segmentaci.
- Q je množina stavů systému,
- $\Lambda : Q \times X \rightarrow Y$ je výstupní funkce

- $\Delta : Q \times \Omega \rightarrow Q$ je globální přechodová funkce, která musí splňovat následující vlastnosti:
  - Konzistence: $\Delta(q, \omega_{(t,t\rangle}) = q$.
  - Vlastnost pologrupy: $\forall \omega : \langle t_1, t_2 \rangle \rightarrow X \in \Omega$,
    $t \in \langle t_1, t_2 \rangle : \Delta(q, \omega_{\langle t_1, t_2 \rangle}) = \Delta(\Delta(q, \omega_{\langle t_1, t \rangle}), \omega_{\langle t, t_2 \rangle})$.
    Tato vlastnost zaručuje že množina stavů Q zachycuje veškerou historii systému tak, že systém může být přerušen v libovolném čase a pokračování z tohoto času povede ke stejnému výsledku jako bez přerušení.
  - Kauzalita: $\forall \omega, \bar{\omega} \in \Omega$ jestliže platí $\forall t \in \langle t_1, t_2 \rangle : \omega(t) = \bar{\omega}(t)$ potom $\Delta(q, \omega_{\langle t_1, t_2 \rangle}) = \Delta(q, \bar{\omega}_{\langle t_1, t_2 \rangle})$

Množina vstupů X a výstupů Y spolu s množinou přípustných vstupních segmentů Ω definuje *vstupní a výstupní rozhraní* systému.

Okolí systému

- Každý otevřený systém podle předcházející definice je obklopen dalšími systémy, které definují jeho okolí.
- Toto okolí poskytuje systému vstupní hodnoty a sleduje výstupy systému.
- Prakticky většinou uvažujeme pouze okolní systémy pro model významné a všechny ostatní zanedbáváme.

Podsystémy

- Podsystém je systém, který definuje určitou část rozsáhlejšího systému.
- Strukturu složeného systému můžeme popsat grafem (uzly jsou elementární podsystémy, hrany představují vzájemné interakce).
- Hierarchické podsystémy:
  - Redukují celkovou složitost systému.
  - Musíme definovat elementární úroveň popisu.

Model dynamického systému

- *Model* je speciální případ systému, který má přesně definovaný (homomorfní) vztah k originálnímu vzorovému systému.
- Existuje zobrazení (modelování), které mapuje systém na model a přitom zachová podstatné vlastnosti systému. Toto zobrazení je obecně nejednoznačné, to znamená, že jeden systém můžeme modelovat více modely podle účelu a úrovně abstrakce a jeden model může odpovídat více vzorům (to je možné, protože model abstrahuje některé detaily systému).
- Homomorfní zobrazení mezi systémem a modelem dovoluje zjednodušení struktury modelu.

Strukturované množiny

V definici obecného dynamického systému nejsou kladena žádná omezení na množiny vstupů, výstupů a stavů — nemusí to být jen čísla.
Pro simulační modelování budeme používat *strukturované množiny*, definované takto:

$$S = (V, S_1 \times S_2 \times \ldots \times S_n)$$

kde V je uspořádaná množina n proměnných a kartézský součin množin $S_1, \ldots, S_n$ definuje n-tice, ve kterých každý prvek odpovídá hodnotě proměnné z množiny V.

Strukturované množiny

Pro zjednodušení zápisu budeme strukturované množiny zapisovat:

$$S = \{(v_1, v_2, \dots, v_n) | v_1 \in S_1, \dots, v_n \in S_n\}$$

Prvky $v_1, v_2, \dots, v_n$ jsou *proměnné* nebo *porty* v závislosti na použité množině:

- pro vstupní a výstupní množiny použijeme termín port,
- u množiny stavů použijeme termín stavová proměnná.

Používáme operace $variables(s) = (v_1, v_2, \dots, v_n)$ pro získání množiny proměnných a operaci $range_{v_i}(S) = S_i$, pro získání rozsahu proměnné v_i.

Strukturované množiny — projekce

Dále definujeme operaci *projekce*, která zpřístupní zadanou souřadnici ve strukturované množině

$$S = \{(v_1, v_2, \ldots, v_n) | v_1 \in S_1, \ldots, v_n \in S_n\}$$

s prvkem $s = (s_1, \ldots, s_n)$:

$$. \quad : \quad S \times V \rightarrow \bigcup_{i=1}^{n} S_i$$

$$s.v_i := s_i$$

Poznámka: Funkce $f : A \rightarrow B$, jejíž doména A a rozsah hodnot B jsou strukturované množiny, se nazývá strukturovaná funkce. Stejně jako pro strukturované množiny definujeme projekci i pro strukturované funkce.

DEVS formalismus

DEVS (*Discrete EVent specified System*) [Zeigler]
Byl rozšířen o možnost popisovat další kategorie systémů:

- DEVS – popisuje jen diskrétní systémy
- DTSS (*Discrete Time Specified System*) – popis systémů s diskrétním časem
- DESS (*Differential Equation Specified System*) – popisuje spojité systémy
- DEVS&DESS – kombinované systémy
- Modulární DEVS – skládání z komponent
- Parallel DEVS – varianta vhodná pro paralelní implementaci
- různá speciální rozšíření: Fuzzy-DEVS, Real-Time DEVS, Dynamic Structure DEVS, Cell-DEVS

DEVS formalismus

- DEVS formalismus rozlišuje *atomické* a *složené* modely. Atomický DEVS je modulární jednotka s rozhraním tvořeným vstupními a výstupními *porty*.
- Chování je popsáno událostmi, které nastávají v čase. DEVS reaguje na externí vstupní události externí přechodovou funkcí a na interní (časově plánované) události reaguje interní přechodovou funkcí.
- Plánování interních událostí je prováděno funkcí posuvu času (*Time advance function*), která definuje čas zbývající do další interní události.

definice DEVS formalismu

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta)$$

kde

- $X = \{(x_1, x_2, \ldots, x_m) | x_1 \in X_1, x_2 \in X_2, \ldots, x_m \in X_m\}$ je množina vstupů
- $Y = \{(y_1, y_2, \ldots, y_p) | y_1 \in Y_1, y_2 \in Y_2, \ldots, y_p \in Y_p\}$ je množina výstupů
- $S = \{(s_1, s_2, \ldots, s_n) | s_1 \in S_1, s_2 \in S_2, \ldots, s_n \in S_n\}$ je množina stavů,
- $\delta_{ext} : Q \times X \to S$ je externí přechodová funkce kde $Q = \{(s, e) : s \in S, 0 \leq e \leq ta(s)\}$ je množina totálních stavů,
- $\delta_{int} : S \to S$ je interní přechodová funkce,

Definice DEVS formalismu – pokračování

- $\lambda : S \rightarrow Y$ je výstupní funkce
- $ta : S \rightarrow \mathcal{R}_0^+ \cup \{\infty\}$ je funkce posunu času udávající čas zbývající do výskytu následující plánované události.

Takto definovaný systém implicitně omezuje prvky odpovídajícího dynamického systému:

Definice DEVS formalismu – pokračování

- Časová základna T musí být množina reálných čísel $\mathcal{R}$;
- X, Y, S mohou být libovolné množiny.
- $X^{\emptyset} = X \cup \{\emptyset\}$ (vstupní množina dynamického systému) je vstupní množinou DEVS sjednocenou se symbolem $\emptyset \notin X$, který reprezentuje žádnou událost (non-event).
- $Y^{\emptyset} = Y \cup \{\emptyset\}$ je výstupní množina dynamického systému.
- Množina stavů Q je stavovou množinou dynamického systému.
- Množina Ω přípustných vstupních segmentů je množina všech DEVS segmentů nad X a T.
- Stavové trajektorie jsou po částech konstantní segmenty nad S a T.
- Výstupní trajektorie jsou DEVS segmenty nad Y a T.

Hierarchický model — složený DEVS

Složený DEVS (DEVN) je definován takto:

$$N = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$$

kde

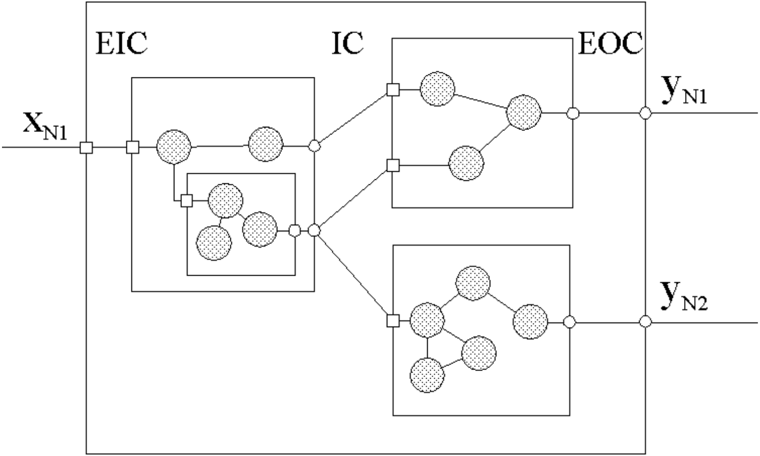
- X je množina vstupních událostí,
- Y je množina výstupních událostí,
- D je množina odkazů na DEVS komponenty,
- pro každé $d \in D$ platí, že M_d je odpovídající DEVS model
- pro každé $d \in D \cup \{N\}$ je I_d množina *"influencer"* = množina komponent, jejichž výstup ovlivňuje d: $I_d \subseteq D \cup \{N\}$, $d \notin I_d$

DEVN

- pro každé $i \in I_d$ je $Z_{i,d}$ funkce popisující propojení komponent (transformaci událostí):
  $Z_{i,d} : X \rightarrow X_d$ pro $i = N$
  $Z_{i,d} : Y_i \rightarrow Y$ pro $d = N$
  $Z_{i,d} : Y_i \rightarrow X_d$ pro $d \neq N$ a $i \neq N$
- $Select : 2^D \rightarrow D$ je funkce (tie-breaking function) pro rozhodování v případě výskytu více událostí současně.

Poznámka: Množina DEVS systémů je uzavřená vzhledem k operaci skládání, tj. každý složený systém je současně DEVS. Tato vlastnost je podstatná pro vytváření hierarchických systémů a byla formálně dokázána. Podobným způsobem lze definovat ostatní strukturované systémy s jiným chováním a jejich kombinace.

Hierarchický model



Systém popsaný diferenciálními rovnicemi

Atomický systém popsatelný diferenciálními rovnicemi (Differential Equation Specified System, DESS) je struktura:

$$DESS = (X, Y, S, f, \lambda)$$

kde

- X je množina vstupů,
- Y je množina výstupů,
- S je množina stavů,
- $f : S \times X \rightarrow S$ je funkce udávající rychlost změny stavu,
- $\lambda : S \rightarrow Y$ nebo $\lambda : S \times X \rightarrow Y$ je výstupní funkce Moorova resp. Mealyho typu.

DESS

Tento systém vyžaduje následující omezení vlastností množin obecného dynamického systému:

- časová základna T musí být množina reálných čísel $\mathcal{R}$,
- X, Y a S musí být vektorové prostory nad $\mathcal{R}$,
- množina Ω přípustných vstupních segmentů je množina všech po částech spojitých vstupních segmentů,
- stavové a výstupní trajektorie jsou po částech spojité

DESS

DESS má dynamické chování definované takto:

- pro daný vstupní segment $\omega : \langle t_1, t_2\rangle \to X$ a počáteční stav q v čase t_1 požadujeme, aby stavová trajektorie (*STRAJ*) dynamického systému v libovolném čase $t \in \langle t_1, t_2\rangle$ splňovala podmínku
$$STRAJ_{q,\omega}(t_1) = q$$
$$\frac{d\ STRAJ_{q,\omega}(t)}{dt} = f(STRAJ_{q,\omega}(t), \omega(t))$$
- Výstupní funkce Λ dynamického systému je $\Lambda(q, x) = \lambda(q)$ pro Moorovu variantu a $\Lambda(q, x) = \lambda(q, x)$ pro Mealyho variantu systému.

Aby řešení dynamického systému bylo jednoznačné, musí být splněna Lipschitzova podmínka.

Kombinace DEV&DESS

Kombinovaný model DEV&DESS zahrnuje spojité i diskrétní změny stavu systému (originální definice je mírně upravena):

$$DEV\&DESS = (X, X_c, Y, Y_c, S, \delta_{ext}, C_{int}, \delta_{int}, \lambda, ta, f, \lambda_c)$$

kde

- X, Y, S odpovídají obdobným množinám v tradičním DEVS,
- $X_c = \{(x_{c_1}, x_{c_2}, \ldots) | x_{c_1} \in X_{c_1}, x_{c_2} \in X_{c_2}, \ldots\}$ je strukturovaná množina hodnotových vstupů se vstupními proměnnými x_{c_i},
- $Y_c = \{(y_{c_1}, y_{c_2}, \ldots) | y_{c_1} \in Y_{c_1}, y_{c_2} \in Y_{c_2}, \ldots\}$ je strukturovaná množina hodnotových výstupů s výstupními proměnnými y_{c_i},

DEV&DESS

- $\delta_{ext} : Q \times X_c \times X \to S$,
- $\delta_{int} : S \times X_c \to S$,
- $\lambda : S \times X_c \to Y$,
- $ta : S \times X_c \to \mathcal{R}_0^+ \cup \{\infty\}$,
- $\lambda_c : S \times X_c \to Y_c$ je výstupní funkce pro definování hodnot výstupních proměnných,

DEV&DESS

- $C_{int} : S \times X_c \rightarrow \mathcal{B}$ je interní podmínková funkce podmiňující provádění interních (stavových) událostí,
- $S = S_d \times S_c$ je kartézský součin diskrétních a spojitých stavů $S_c = \{(s_{c_1}, s_{c_2}, \ldots) | s_{c_1} \in S_{c_1}, \ldots\}$,
- $f : S \times X_c \rightarrow S_c$ je funkce popisující časové změny spojitého stavu (derivative function).

Poznámka: Interní podmínková funkce a interní přechodová funkce jsou použitelné pro modelování *stavových podmínek a stavových událostí*.

Systém popsaný DEV&DESS

Tento formalismus omezuje obecný dynamický systém následujícím způsobem:

- Časová základna T musí být množina reálných čísel $\mathcal{R}$.
- $X \cup X_c$ je množina vstupů systému.
- $Y \cup Y_c$ je množina výstupů systému.
- X_c, Y_c a S_c musí být vektorové prostory nad $\mathcal{R}$.
- Množina Ω přípustných vstupních segmentů je množina všech po částech spojitých vstupních segmentů sjednocená s množinou všech DEVS segmentů nad X a T.
- Stavové trajektorie jsou po částech spojité a po částech konstantní segmenty.
- Výstupní trajektorie jsou po částech spojité (Y_c) a po částech konstantní segmenty (Y).

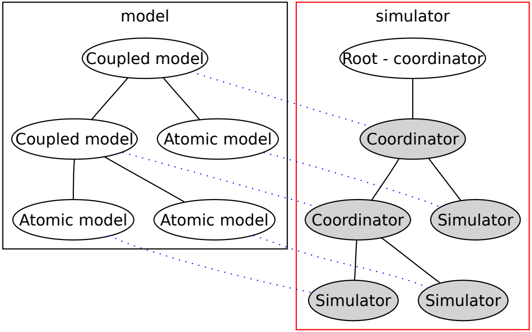
Systém popsaný DEV&DESS

Formalismus musí definovat pořadí událostí v případě jejich současného výskytu. Dynamické chování systému je popsáno v literatuře.

Poznámka: Systém popsatelný diferenčními rovnicemi (DTSS) je ekvivalentní DEV&DESS a nebude zde uvažován jako zvláštní případ.

DEVS simulátor

Simulace je transformace, která transformuje specifikaci systému, počáteční stav a vstupní segmenty na odpovídající stavové a výstupní segmenty. Simulaci modelu provádí simulátor:
- simulátor pro atomický DEVS model
- koordinátor pro složený (coupled) DEVS model



Simulátor

Simulátory ke komunikaci používají 4 typy zpráv:

- *Inicializační zprávy* (i, t)
  posílá nadřazený simulátor před začátkem simulace.
- *Interní zprávy (internal state-transition message)* $(*, t)$
  posílá koordinátor podřízeným – informují o plánovaných interních událostech.
- *Vstupní zprávy* (x, t)
  posílá koordinátor podřízeným – informují o externích událostech.
- *Výstupní zprávy* (y, t)
  posílají podřízení nadřízeným koordinátorům – informují o výstupních událostech.

Simulátor

DEVS simulátor interpretuje dynamiku DEVS modelu.

- Proměnné:
  tl – čas poslední události,
  tn – čas příští události (next event).
  Platí: $tn = tl + ta(s)$
- Pro daný simulační čas t můžeme vypočítat:
  čas od poslední události: $e = t - tl$
  čas zbývající do příští události: $\sigma = tn - t = ta(s) - e$

Čas tn se posílá nadřazenému koordinátoru, který jej používá pro synchronizaci.

Simulátor

Proměnné simulátoru:

parent – nadřazený koordinátor,
tl – čas poslední události,
tn – čas příští události,
DEVS – model se stavem (s, e),
y – aktuální výstup modelu

Simulátor – inicializace

Když DEVS-simulátor přijme inicializační zprávu v čase t, provede:

```
message(i,t) {
  tl = t
  tn = tl + ta(s)
}
```

Simulátor – plánovaná událost

Když DEVS-simulátor přijme interní zprávu v čase t, provede:

```
message(*,t) {
    if(t!=tn)
        ERROR: chybná synchronizace

    y = \lambda(s)                  // výstupní funkce
    send_message((y,t),parent) // výstupní zpráva
    s = \delta_{int}(s)     // interní přechodová fce

    tl = t
    tn = tl + ta(s)
}
```

Simulátor – vstupní událost

Když DEVS-simulátor přijme vstupní zprávu s hodnotou x v čase t, provede:

```
message(x,t) {
    if( t mimo rozsah tl..tn )
        ERROR: chybná synchronizace

    e = t - tl
    s = \delta_{ext}(s,e,x) // externí přechodová fce

    tl = t
    tn = tl + ta(s)
}
```

DEVS koordinátor

Koordinátor obsahuje kalendářní seznam a zajišťuje synchronizaci komponent složeného DEVS.

- První položka v kalendáři je plánována na čas:

$$tn = min\{tn_d | d \in D\}$$

  a tento čas je poslán nadřízenému koordinátoru jako čas příští události.
- Čas poslední události v podřízených simulátorech:

$$tl = max\{tl_d | d \in D\}$$

Koordinátor musí zpracovávat události podobně jako simulátor.

Koordinátor – proměnné

Proměnné koordinátoru

DEVN — model složeného DEVS:
$DEVN = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, Select)$

parent — nadřazený koordinátor

tl — čas poslední události,

tn — čas příští plánované události,

`event_list` — kalendář událostí,

$d*$ — aktuálně vybraný prvek

Koordinátor – inicializace

Když DEVS-koordinátor přijme inicializační zprávu v čase t, provede:

```
message(i,t) {
    for-each d in D {
        send message(i,t) to d
    }
    sort event_list   // podle tn_d a Select
    tl = max(tl_d)    // pro všechna d \in D
    tn = min(tn_d)
}
```

Koordinátor – plánovaná událost

Když DEVS-koordinátor přijme interní zprávu v čase t, provede:

```
message(*,t) {
    if(t!=tn)
        ERROR: chybná synchronizace

    d* = first(event_list)    // výběr z kalendáře
    send message(*,t) to d*   // přeposlání komponentě

    sort event_list           // podle tn_d a Select

    tl = t
    tn = min(tn_d)            // pro všechny komponenty
}
```

Koordinátor – vstupní událost

Když DEVS-koordinátor přijme vstupní zprávu x v čase t, provede:

```
message(x,t) {
    if( t mimo rozsah tl..tn )
        ERROR: chybná synchronizace
    // zjistíme, kterých komponent se týká:
    receivers = {r|r ∈ D, N ∈ I_r, Z_{N,r}(x) ≠ Φ}

    for-each r in receivers
        send message(x_r,t) to r    //  x_r = Z_{N,r}(x)

    sort event_list // podle tn_d a Select
    tl = t
    tn = min(tn_d)  // pro všechny komponenty
}
```

Koordinátor – výstupní událost

Když DEVS-koordinátor přijme od komponenty $d*$ výstupní zprávu s výstupem Y_{d*} v čase t, provede:

```
message(y_d*, t) {
  // kontrola externího propojení komponent
  if (d* ∈ I_n && Z_{d*,N}(y_{d*}) ≠ Φ)
       send message(y_N,t) to parent // y_N = Z_{d*,N}(y_{d*})

  // kontrola a informování komponent o změně y_{d*}
  receivers = {r|r ∈ D, d* ∈ I_r, Z_{d*,r}(y_{d*}) ≠ Φ}

  for-each r in receivers
     send message(x_r, t) to r
        // vstup x_r = Z_{d*,r}(y_{d*})

}
```

Hlavní koordinátor

Implementuje hlavní smyčku simulačního algoritmu.

Proměnné

t – simulační čas

child – podřízený simulátor nebo koordinátor

```
Simulace() {
    t = t_0                          // počáteční čas
    send message(i,t) to child   // inicializace
    t = tn of the child
    do {
        send message(*,t) to child
        t = tn of the child
    while( not end of simulation )
}
```

Simulátor DESS

Pro vícekrokovou metodu řádu m musí simulátor uchovat m hodnot stavu q, derivací r a vstupu x.

$$q(t+h) = METODA(h, q(t), q(t-h), ..., r(t), r(t-h), ...)$$

Poznámka: Problém startu metody.

Simulátor DESS – inicializace

Proměnné simulátoru:

$$\begin{aligned} DESS &= (X, Y, Q, f, \lambda) && \text{– model} \\ qvec &= q(t), q(t-h), \ldots && \text{– vektor uschovaných stavů} \\ rvec &= r(t), r(t-h), \ldots && \text{– vektor uschovaných derivací} \\ xvec &= x(t), x(t-h), \ldots && \text{– vektor uschovaných vstupů} \\ \mathrm{h} & && \text{– velikost kroku} \end{aligned}$$

Když DESS-simulátor přijme inicializační zprávu v čase t, provede:

```
message(i,t) {
    inicializace vektorů uschovaných hodnot
}
```

Simulátor DESS – plánovaná událost

Když DESS-simulátor přijme interní zprávu v čase t, provede:

```
message(*,t) {
    y = \lambda(q)                   // výstupní funkce
    send_message((y,t),parent) // výstupní zpráva
}
```

Simulátor DESS – vstupní událost

Když DESS-simulátor přijme vstupní zprávu s hodnotou x v čase t, provede:

```
message(x,t) {
    aktualizace xvec a rvec
    q(t+h) = METODA(h, qvec, rvec)
    aktualizace qvec
}
```

Spojitá simulace

Přehled:

- Principy
- Aplikace spojité simulace
- Nástroje pro spojitou simulaci
  - Knihovny podprogramů: ode, GSL, ...
  - Specializovaná prostředí: Matlab, Octave, Scilab, ...
  - Simulační jazyky: Modelica, ACSL, ...
- Numerické metody
- Algebraické (rychlé) smyčky
- Tuhé (stiff) systémy

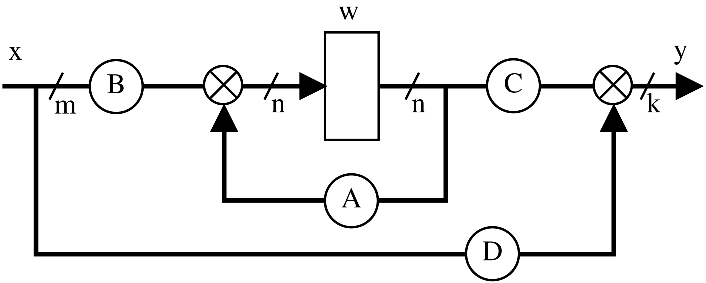
Formy popisu spojitých systémů

- Soustavy obyč. diferenciálních rovnic (ODE)
- Soustavy algebraických rovnic
- Bloková schemata
- Parciální diferenciální rovnice (PDE)
- + Kombinace více způsobů popisu
- ...
- Grafy signálových toků, Kompartmentové systémy, Systémová dynamika, Bond-graphs

Soustavy rovnic: maticový popis

$$\begin{aligned}\frac{d}{dt}\vec{w}(t) &= \mathbf{A}(t)\vec{w}(t) + \mathbf{B}(t)\vec{x}(t)\\ \vec{y}(t) &= \mathbf{C}(t)\vec{w}(t) + \mathbf{D}(t)\vec{x}(t)\end{aligned}$$

kde $\vec{x}$ je vektor m vstupů, $\vec{w}$ vektor n stavových proměnných, $\vec{y}$ vektor k výstupů a $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ matice koeficientů



Typy obyč. diferenciálních rovnic

Koeficienty (prvky matic $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$) mohou být:

- nezávislé na čase (stacionární systémy)
- časově proměnné
- konstanty (lineární systémy)
- nelineární funkce (nelineární systémy)

Poznámky:

- Vztah k $DESS = (X, Y, S, f, \lambda)$
- Algebraicko-diferenciální rovnice (DAE)
- Problémy při řešení: nespojitosti, ...
- Numerické metody a jejich vlastnosti

Knihovny podprogramů

Základ pro všechny běžné simulační nástroje.

Řada komerčních i volně dostupných zdrojů:

- Archív netlib (většinou "public domain")
  - `http://www.netlib.org/ode/`
  - `http://www.netlib.org/odepack/`
  - ...
- GSL: GNU Scientific Library (pro C, licence GPL)
- NumPy, SciPy (pro Python)
- ...
- Komerční zdroje – viz přehledy na WWW

Matlab, Scilab, Octave

Integrovaná prostředí:

numerické knihovny + vestavěný jazyk + vizualizace + ...

- programovatelné, vestavěný interpret jazyka
- operace s maticemi, vektory, atd.
- vstup/výstup, vizualizace 2D/3D
- vestavěné knihovny obsahují algoritmy pro řešení:
  - obyčejných diferenciálních rovnic (ODE)
  - diferenciálně–algebraických rovnic (DAE)
  - algebraických rovnic (rychlé smyčky)
- rozšiřitelné: toolboxy, propojení s jinými nástroji
- interaktivní: GUI, ladicí nástroje, editory: Matlab/Simulink, Scilab/SciCos

Simulační jazyky

Specializované jazyky a překladače:

- Modelica `http://www.modelica.org/`
  - objektově-orientovaný jazyk (mechanické, elektrické, hydraulické, tepelné modely, ...)
  - popis rovnicemi, automaticky upravuje rovnice
  - součást systému Dymola
  - OpenModelica
  - +rozsáhlé knihovny modelů
- ACSL – Advanced Continuous Simulation Language
  - možnost blokového/grafického popisu modelů
- existuje celá řada dalších jazyků/systémů

Numerické metody

Numerické řešení obyčejné diferenciální rovnice:

$$\frac{dy}{dt} = f(t, y), \qquad y(0) = y_0$$

v diskrétních časových okamžicích:

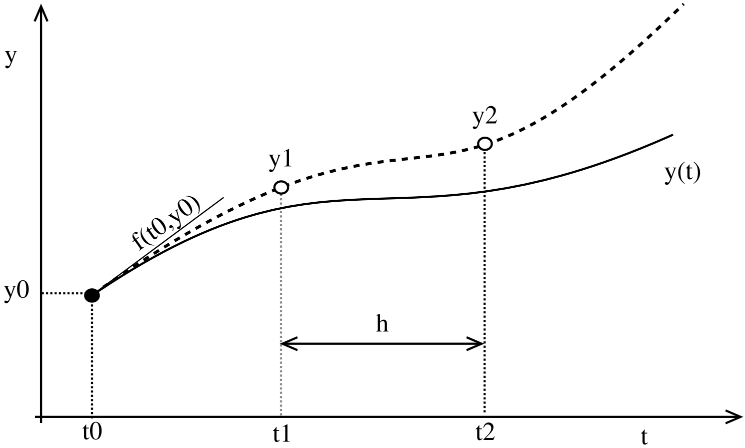
$$t_j = t_0 + jh$$

kde h je integrační krok.

Poznámky:

- Krok nemusí být konstantní
- Pozor na vlastnosti numerických metod

Numerické řešení ODE



Eulerova metoda

Taylorův rozvoj:

$$y(t) = y(t_0) + (t - t_0) \left.\frac{dy}{dt}\right|_{t_0} + \frac{(t - t_0)^2}{2} \left.\frac{d^2y}{dt^2}\right|_{t_0} + \dots$$

Ponecháme pouze první derivaci vyjádřenou funkcí $f(t_0, y_0)$ a dostaneme aproximaci:

$$y(t) \approx y(t_0) + (t - t_0) f(t_0, y_0)$$

Výpočet následující hodnoty j-tého kroku:

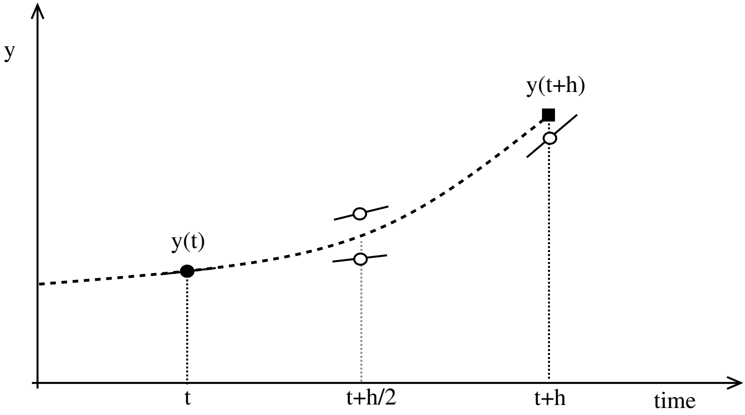
$$y_{j+1} = y_j + hf(t_j, y_j)$$

Eulerova metoda – příklad v C – (opakování)

```
double  dy[2],     y[2] = { 1.0, 0.0 };
double  time = 0,  h = 0.001;
void f() {  // výpočet derivací (vstupů integrátorů)
    dy[0] = y[1];       // y'
    dy[1] = -y[0];      // y''
}
void integrate_euler() { // krok num. integrace
    f();                            // výpočet derivací
    for (int i = 0; i < 2; i++) // pro všechny integrátory
        y[i] += h * dy[i];      //   Euler
    time += h;                  // posun času
}
int main() {
    while (time < 20) {
        printf("%10f %10f\n", time, y[0]);
        integrate_euler();
    }
}
```

Metody Runge-Kutta

Používají vážený průměr derivací v několika bodech uvnitř kroku:



Metoda Runge-Kutta 4. řádu

RK4

$$
\begin{aligned}
k_1 &= f(t_j, y_j) && (1)\\
k_2 &= f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_1) && (2)\\
k_3 &= f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_2) && (3)\\
k_4 &= f(t_j + h, y_j + hk_3) && (4)\\
y_{j+1} &= y_j + h(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}) && (5)
\end{aligned}
$$

Odhad lokální chyby je řádově $O(h^4)$

Poznámka: Existuje řada jiných variant

Numerické metody – přehled

Klasifikace:

Jednokrokové: Euler, Runge-Kutta, ...

Vícekrokové: Adams-Bashforth, ...

Varianty:

- Explicitní: $y(t+h) := g(h, t, y(t), ...)$
- Prediktor-korektor (PECE = Predict, Evaluate, Correct, Evaluate)
- Implicitní: $y(t+h) = g(h, t, y(t+h), y(t), ...)$

Poznámky:

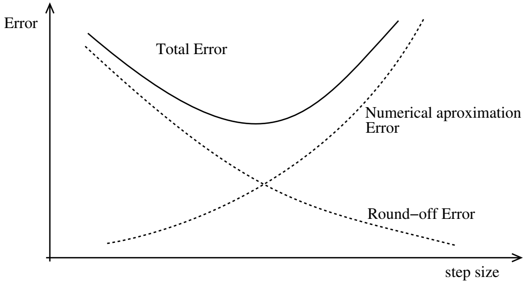
- Volba kroku (automatická: RK45, DOPRI45, ...)
- Metody vyšších řádů jsou efektivnější

Přesnost numerických metod

Lokální chyba v libovolném *j*-tém kroku:

$$e_j = y_j - y(t_j)$$

kde $y(t_j)$ je přesné řešení.



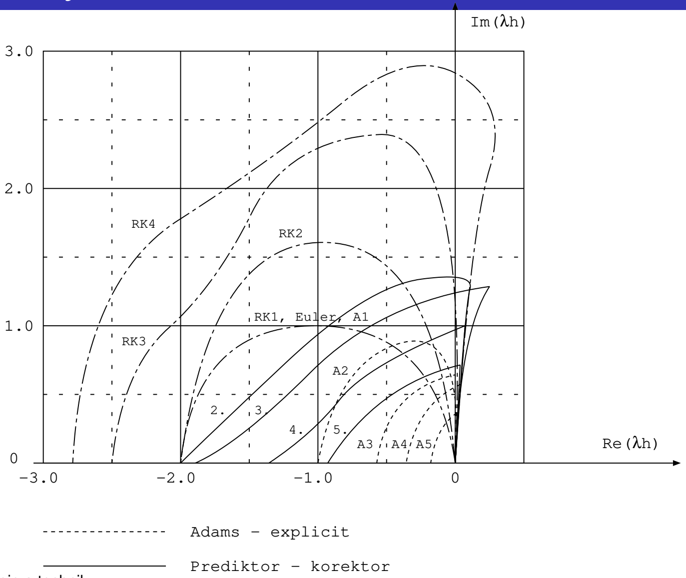
Globální diskretizační chyba: $GDE = max(e_j) \quad j = 1, \ldots$
zahrnuje akumulované lokální chyby.

Stabilita numerických metod

- Výpočet může být pro příliš velkou hodnotu kroku nestabilní (bez omezení se vzdaluje od přesného řešení).
- Každá metoda má určitou charakteristickou oblast stability (jde o plochu v komplexní rovině λh ve které leží vlastní čísla λ matice A soustavy řešených dif. rovnic).
- Stabilita je velmi důležitou vlastností numerické metody.

Poznámka: Analytická stabilita vs. stabilita num. metody

Oblast stability metod



Používané numerické integrační metody

Nejčastěji se používají varianty metody Runge-Kutta a pro speciální případy jiné metody.

- Matlab:

  ode45 – Runge-Kutta 4. a 5. řád,
  ode23 – Runge-Kutta 2. a 3. řád,
  ode113 – Adams-Bashforth-Moulton prediktor-korektor, 1.-13. řádu

- Octave:
  ode45, lsode (více metod: Adams,...), odessa, ...

- Knihovny:
  GSL: `gsl_odeiv_step_rkf45` ...

Tuhé (stiff) systémy rovnic

Tuhost soustavy diferenciálních rovnic způsobuje numerickou nestabilitu některých integračních metod.

- Přesná definice pojmu tuhost (stiffness) je obtížná. Příklady najdete např na `http://en.wikipedia.org/wiki/Stiff_equation`
- Metody, které jsou tzv. "A-stabilní", jsou použitelné pro řešení tuhých rovnic.
- Oblast absolutní stability u A-stabilních metod obsahuje celou komplexní polorovinu definovanou rovnicí $Re(z) < 0$

Poznámka: Tuhý systém vyžaduje při řešení rovnic běžnými metodami použití příliš krátkého integračního kroku.

Používané metody pro tuhé systémy

- Matlab:
  ode15s – vícekroková 1.-5. řád,
  (ode23s – RK pro středně tuhé systémy)
- Octave:
  lsode + extra parametry, ...
- Knihovny:
  GSL: Bulirsch-Stoer, ...

Poznámka: Existují varianty pro různě tuhé systémy

DAE (Differential-Algebraic Equations)

Při výpočtu derivací je v některých modelech třeba řešit soustavy algebraických rovnic. (Typické například při popisu elektrických obvodů.) Obecná forma zápisu:

$$u' = f(t, u, v)$$
$$0 = g(t, u, v)$$

kde u jsou stavové a v jsou algebraické proměnné
Nejjednodušší způsob řešení je pro každé vyhodnocení f řešit algebraické rovnice g. (Lepší metody viz např. Matlab.)
Existují speciální podprogramy pro řešení DAE:

- Matlab: ode23t
- Octave: dassl, daspk, dasrt
- Knihovny: DAEPACK, ... viz netlib

Simulace elektrických obvodů

Speciální případ spojitých simulací:
analýza chování elektrických součástek a obvodů (ustálený stav, dynamické chování, vlastnosti při různých frekvencích, ...)

- SPICE – specializovaný nástroj – existují různé implementace
- Univerzální simulační nástroje: Modelica, ...

Parciální diferenciální rovnice

Rovnice s derivacemi podle více proměnných – např.:

$$\frac{\partial^2 y}{\partial t^2} = a\frac{\partial^2 y}{\partial x^2}$$

Klasifikace: řád, (2: eliptické, parabolické,hyperbolické)

Značení

$$\frac{\partial u}{\partial x} = \partial_x u = u_x \qquad \frac{\partial^2 u}{\partial x^2} = u_{xx}$$

- Počáteční podmínky
- Okrajové podmínky

Metody pro řešení PDE

Typy numerických metod pro řešení PDE:

- Metoda konečných diferencí ("Finite difference", FDM)
- Metoda přímek (MOL, "Method of lines")
- Metoda konečných prvků ("Finite element", FEM)
- Metoda konečných objemů ("Finite volume", FVM)
- Další metody: Monte-Carlo, ...

Poznámka: Problém generování sítě ("mesh").

Software:

- Komerční: FLUENT, ANSYS, ...
- Volně dostupné: OpenFOAM, FreeFEM, ...

Metoda konečných diferencí (metoda sítí)

1. Pokrytí oblasti, v níž hledáme řešení, sítí konečného počtu uzlových bodů.
2. Derivace v uzlových bodech nahradíme příslušnými diferencemi, tj. lineárními kombinacemi funkčních hodnot v okolních bodech. (V závislosti na tom, zda volíme diference dopředné či zpětné, dostáváme různé typy metody sítí.)
3. Po záměně derivací diferencemi ve všech uzlových bodech dostáváme soustavu lineárních algebraických rovnic s neznámými hledanými hodnotami v těchto uzlových bodech.
4. Řešení soustavy.

Poznámka: Hledanými hodnotami mohou být například výchylky nosníků atd.

Metoda konečných prvků (FEM)

Patří mezi metody variační, vycházející z minimalizace energetického potenciálu.
Postup řešení:

1. Generování sítě (mesh) = diskretizace. Nejčastěji používaným typem konečných prvků v rovině jsou trojúhelníky.
2. Na každém konečném prvku se volí vhodná aproximační funkce přesného řešení, která jednoznačně definuje stav uvnitř tohoto prvku pomocí jeho uzlů.
   Na základě této aproximace se pak s využitím podmínek pro minimalizaci energetického potenciálu odvodí pro každý uzel rovnice rovnováhy, která je funkcí těchto neznámých v uzlových bodech sítě.

Metoda konečných prvků

3. Řešením této soustavy algebraických rovnic získáme hodnoty v uzlových bodech.
4. Tyto hodnoty pak společně s dalšími charakteristikami definují stav jak uvnitř prvku, tak i na jeho hranicích.

Metoda konečných prvků umožňuje řešit úlohy se složitými okrajovými podmínkami, se složitou geometrií, každý prvek může mít odlišné vlastnosti.

Metoda konečných objemů

- Používána v CFD (Computational Fluid Dynamics).
- Řeší problém přesunu tekutin v prostoru – modelem je Navier-Stokesova rovnice

Podrobnosti viz literatura, např:
`http://en.wikipedia.org/wiki/Navier-Stokes_equations`

Metoda přímek

- Postupujeme podobně jako u metody konečných diferencí, ale ponecháme derivaci podle jedné proměnné.
- Výsledkem je soustava obyčejných diferenciálních rovnic, které řešíme obvyklými metodami.
- Metody přímek rozdělujeme podle toho, která z nezávisle proměnných zůstává spojitá. Provedeme-li diskretizaci v prostorových souřadnicích jde o metodu typu DSCT (*Discrete Space Continuous Time*).
  Tato metoda je vhodná pro výpočet dynamického chování systému v čase.

Příklad: kmitání struny (viz WWW: SIMLIB)

Příklad: Kmitání struny, metoda přímek

Vlnová parciální diferenciální rovnice

$$\frac{\partial^2 y}{\partial t^2} = a\frac{\partial^2 y}{\partial x^2}$$

(sledujeme výchylku pouze ve směru osy y)

Počáteční podmínky:

$y(x,0) = -\frac{4}{L^2}x^2 + \frac{4}{L}x$
$y'(x,0) = 0$

Okrajové podmínky:

$y(0,t) = y(L,t) = 0$
(Struna délky L je na koncích upevněna.)

Příklad: Kmitání struny – pokračování

Čas ponecháme spojitý a prostorovou souřadnici x diskretizujeme:

- Hledáme řešení v $n + 1$ bodech intervalu $\langle 0, L \rangle$.
- Vzdálenost dvojice bodů ve směru osy x bude Δx (rovnoměrné rozdělení bodů).
- Parciální derivaci podle času nahradíme obyčejnou derivací podle času.
- Parciální derivaci podle x nahradíme vhodným diferenčním vztahem, např.

$$\left.\frac{\partial^2 y}{\partial x^2}\right|_{x_i} = \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2}$$

Tento vztah lze získat z Taylorova rozvoje funkce y v bodech $(x_i + \Delta x)$ a $(x_i - \Delta x)$.

Příklad: Kmitání struny – pokračování

- Po dosazení do rovnice dostaneme soustavu obyč. diferenciálních rovnic 2. řádu:

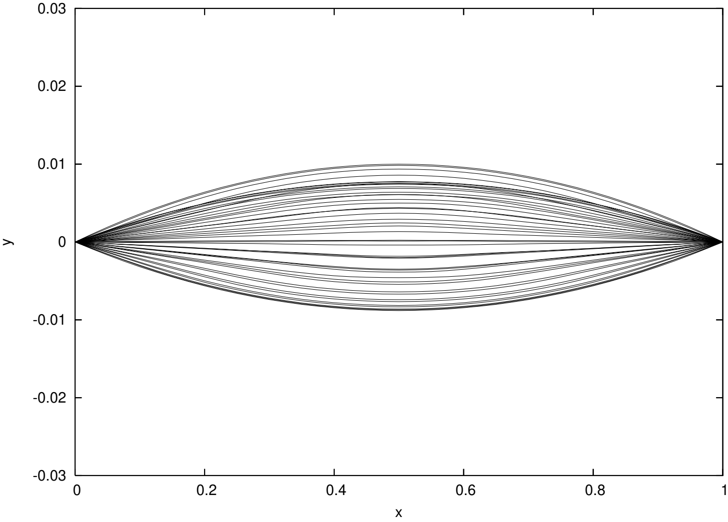
  $$\frac{d^2 y_i}{dt^2} = \frac{a}{\Delta x^2}(y_{i+1} - 2y_i + y_{i-1})$$

  pro $i = 1, ..., n-1$,
  Okrajové podmínky: $y_0 = 0, \quad y_n = 0$
  Počáteční podmínky: $y_i(0) = -\frac{4}{L^2}x_i^2 + \frac{4}{L}x_i$

  Tuto soustavu již umíme upravit metodou snižování řádu derivace a řešit.

- Výsledkem jsou časové průběhy výchylky $y_i = y(x_i)$ pro $i = 0, 1, ..., n$.

Příklad: Kmitání struny – výsledek



Poznámky

Diskrétní simulace

- Opakování — viz předmět IMS
- Principy
  - Next-event: kalendář
  - Události
  - Procesy
  - Aktivity ("Activity scanning")
- Použití
- Nástroje: diskrétní simulační jazyky
- ...

Diskrétní simulace – úvod

- Modelový čas: spojitý (ve speciálních případech i diskrétní)
- Stav modelu: jakýkoli, změna pouze v diskrétních časových okamžicích
- Sledování stavu: časy událostí, doba trvání obsluhy, statistiky, ...
- Základní formy popisu: události, procesy, aktivity

Poznámka: Opakování: Petriho sítě, SIMLIB/C++ (viz IMS)

Události a procesy

Konceptuální rozdíl v přístupu k modelování

- Události: popis změn stavu.
- Procesy: posloupnosti na sebe navazujících událostí.
  Typický přístup používá:
  - aktivní transakce (popisuje posloupnost souvisejících událostí),
  - pasivní zařízení (reagují jen na požadavky transakcí)

Poznámka: Agent-based simulation: zapouzdření procesů do objektů/agentů (obvykle s ”inteligentním” chováním popsaným pravidly).

Next-event přístup

Kalendář (*Pending Event Set*): datová struktura typu prioritní fronta

- Každá naplánovaná budoucí událost ”next event”) má v kalendáři záznam $[(acttime_i, priority_i, event_i), ...]$
- Kalendář definuje operace:

FindMin: nalezení záznamu s nejmenším aktivačním časem a prioritou (lze použít i na testování prázdného kalendáře)

DeleteMin: vyjmutí záznamu s nejmenším aktivačním časem a prioritou

Insert: vložení nového záznamu

Delete: rušení zadaného záznamu

Init: inicializace kalendáře

Destruct: zrušení kalendáře

Algoritmus řízení simulace

Diskrétní simulátor typu ”next-event” se řídí algoritmem:

```
Time = T_START; // modelový čas
calendar.Init();
model.Init();
while( act = calendar.FindMin() ) {
    if(act.atime > T_END)
        break;
    calendar.DeleteMin(); // vyjmout
    Time = act.atime;     // nastavit modelový čas
    act.event.Run();      // provést událost
}
Time = T_END;   // konec simulace
```

Poznámky: Stav modelu se mezi událostmi nemění.
Procesy jsou implementovány jako posloupnosti událostí.

Implementace kalendáře událostí

Nejdůležitější je časová složitost operací vložení do kalendáře a výběru nejmenšího prvku.

Používané implementace a složitost operací:

- Lineární seznam: $O(n)$, $O(1)$
- Stromy: heap, pairing heap, `priority_queue`, ... $O(\log n)$
- Calendar queue $O(1)$
- Různé další varianty seznamů (skiplist, ...)
- "Lazy queue" – omezení velikosti kalendáře

Poznámka: Viz příklady

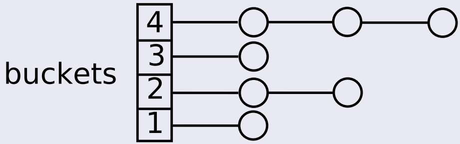
Implementace kalendáře: Seznam

- Seznam (double-linked list)

- Časová složitost operací:
  - Vložení: $O(N)$
  - Výběr prvního prvku: $O(1)$
- Vhodné pro malé počty plánovaných událostí (max stovky).

Implementace kalendáře: Calendar queue

- Pole seznamů (*buckets*)

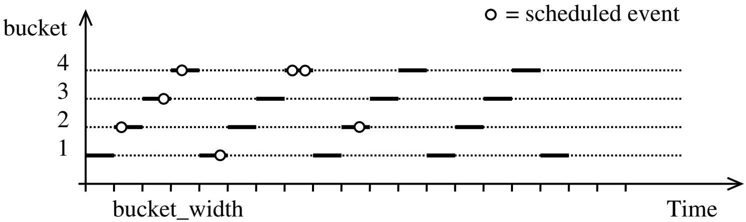


- Přizpůsobuje velikost pole ($nbuckets = 2^k$)
- Průměrná časová složitost operací:
  - Vložení: $O(1)$
  - Výběr prvního prvku: $O(1)$
- Vhodné pro velký počet naplánovaných událostí.

Literatura: R. Brown. "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem". Comm. ACM, 31(10):1220–1227, Oct. 1988.

Calendar queue – pokračování

- Index do pole je "den", v seznamech jsou uspořádány záznamy pro daný "den". Rozložení záznamů závisí na délce "dne" (parameter *bucket width*):

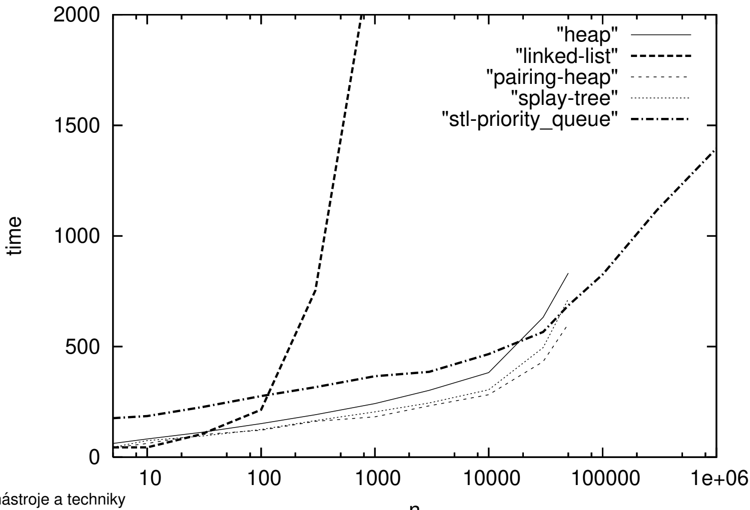


- Průměrná délka seznamů je cca 1.5, pokud se zvýší, pole se zvětší (rozdělíme "den" na "půlden") a naopak.
- Problém: stanovení vhodné délky "dne"

Implementace: viz literatura

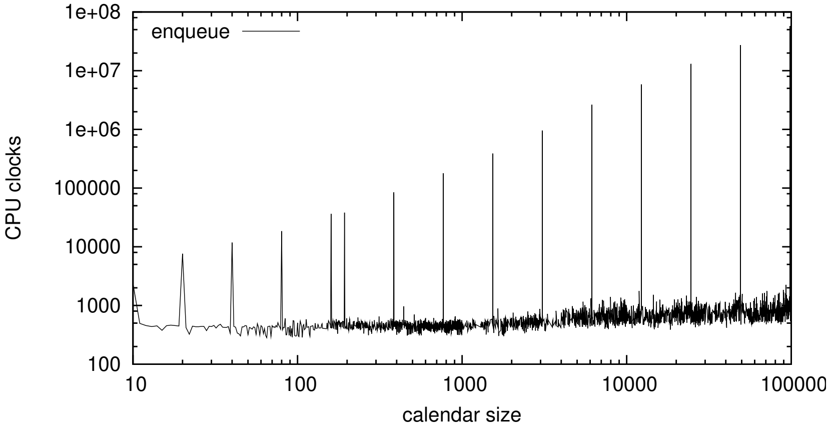
Příklad: efektivita implementace

Experimentálně získaná doba trvání operací pro různé typy kalendáře:



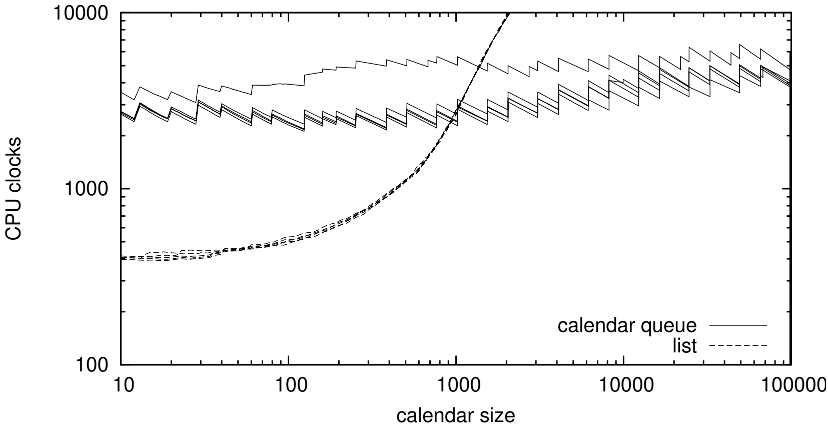
Příklad: efektivita implementace CQ

Doba trvání operace vkládání v závislosti na počtu položek:



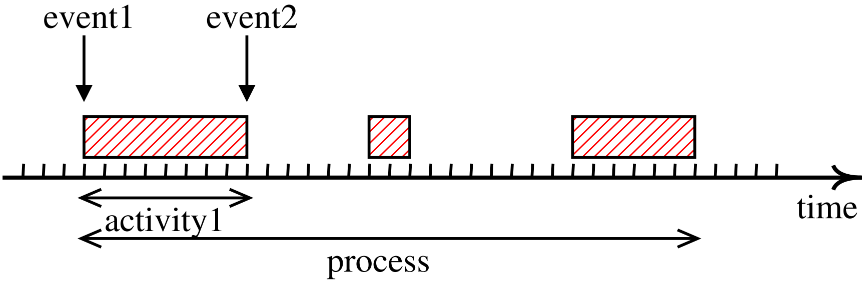
Příklad: efektivita

Průměrný čas pro obě operace (vkládání + výběr), různá rozložení:



Poznámky: neoptimalizováno, další varianta: SNOOPY

Události, procesy, aktivity



Algoritmus ”Activity scanning”

Chování modelu je popsáno množinou podmínek a jim odpovídajících operací. Simulátor posouvá čas po krocích.

```
inicializace
while(Time<TEND) {
    if(podmínka1)
        akce1  // = zahájení nebo ukončení aktivity
    if(podmínka2)
        akce2
    if(podmínka3)
        akce3
    // ...
    Time += deltaT;  // změna času
}
```

Použití "Activity scanning"

- Vhodné pro modely s velkým počtem událostí rovnoměrně rozložených v čase.
- Nevýhody proti next-event přístupu:
  - menší přesnost v případě pevného časového kroku
  - nutnost testovat všechny podmínky pro každý interval (i když se dlouho nic neděje)
- Použití: animace (musíme zobrazovat každý krok), hry
- Souvislosti:
  - podmínky typu WaitUntil
  - kombinovaná simulace (stavové podmínky/události)
  - ...

Nástroje pro diskrétní simulaci

- Klasifikace simulačních jazyků:
  - orientované na události (SIMSCRIPT,...)
  - orientované na procesy (Simula,...)
  - orientované na aktivity (CSL,...)
- Speciální případy optimalizované pro číslicové systémy, celulární automaty, ...
- Různé implementace: paralelní, distribuované (HLA)
- ...

Stručný přehled a příklady – viz WWW

Implementace diskrétních simulátorů

Pouze přehled (podrobnosti viz zdrojové texty nástrojů).

- Algoritmus řízení simulace a pomocné datové struktury (kalendář)
- Události, aktivity: implementace obyčejnými funkcemi/metodami.
- Procesy: problém "přerušitelnosti" prováděného kódu při čekání (SIMLIB/C++: příkazy Wait, WaitUntil, Seize, Enter, Passivate). Možnosti implementace:
  - preprocesor/interpret
  - "vlákna" (coroutines) implementovaná např. setjmp/longjmp

Kombinovaná simulace

Kombinovaná (hybridní) simulace spojuje spojitou i diskrétní simulaci.

Základní principy:

- Stavové podmínky a stavové události.
- Synchronizace spojité a diskrétní části (”dokročení”).

- Použití: často je nutné kombinovat více přístupů.
- Nástroje: většina simulačních systémů je schopna (na různé úrovni) kombinovat spojité a diskrétní přístupy (viz Modelica).

Stavové podmínky

Zápis:

```
if(podmínka)
   akce-stavová-událost;
```

nebo speciální bloky/objekty (viz např. SIMLIB/C++)

Implementace — řešení algebraických rovnic:

- Půlení intervalu (bisection)
- Newtonova metoda
- Regula-falsi
- ...

Poznámka: Opakování — viz IMS. Jazyk Modelica bude později.

Stavové události

- Implementace: obyčejné funkce/metody
- Mohou měnit stav spojité i diskrétní části modelu.
  - Skoková změna stavu spojité části obvykle vyžaduje speciální inicializaci numerické integrační metody (např. u vícekrokových metod).
  - Diskrétní změnou může být např. plánování události atd.

Nástroje pro kombinovanou simulaci

Pouze stručný přehled, příklady

Implementace: Lze použít speciální metody pro řešení algebraicko-diferenciálních rovnic a dif. rovnic se zpožděním:

- FORTRAN: DASSL, LSODAR, ...
- Scilab/Octave/Matlab
- Modelica/Dymola (`when`)
- SIMLIB/C++: speciální objekty (`Condition`), metoda půlení intervalu
- ...

Analýza citlivosti systému na parametry

Princip:

- Reakce systému na vstupy je závislá na parametrech, které měříme nebo odhadujeme
- I malá změna parametrů může velmi ovlivnit výsledné chování systému
- Souvislosti: chaotické chování, bifurkace

- Použití: důležité pro nastavení výrobních systémů (dovolené tolerance součástek, nastavení strojů atd.)
- Nástroje: vestavěné do sim. systémů, toolboxy, ...

Koeficient citlivosti

Označuje-li y_i nějakou výstupní proměnnou systému S a p_j parametr, jehož vliv vyšetřujeme, můžeme definovat *citlivost proměnné* y_i na *parametr* p_j *(koeficient citlivosti)* v určitém bodu x_0 vstupního prostoru a hodnotě parametru p_{j0} jako parciální derivaci

$$S_{y_i p_j} = \left. \frac{\partial y_i}{\partial p_j} \right|_{x_0, p_{j0}}$$

Příklad výpočtu koeficientu citlivosti

Mějme spojitý systém se vstupem x a výstupem y

$$y' + ay = bx$$

s počáteční podmínkou $y(0) = y_0$.

- Máme dva parametry a a b
- Hledáme citlivost výstupu y na parametr a pro jednotkový skok na vstupu x v okolí hodnoty parametru $a = a_0$.
- Předpokládejme, že se změní hodnota parametru a o Δa. Následkem toho se změní i výstup y. *Odchylku(perturbaci)* označíme Δy což je obecně funkce času.

Příklad – řešení

Můžeme zapsat tzv. *rovnici odchylek (perturbací)*

$$(y + \Delta y)' + (a_0 + \Delta a)(y + \Delta y) = b \qquad \Delta y(0) = \Delta y_0$$

a řešit ji pro neznámou odchylku Δy. Dostaneme opět diferenciální rovnici

$$(\Delta y)' + a_0 \Delta y + \Delta a y + \Delta a \Delta y + y' + a_0 y = b_0$$

odkud po dosazení za y' z původní rovnice a po linearizaci (zanedbání členu $\Delta a \Delta y$) obdržíme

$$(\Delta y)' + a_0 \Delta y = -\Delta a y \qquad \Delta y(0) = \Delta y_0$$

Jejím řešením je časový průběh odchylky výstupu y při změně parametru a o Δa.

Příklad – pokračování

Jinou možností je zjišťovat průběh koeficientu citlivosti $S_{ya}(t)$. Mezi odchylkou Δy a koeficientem citlivosti S_{ya} platí pro malé hodnoty Δa vztah: $\Delta y(t, a) = S_{ya}(t, a)\Delta a$

Derivováním výchozí rovnice pro $x = 1$ podle parametru a získáme rovnici tvaru

$$\frac{\partial}{\partial a} y' + \frac{\partial}{\partial a}(ay) = 0 \tag{6}$$

Protože parciální derivace výstupu y podle a značí koeficient citlivosti S_{ya} a parametr a není časově proměnný, můžeme rovnici upravit na tzv. *rovnici citlivosti*, která má pro $a = a_0$ tvar

$$S'_{ya} + a_0 S_{ya} = -y \tag{7}$$

s počáteční podmínkou $S_{ya}(0) = S_{ya0}$.

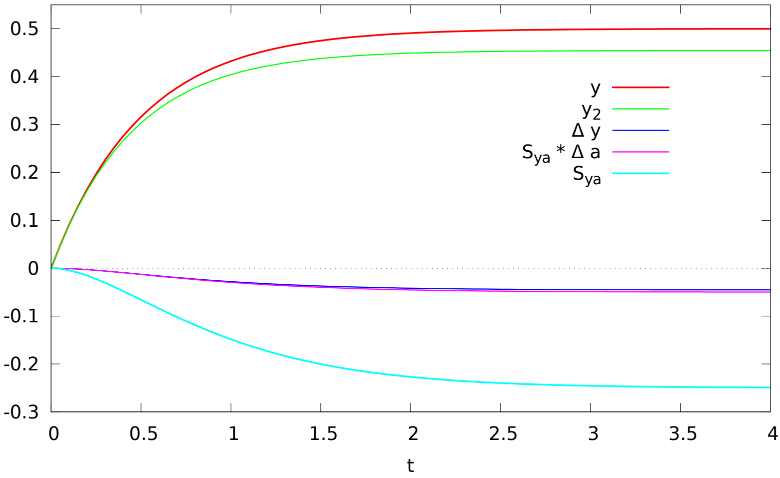
Příklad – pokračování

Z rovnice citlivosti je vidět, že výstupní veličina y je zde ve funkci vstupu. Proto lze vytvořit simulační model, který bude současně řešit průběh výstupu y i koeficientu citlivosti S_{ya}, resp. odchylky Δy.

Průběh odchylky a koeficientu citlivosti našeho jednoduchého příkladu pro $b = 1, a = 2$ a $\Delta a = 0.2$ jsou na obrázku.

Poznámka:
Obrázek obsahuje i výstupní průběh pro $a = a_0 + \Delta a$

Příklad: Průběh koeficientu citlivosti



Shrnutí

Poznámky:

- Koeficienty citlivosti lze počítat i pro diskrétní systémy (SHO, ...)
- Citlivostní analýza
- Lze provádět i složitější analýzy, jejichž cílem je zjištění citlivosti systému na několik parametrů současně, na vlastnosti podsystémů apod.

Závěr:

Ukázali jsme pouze některé základní pojmy, s nimiž se v oblasti citlivostní analýzy setkáváme.

Optimalizační metody – úvod

- Operační výzkum — jak formulovat matematické modely složitých systémů a jak je analyzovat abychom získali přehled o možných řešeních.
- Optimalizace — jak nalézt takovou kombinaci hodnot parametrů modelu při daných omezeních (constraints), aby výsledek (daný hodnotící funkcí) byl nejlepší.
- Numerické optimalizační metody
- Použití: hledání (sub)optimálního řešení problému (obchodní cestující, počet strojů v továrně, nastavení parametrů stroje nebo výrobku, ...)

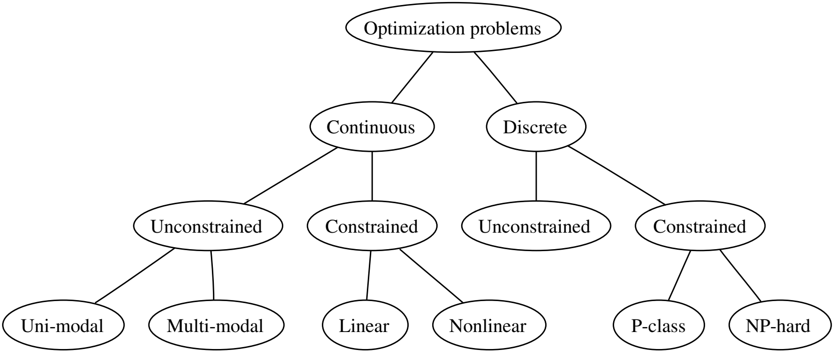
Optimalizační metody

Formulace problému:

- Cenová funkce: $f(x_1, ..., x_n)$=*min* (nebo *max*),
- Omezení (*constraints*):
  $g_i(x_1, ..., x_n)[\leq=\geq]b_i \quad i = 1, ..., m$
  kde f, $g_1, ..., g_m$ jsou dané funkce a $b_1, ..., b_m$ konstanty.

- Linearita/nelinearita f a/nebo g definuje zda jde o problém lineárního programování (LP) nebo nelineárního programování (NLP). *Pozor – pojem "programování" se používá z historických důvodů a nesouvisí s programováním počítačů.*
- Diskrétní/spojité parametry — pojmy *"Integer program", "Mixed-integer program"*
- Vícekriteriální optimalizace (*multiobjective optimization*)
- Nástroje: často vestavěné do sim. systémů (toolboxy,...)

Optimalizace – základní přehled



Optimalizační metody

- Heuristické metody
- "Simplex method" – pro LP
- "Interior point method" – pro LP
- Diskrétní: hledání nejkratší cesty v grafu, ...
- Gradientní metody
- Newtonova metoda (derivace f je nulová), ...
- Simulované žíhání (simulated annealing)
- Genetické algoritmy

Poznámka:
Většina inženýrských problémů je nelineární (některé je ale možné linearizovat)

Gradientní metody

1. Inicializace: startovací bod $\vec{x}_0$, cílová tolerance $\epsilon > 0$
2. Výpočet gradientu hodnotící funkce: $\nabla f(\vec{x}_i)$
3. Test ukončení: když norma gradientu $||\nabla f(\vec{x}_i)|| < \epsilon$
4. Směr posunutí: $\Delta\vec{x}_i = \pm\nabla f(\vec{x}_i)$
5. Posunutí: řešíme jednorozměrný problém hledání minima/maxima funkce $f(\vec{x}_i + \lambda\Delta\vec{x}_i)$
6. Nový bod: $\vec{x}_{i+1} = \vec{x}_i + \lambda_{i+1}\Delta\vec{x}_i$
7. $i = i + 1$ a jdeme na bod 2.

Poznámka: Problém lokálního minima, konvergence, ...

Simulované žíhání

1. Inicializace: startovací bod $\vec{x}_0$, počáteční "teplota" $t > 0$, limit počtu iterací `IMAX`, ...
2. Test ukončení: dosaženo `IMAX`, nebo nebylo nalezeno lepší řešení...
3. Náhodně zvolený bod posunutý o $\Delta x \in \mathcal{M}$: hodnotící funkce se změní o Δobj
4. Pokud je nové řešení lepší nebo s pravděpodobností $e^{\Delta obj/t}$ i pokud je horší ($\Delta obj \leq 0$), přijmeme nový bod: $\vec{x}_{i+1} = \vec{x}_i + \Delta x$ Jinak se vracíme k bodu 3.
5. Jestliže nové $\vec{x}_{i+1}$ je lepší, zapamatujeme si jej jako prozatím nejlepší výsledek.
6. Zmenšení "teploty" t pokud byl proveden dostatečný počet iterací.
7. $i = i + 1$ a jdeme na bod 2.

Genetické algoritmy

1. Inicializace: volba velikosti populace p, vytvoření p počátečních hodnot $\vec{x}$, max. počet generací `IMAX`, rozdělení populace na elitu p_e, imigranty p_i a "crossovers" p_c.
2. Test ukončení: po zadaném počtu generací `IMAX` vrátíme nejlepší řešení v populaci.
3. Elita zůstává nezměněna i v další generaci.
4. Imigranti: Přidáme p_i libovolných nových imigrantů do populace v další generaci.
5. Crossovers: vybereme $p_c/2$ dvojic ze stávající generace i a provedeme křížení dvojic (např. náhodnou kombinací chromozomů) – tím vznikne nová generace
6. $i = i + 1$ a jdeme na bod 2.

Ant Colony Optimization – ACO

Zjednodušený pseudokód algoritmu (speciálně pro TSP):

1. Inicializace: počet "mravenců" (agentů), feromonové stopy
2. Test ukončení: po zadaném počtu iterací `IMAX` vrátíme nejlepší nalezené řešení.
3. Každý mravenec vyhodnotí nejlepší směr dalšího postupu (heuristika, která použije i feromonové stopy).
4. Každý mravenec provede následující krok.
5. Aktualizace feromonových stop pro každý úsek:

   $$f = (1 - \rho)f + \sum_m \Delta f_m$$

   kde ρ je koeficient odpařování feromonů a m jsou mravenci, kteří prošli daný úsek.
6. Jdeme na bod 2.

Příklady

Viz literatura:

- Rardin R.: Optimization in Operations Research, Prentice Hall, 1998
- MIT OpenCourseWare: 16.888 / IDS.338J Multidisciplinary System Design Optimization, Spring 2010

Celulární automaty (CA) – úvod

- Princip CA – opakování
- Varianty CA:
  - diskrétní a deterministické (většina CA)
  - stochastické (např. *Lattice-Boltzman*)
  - spojité
- Použití CA:
  - Zpracování obrazu, generování textur, fraktály
  - Doprava, sypání písku, sněhové závěje
  - Šíření epidemie, požáru, vln, trhlin, ...
  - Difůze, růst krystalů
  - Proudění tekutin, turbulence
  - Chemické reakce
  - Umělý život, evoluce
  - ...
- Souvislosti: chaos, složitost, přírodní CA, kryptografie

Definice CA

Typicky jde o diskrétní systém:

- Buňka (Cell): obsahuje stav
- Pole buněk (Lattice): obecně n-rozměrné
- Okolí (Neighbourhood)
- Pravidla (Rules): Funkce stavu buňky a jejího okolí definující nový stav buňky v čase:

$$s(t+1) = f(s(t), N_s(t))$$

Různé typy pravidel:

"legal" – z nulového vstupního stavu nesmí vzniknout nenulový

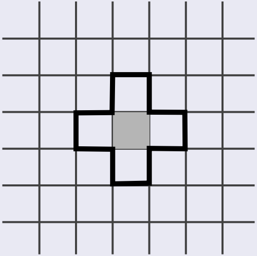
"totalistic" – rozhoduje *součet* vstupních stavů

variaty: CellDEVS, dynamické, stochastické, ...

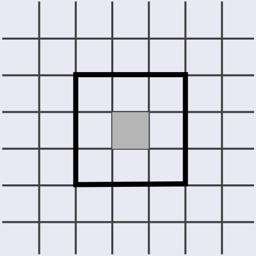
Typy okolí

1D, 2D, 3D

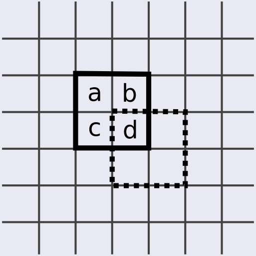
Některá 2D okolí:



Von Neumann



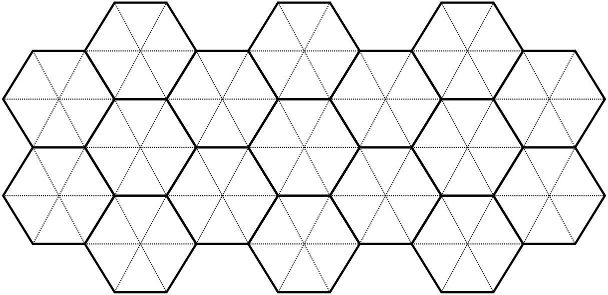
Moore



Margolus

Typy okolí – pokračování

- Šestiúhelníkové okolí



Poznámky:

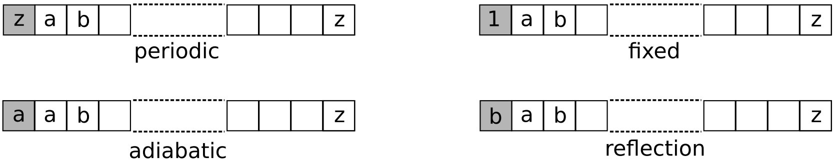
Použití: např. růst sněhových vloček, šíření vln (FHP)

Implementace: převod šestiůhelníková → čtvercová struktura

Okrajové podmínky

Problém hranic při omezené velikosti pole buněk

- Periodické (periodic): topologie typu kružnice/toroid
- Pevné (fixed): konstantní hodnota na hranici
- (adiabatic): hodnota vedlejší buňky (nulový gradient)
- (reflection): hodnota předposlední buňky



Třídy CA

CA můžeme rozdělit podle jejich dynamického chování do 4 kategorií:

- třída 1: Po konečném počtu kroků dosáhnou jednoho konkrétního ustáleného stavu
- třída 2: Dosáhnou periodického opakování (s krátkou periodou) nebo zůstanou stabilní.
- třída 3: Chaotické chování (výsledné posloupnosti konfigurací tvoří speciální fraktální útvary).
- třída 4: Kombinace běžného a chaotického chování (například Life), nejsou reverzibilní.

Poznámka: Viz Wolfram S.: NKS

Metody implementace

Implementace uložení buněk

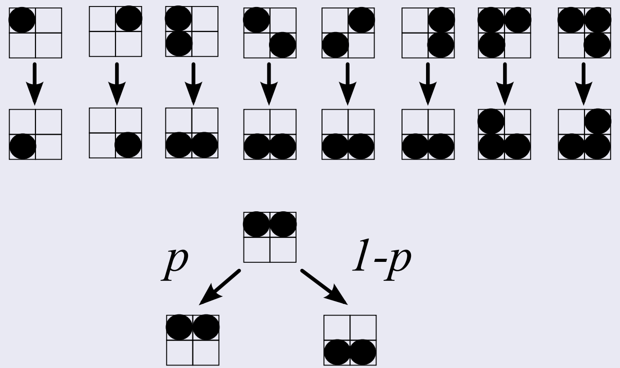
- Přímá: každá buňka uložena zvlášť (`pole[i]`)
- Vyhledávací tabulka: uloženy jen ”nenulové” buňky
- SIMD styl (”multispin coding”): více buněk v jednom prvku pole (např. 32 buněk v `int` + bitové operace)
- ”Hash life”: cache + quadtree
- ...

Poznámka: Snadno paralelizovatelné

Příklad – sypání písku

Sypání písku a podobných materiálů – *”sand rule”, ”sandpile”*

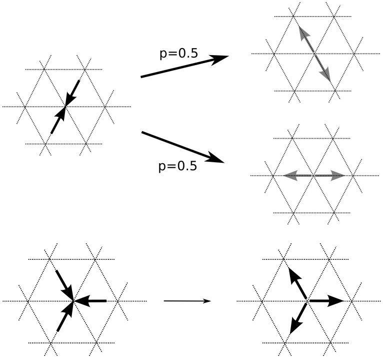
- Okolí typu Margolus
- Pravidla:



Příklad: FHP

Model pohybu tekutiny:

- Šestiúhelníkové okolí
- Buňky obsahují částice a jejich směr pohybu
- Pravidla viz obrázky + volný průlet v ostatních případech



Další aplikace CA

- Doprava
  - pravidlo 184
  - Nagel-Schreckenberg
  - modely křižovatek
- Řešení parciálních dif. rovnic (např. šíření signálu)
- Generování pseudonáhodných čísel (pravidlo 30)
- Pohyb pevných těles: deformace, pružnost
- Statistická mechanika: Lattice Boltzmann
- Biologie: růst organismů
- Krystalografie: růst krystalů
- ...

"Agent-based simulation"

- Definice pojmů: agent, prostředí, ...
- Principy
- Souvislosti s umělou inteligencí
- Oblasti použití:
  modely dopravy,
  biologické, ekonomické, socio-technické, ...
- Nástroje
- ...

Definice pojmů

Agent ["Agere" = latinsky jednat (za někoho)]:

- Podle literatury [Weiss, p. 29]:
  Agent je počítačový systém umístěný do nějakého prostředí, který je schopen samostatné činnosti v tomto prostředí aby dosáhl určitých cílů.
- Podle [Russell and Norvig, p. 7:]
  Agent je cokoli co sleduje okolí a něco dělá.

Definice pojmů

- Podle [Ferber, p. 9]:
  Agent je fyzická nebo virtuální entita, vykazující následující vlastnosti:
  - schopnost pracovat v daném prostředí,
  - může komunikovat s ostatními agenty,
  - je řízen množinou tendencí vedoucích ke splnění jeho cílů,
  - disponuje vlastními prostředky,
  - je schopen sledovat okolní prostředí,
  - má jen částečnou reprezentaci okolního prostředí,
  - má jisté schopnosti a může poskytovat služby,
  - může mít schopnost se rozmnožovat.

Definice pojmů

- Prostředí (environment)
  Okolí agenta – podobné jako okolí modelu se vstupy a výstupy.
- Poznámky

Simulace

Můžeme rozlišit dvě významné varianty simulace s agenty:

- Agenti komunikují přes sdílené prostředí.
  Stav prostředí definuje stav celého systému. Každý agent má částečné znalosti prostředí přes svoje vstupy a na jejich základě mění prostředí přes svoje výstupy – provádí specifické akce.
- Agenti komunikují navzájem prostřednictvím zpráv. Agenti a jejich stav určují stav celého systému. Změny stavu se šíří komunikací mezi agenty.

Nástroje

Agenti se používají v celé řadě aplikací – nejen v simulaci.

- Swarm – Java, multi-agent simulace
- další – viz WWW

Neurčitost (uncertainty)

Při práci s reálnými systémy se stěží můžeme vyhnout neurčitosti. Neurčitost je součást téměř každého měření (chyby měření), na úrovni popisu znalostí se vyskytuje nejednoznačnost přirozeného jazyka, ve společnosti je často používána k různým účelům (utajení, ...).

- Typy neurčitosti
- Formy popisu neurčitosti
- Použití v simulaci
- Nástroje

Literatura: Klir G.: Uncertainty and Information, Willey, 2006

Způsoby reprezentace neurčitosti

Podle aplikačních oblastí [Klir]:

- *possibility/necessity*: vyjádření neurčitosti jako jedné z N možností. Velké N vede k méně specifickým předpovědím než malé N. Pokud N=1 nejde o neurčitost.
- Klasická teorie pravděpodobnosti: Míra neurčitosti vyjádřená pravděpodobností (0..1) výskytu alternativy v dané podmnožině všech možných jevů.
- Teorie fuzzy množin: umožňují vyjádření nespecifičnosti a vágnosti. Vágnost se liší od nespecifičnosti tím, že vychází z nepřesnosti definic (především z hlediska jazyka). Příslušnost do fuzzy množin nevyjadřuje potvrzení/odmítnutí ale stupeň významnosti.

Způsoby reprezentace neurčitosti 2

- ”Fuzzy measure theory”: viz literatura
  Tato teorie se značně liší od teorie fuzzy množin – podmínky příslušnosti do množin jsou přesné, ale informace o prvcích nepostačuje k určení příslušnosti.
- ”Rough set theory”: viz literatura
  Formální aproximace klasické množiny párem množin, které definují dolní a horní mez (lower/upper approximation). Tyto množiny mohou být i ”fuzzy”.

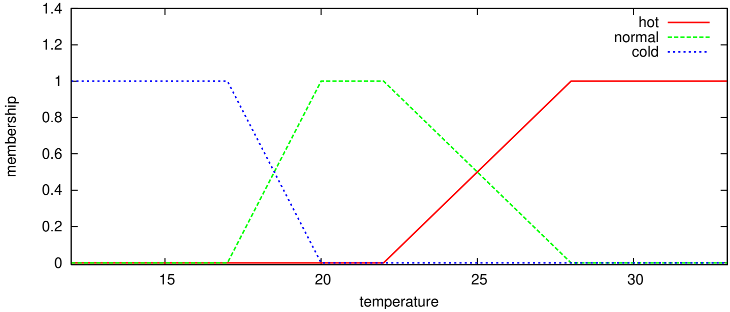
Fuzzy logika

- Rozšíření Booleovské logiky, (1965, Lofti Zadeh)
- Popisuje ”vágnost” ve vyjadřování. (např. Co přesně znamená, že je teplota ”vysoká” nebo ”nízká”?)
- Pojmy: fuzzy množina, funkce příslušnosti
- Míra příslušnosti do fuzzy množiny je číslo od 0 do 1 (ale pozor – nesouvisí s pravděpodobností)
- Použití fuzzy logiky: řízení, expertní systémy, ...

Poznámka: Podrobnosti viz např. PDF na WWW:
Navara M., Olšák P.: *Základy fuzzy množin*, ČVUT, Praha, 2002

Fuzzy množina, funkce příslušnosti

Příklad: teplota v místnosti, 3 fuzzy množiny:
malá – střední – velká (cold – normal – hot)



Fuzzifikace: převod "ostré" hodnoty na příslušnost do množin
(Příklad: 18 $^\circ\mathrm{C} \rightarrow$ cold:0.5, normal:0.5, hot:0)

Operace fuzzy logiky

Negace: $\neg_s \alpha = 1 - \alpha$

Konjunkce: $\alpha \bigwedge_s \beta = min(\alpha, \beta)$

Disjunkce: $\alpha \bigvee_s \beta = max(\alpha, \beta)$

kde α a β jsou funkce příslušnosti

Poznámky:

- Existují i jiné definice operací
- Pokud budou hodnoty funkce příslušnosti omezeny pouze na 0 a 1, dostaneme Booleovu logiku.

Fuzzy blok

Postup vyhodnocování:

- převod vstupu na fuzzy hodnoty (fuzzification)
- pravidla (if-then rules)
  Příklad pravidel:

```
IF teplota IS malá    THEN topení=hodně
IF teplota IS střední THEN topení=málo
IF teplota IS velká   THEN topení=chladit
```

- spojení výstupů pravidel (aggregation)
- převod na "ostré" hodnoty (defuzzification)

Nástroje

- Matlab: Fuzzy Logic Toolbox
- SciLab: sciFLT (Fuzzy Logic Toolbox)
- Dymola/Modelica: Fuzzy Control Library
- ...

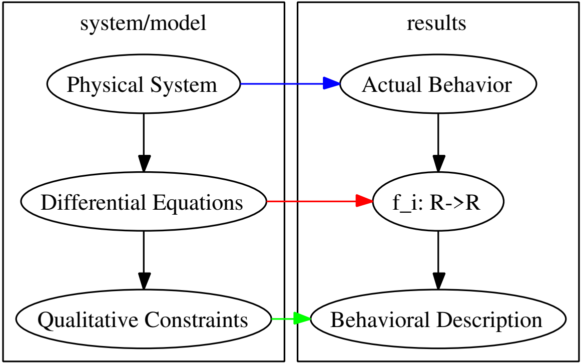
Kvalitativní simulace

Použitelná v případech, kdy neznáme dostatečně přesné numerické informace o řešeném problému.

- Kvalitativní informace, proměnné
- Kvalitativní diferenciální rovnice
- Cílem je získat kvalitativní informace o chování systému (například: "hladina vody se ustálí a bude na vyšší úrovni").
- Může být doplněno kvantitativní informací – semikvantitativní modely.
- Nástroje: QSIM algoritmus, ...

Poznámka: Klíčová jména pro hledání: Kuipers, Kleer, Forbus

Souvislosti



Princip

Algoritmy pro práci s kvalitativními údaji musí definovat:

- jak popsat kvantitu kvalitativně: například $(-, 0, +)$, počet hodnot se může při simulaci zvětšovat viz [Kuipers]
- jak se provádí změna stavu modelu: pravidla pro výpočet následujících stavů
- (zda kvantitativní údaje odpovídají matematické analýze)
- zda kvalitativní simulace poskytuje všechna (a platná) chování pro třídu systémů popsaných modelem.

Chování systému je popsáno stromem nebo grafem kvalitativních stavů. Větvení znamená, že stav má více možných následníků (to je způsobeno nepřesností v popisu modelu).

Způsoby reprezentace hodnot

Kvalitativní hodnota v proměnné v je dvojice $(qmag, qdir)$:

- *qmag* množina totálně uspořádaných hodnot nebo intervalů mezi dvěma hodnotami. Počet hodnot je obvykle velmi malý.
- *qdir* směr změny hodnoty (znaménko derivace) – jedna z hodnot *inc* (rostoucí), *std* (ustálená), *dec* (klesající).

Kvalitativní stav modelu je dán hodnotami všech n proměnných – rozměr stavového prostoru je $2 * n$.

Hodnoty stavu jsou definovány v konkrétních časových okamžicích nebo v intervalech.

Kvalitativní dif. rovnice

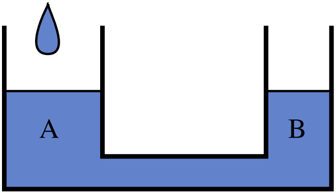
QDE je abstrakce ODE a lze ji definovat jako čtveřici:

$$< V, Q, C, T >$$

kde:

- V je množina proměnných
- Q množina prostorů hodnot proměnných
- C množina omezujících rovnic (constraints)
- T množina přechodů definujících hranice použitelnosti QDE

Příklad – Spojené nádoby



Omezení:

- Tlak vody je úměrný výšce hladiny
- Množství vody protékající mezi nádobami závisí na rozdílu tlaků
- Průtok snižuje množství vody v jedné nádobě a zvyšuje ve druhé

Příklad – pokračování

Chování systému:

- Na začátku je systém v rovnováze a přidáme vodu do nádoby A
- Zvýší se množství vody a tlak v nádobě A, což vede k průtoku do B
- Voda teče z A do B, výška hladiny A klesá, B roste. Rozdíl tlaků i průtok vody klesají k nule.
- Systém se ustálí když rozdíl tlaků i průtok klesne na nulu, hladiny v obou nádobách jsou vyšší než na začátku.

Kvalitativní simulace určí chování systému a nepotřebuje počáteční výšku hladiny, ani množství přidané vody.
Nezjistíme o kolik se zvýšila hladina ani jak dlouho proces ustalování stavu trval.

Algoritmus QSIM

1. Nastavit počáteční stav do seznamu ACTIVE
2. While(not empty(ACTIVE)) opakovat následující body:
3. Vybrat kvalitativní stav z ACTIVE
4. Pro každou funkci vybrat množinu možných následujících stavů
5. Pro každou rovnici (constraint) generovat množinu n-tic přechodů jejich argumentů. Vyřadit nekonzistentní n-tice.
6. Test konzistence množin n-tic pro všechny rovnice v systému
7. Generovat všechny možné interpretace zbylých n-tic (pokud žádné nezbyly, je chování nekonzistentní). Vytvořit nové stavy vzniklé interpretací a nazvat je následníky aktuálního stavu.
8. Vyřadit stavy nevyhovující pravidlům a zbývající přidat do ACTIVE.

Nástroje, Literatura

- QSIM algoritmus
- Kuipers B.: Qualitative Simulation, Artificial Intelligence 29, p. 289-338, 1986
- ...

Multimodely

= modely složené z jiných modelů.

- Většina reálných modelů jsou multimodely (složité modely obvykle nelze popsat jedním formalismem).
- Modelujeme na různých úrovních abstrakce a různými způsoby – *heterogenní* popis multimodelů.
- Skládání modelů z komponent – *hierarchické* modely.
- Nástroje: většina simulačních systémů

Úrovně abstrakce

- Konceptuální modely
- Kvalitativní modely
- Modely s neurčitostí (fuzzy, stochastické, ...)
- Spojité a diskrétní modely
- (Fyzikální modely)
- Reálný systém (přesné výsledky)

Poznámka: Při vytváření modelu obvykle zpřesňujeme popis.

Příklady

Literatura

- Fishwick P.: Simulation Model Design and Execution, Prentice Hall, 1995
- ...

Paralelní a distribuovaná simulace

- Paralelní zpracování
- Distribuované zpracování
- Souvislosti: Architektury počítačů, algoritmy
- Paralelizace simulačních programů (modelů)
- Typy modelů vhodné pro paralelizaci
- Paralelní simulátory, synchronizace

Poznámka: Superpočítače

Nástroje

- OpenMP, PVM, MPI, Java RMI, CORBA, ...
- HLA – High Level Architecture (CORBA)
- Multi-Simulation Interface – MSI (XML)
- Běžné simulační nástroje podporují i paralelizaci výpočtu (SciLab, MatLab, Dymola, ...)

Přehled viz například [Fishwick], [Fujimoto]

Aplikace

- Vojenské: hry, trenažéry, testování HW
- Zábava: hry (let. simulátory, MUD)
- Výuka
- Telekomunikace: modely sítí, komponent
- Číslicové systémy: VHDL, ...
- Doprava
- ...

Poznámky: ”virtual environment”, VR

Paralelní počítače

- Flynnova klasifikace (1972):
  - SISD - jeden procesor
  - SIMD - vektorové (CRAY) / "array"
  - MIMD - distribuovaná/sdílená paměť, cluster
  - MISD - málo používané
- Přístup do paměti: UMA(SMP) / NUMA / NoRMA(cluster)
- Topologie sítě (statická/dynamická), parametry sítě

Paralelizace výpočtu

- Explicitní: knihovny
- Implicitní: speciální jazyky a překladače
- Paradigmata:
  - Imperativní: Fortran, C, C++
  - Logické: Prolog
  - Funkcionální: Lisp, Scheme

Poznámka: OpenMP

MPI - Message Passing Interface

- Explicitní paralelizace
- Vhodné pro distribuované systémy, clustery
- Nezávislé na prog. jazyku (C, Fortran90, C++)
- Standardní API:
  - MPI-1 (1992-1994)
  - MPI-1.2 (1996) - statické, minimální
  - MPI-2 (1996) - dynamické, náročnější, paralelní vstup/výstup
  - MPI-2.1 (2007)
  - MPI-3.1 (2015), ...
- \+ výkon, škálovatelnost, přenositelnost
- \- obtížné, příliš nízkoúrovňové
- Implementace: MPICH, Open MPI

PVM - Parallel Virtual Machine

- Explicitní paralelizace
- Vhodné pro heterogenní počítačové systémy
- Použitelné pro C, Fortran, C++
- Verze:
  - Verze 1 (1989)
  - Verze 2 (1991)
  - Verze 3 (1993) - přenositelnost, odolnost proti chybám
    Aktuální verze 3.4.6 (2009)
- \+ škálovatelnost, přenositelnost, flexibilita
- \- výkon?
- Implementace: PVM

CORBA

- Common Object Request Broker Architecture
- Objektově orientovaný distribuovaný systém
- Umožňuje volání metod objektů nezávisle na jejich umístění
- Použitelné pro C, C++, Lisp, Smalltalk, Java, Python, ...
- IDL - Interface Description Language
- ORB - Object Request Broker
- Verze:
  - Verze: 1.0 (1991), 2.0 (1996), 3.0 (2002)
  - Aktuální verze 3.4 (2021)
- Implementace: omniORB, GNOME/ORBit, ...

Poznámky: paralelizace modelů

- Deklarativní modely
- Funkcionální modely
- Constraint modely (rovnice) – převod na funkcionální (blokový) popis.
- Prostorové (spatial) modely – jejich složitost často vyžaduje paralelizaci.

Algoritmy

- Lokální virtuální čas – pro každý procesor
- Komunikace prostřednictvím zpráv
- Zprávy nesou také informaci o modelovém čase

Metody paralelní simulace:

- Konzervativní – uspořádání zpráv, provádění událostí v pořadí podle času aktivace
- Optimistická – anti-zprávy rozeslané ostatním procesorům mohou rušit efekt již provedených operací

HLA – High Level Architecture

Software pro vytváření komponentních distribuovaných simulací

- obecné
- podporuje *znovupoužitelnost* komponent
- možnost propojování různých nástrojů
- často real-time simulace
- Standard: DoD, NATO, OMG, IEEE 1516
- Vylepšení starších standardů: DIS, ALSP

HLA – terminologie

- ”federate” – základní jednotka (model+simulátor)
- ”federation” – skupina jednotek tvořící simulaci
- RTI – Runtime Infrastructure – komunikační vrstva (existují různé implementace RTI) poskytuje rozhraní pro služby rozdělené do kategorií:
  - Federation management (20)
  - Declaration management (12)
  - Object management (17)
  - Ownership management (16)
  - Time management (23)
  - Data distribution management (13)

HLA – terminologie

- Každá služba RTI je specifikována položkami:
  - Jméno a popis
  - Parametry
  - Návratové hodnoty
  - Pre- a Post-conditions
  - Výjimky
  - Související služby
- HLA pravidla, OMT-Object Model Template: FOM-Federation, SOM-Simulation(=federate), MOM-Management Object Model
- Existují API pro: CORBA IDL, C++, Ada, Java

HLA – čas

- Logický – federate time (při TSO), každá jednotka může mít jiný, interní zpracování času v jednotce nesmí být viditelné z okolí
- Reálný
- ...
- RTI – Time management:
  - žádná správa času – každá jednotka pracuje nezávisle, například v reálném čase
  - konzervativní – posuv času pouze když je zaručeno, že nebude problém
  - optimistická – možnost "Roll-back"
  - "activity scan" – posun času po vzájemné dohodě jednotek, např. konstantní časový krok

HLA – čas 2

Pořadí zpracování zpráv:

- TSO – Time Stamp Order
  RTI zaručí, že nedojde k příjmu zpráv z minulosti
  Musí být explicitně zapnuto příslušnými službami.
- RO – Receive Order

Požadavky na posun času:

- Spojitá simulace: "Request Time Advance", RTI odpoví, zda je to možné "Time Advance Grant"
- Diskrétní simulace: Služba "Next Event Request (t1)" požaduje posun času na další událost nebo na t1,podle toho co nastane dříve. RTI odpoví kdy posunout čas a o kolik zprávou "Time Advance Grant"

HLA – čas 3

Simulace řízená reálným časem:

- speciální případ
- nutnost synchronizace hodin
- real-time zpracování
- kompenzace komunikačního zpoždění (například predikce pohybu na základě předchozí pozice a rychlosti)
- vše musí implementovat jednotka - ”federate”

MSI – Multi-Simulation Interface

- Alternativa k HLA
- Propojování více simulací (CSIM,...)
- Synchronizace času
- Ve srovnání s HLA je:
  - jednodušší a menší,
  - méně náročné na přenos dat,
  - umožňuje hierarchické simulace
- Použitelné s různými jazyky (např. C, C++, Java, Lisp, Perl)

Parallel DEVS formalismus

- Proč Parallel DEVS
- Souběžné provádění událostí
- Rozšíření formalismu
- *Parallel DEVS Simulation Protocol*
- ...

Literatura:

Zeigler, B. et al.: Theory of Modelling and Simulation, AP, 2000
Nutaro, J.: Building Software for Simulation, Wiley, 2011

Proč Parallel DEVS?

- Problém výskytu současných (vstupních a interních) událostí
- Provedení $\delta_{int} \to \delta_{ext}$ nebo naopak?
- Klasický DEVS problém řeší pomocí funkce *Select* v DEVN, ale v případě zpětné vazby může dojít k problému – např. ztráta vstupní události.
- Paralelní DEVS rozhodování o pořadí současných událostí přesouvá do atomického DEVS, kde je lépe řešitelné.
- Vhodné pro paralelní implementaci simulátoru

Definice Parallel DEVS

$$PDEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

kde

- X je množina vstupů,
- Y je množina výstupů,
- S je množina stavů,
- $\delta_{ext} : Q \times X^b \rightarrow S$ je externí přechodová funkce kde $Q = \{(s, e) : s \in S, 0 \leq e \leq ta(s)\}$ je množina totálních stavů,
- $\delta_{con} : S \times X^b \rightarrow S$ je "confluent" přechodová funkce pro současný výskyt vstupní a interní události ($e = ta(s)$). Implicitně se používá $\delta_{con}(s, x) = \delta_{ext}(\delta_{int}(s), 0, x)$. Vždy platí $\delta_{con}(s, \emptyset) = \delta_{int}(s)$.

Definice DEVS formalismu – pokračování

- $\delta_{int} : S \rightarrow S$ je interní přechodová funkce,
- $\lambda : S \rightarrow Y^b$ je výstupní funkce
- $ta : S \rightarrow \mathcal{R}_0^+ \cup \{\infty\}$ je funkce posunu času udávající čas zbývající do výskytu následující plánované události.

Poznámka:
X^b, Y^b jsou množiny multimnožin (*bag*), protože je možné současné zpracování více událostí

Hierarchický model — složený DEVS

Složený paralelní DEVS (PDEVN) je definován takto:

$$PN = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\})$$

kde všechny položky odpovídají DEVN, pouze *Select* chybí.

Paralelní DEVS simulátor

Simulátor pro PDEVS model má jinou strukturu než DEVS simulátor, je bez root-koordinátoru.

Implementace: adevs

Modelování číslicových systémů

- Úrovně popisu
- Algoritmy
- Jazyky VHDL, Verilog, SystemC
- Oblasti použití:
  - Návrh integrovaných obvodů
  - Diagnostika, testování
  - Výuka
- Souvislosti: vizualizace, paralelní simulace

Popis číslicových systémů

Úrovně popisu:

- Elektrická – tranzistory, rezistory, kondenzátory (spojité modely)
- "Switch" – tranzistory nahrazeny spínači, obousměrné signály, RC články (diskrétní)
- Logická ("gate") – hradla, klopné obvody (diskrétní modely)
- RTL (meziregistrové přenosy) – čítače, řadiče, ALU (diskrétní modely)
- Algoritmus – popis programem (diskrétní modely)
- Systémová – procesory, paměti, periferie (hromadná obsluha, výkonnost)

Jazyky

Specializované nástroje:

- SPICE: elektrická úroveň (časová, stejnosměrná, frekvenční, citlivostní analýza)
- VHDL: logická, RTL, algoritmus
- VHDL-AMS (IEEE Std 1076.1): + elektrická
- Verilog: logická, RTL, algoritmus
- Verilog-AMS: + elektrická úroveň
- System C: TLM = "Transaction Level Modelling"
- System C - AMS (IEEE Std 1666.1): + elektrická
- ...

Poznámka: Volně dostupné: ngspice, IRSIM, Icarus Verilog, ...

Modely signálů

- Dvouhodnotové: jen 0 a 1 (málo používané, rychlé)
- Trojhodnotové: 0, 1, X (neurčitá úroveň)
- Hodnoty 0, 1, X, R (Rise) a F (Fall)
  přesnější popis, odhalí více hazardů, pomalejší

Další možnosti:

- Hodnota Z (vysoká impedance)
- Různá "síla" signálu (typické u CMOS)
- Statický (_/_) a dynamický (_/\/~) hazard

Modely signálů — VHDL

VHDL definuje výčet `std_logic`

- ’U’: Uninitialized. This signal hasn’t been set yet.
- ’X’: Unknown. Impossible to determine this value/result.
- ’0’: Logic 0
- ’1’: Logic 1
- ’Z’: High Impedance
- ’W’: Weak X signal
- ’L’: Weak 0 signal (should probably go to 0)
- ’H’: Weak 1 signal (should probably go to 1)
- ’-’: Don’t care.

Modely zpoždění

Zpoždění logických členů:

- 0 – nulové (vhodné jen pro ověření log. funkce)
- 1 – jednotkové
- t_d – pevné (nastavitelné zvlášť pro 0→1 a 1→0)
- $\langle t_1, t_2 \rangle$ — proměnné (rozsah od-do, vyžaduje hodnoty signálu R,F)

Zpoždění na spojích:

- nulové
- nastavitelné (Problém: délky spojů jsou k dispozici až po rozmístění prvků.)

Poznámka: Kontrola časování (např. dodržení předstihu a přesahu u klopných obvodů)

Modely chování komponent

- Rovnice (Booleovské)
- Pravdivostní tabulky
- Podprogram (popis algoritmu)
- Propojení komponent (hierarchické)

Algoritmy řízení simulace

- Řízení událostmi – problém velkého množství událostí v kalendáři. Jen málo hradel mění stav současně – počítáme nový výstup jen pro ta, která se mění (tzv. selektivní sledování). Nevýhoda - režie sledování změn.
- Simulace s pevným krokem – vyhodnocují se vždy všechny prvky (není potřeba kalendář, předpokládá uspořádání prvků, problém zpětných vazeb u sekvenčních obvodů).

Implementace:

- Tabulkový model (simulace = interpretace grafu)
- Kompilovaný model (pro synchronní obvody, je rychlejší, neodhalí problémy časování)

Algoritmus řízení simulace

Dvoufázový algoritmus, selektivní sledování:

```
inicializace
plánování události pro nový vstupní vektor
while (je plánována událost) {
   nastavit hodnotu modelového času na T
   foreach (u in plánované události na čas T) {
      výběr záznamu události u z kalendáře
      provedení u (aktualizace hodnoty signálu)
      přidat všechny připojené prvky do množiny M
   }
   foreach (p in množina M) {
      vyhodnocení prvku p
      if (změna jeho výstupu)
         plánování události/změny
   }
```

Model popsaný tabulkami/grafem

Popis modelu:

- Tabulka prvků (TP) — pro každý prvek včetně primárních vstupů/výstupů obsahuje záznam: (typ, zpoždění, počet vstupů, počet spojů z výstupu, odkazy do dalších tabulek, hodnota výstupu).
- Tabulka vstupů (TV) — popisuje propojení vstupů na prvky
- Tabulka výstupního větvení (TVV) — kam vedou spoje z výstupu

Práce s časem

- Pevný krok – závisí na modelu zpoždění:
  - Jednotkové zpoždění: dva seznamy pro aktuální čas a pro příští čas, po zvýšení času se zamění.
  - Složitější model zpoždění: "časová mapa" – pole seznamů

  Problém: volba vhodného kroku.

  Výhodou je jednoduchost a rychlost.
- Next-event s kalendářem událostí
  - Složitější, náročné na implementaci kalendáře
  - Výhodou je obecnost – lze použít jakékoli zpoždění.

Simulace poruch

Použití při testování logických obvodů – vyhodnocování kvality diagnostických testů (pokrytí testu).

Modely poruch:

- Trvalá 0
- Trvalá 1
- Zkrat mezi vodiči

Činnost:

- Specifikace poruch – které poruchy budou simulovány
- Injekce poruch – transformace modelu na model s poruchou
- Šíření poruch modelem (často se používají zjednodušené modely)
- Detekce poruch – projeví se na funkci?
- Zpracování výsledků – vytvoření podkladů pro řízení testů...

Simulace poruch – optimalizace

Opakovat všechny kroky pro každou poruchu je časově velmi náročné – používají se různé optimalizace

- Paralelní simulace poruch: vhodná pro základní logické členy, používá bitové kódování simulačních hodnot do jedné proměnné a zpracování několika najednou.
  Například u dvouhodnotových signálů můžeme jedinou instrukcí AND zpracovat 32 hodnot.
- Deduktivní: viz literatura
- Souběžná: referenční model doplníme o souběžné modely reprezentující *změny* v chování modelu s poruchou. Každý prvek referenčního modelu má seznam poruchových prvků, které se také simulují a dynamicky vznikají/zanikají podle toho, zda se jejich hodnoty liší od referenčních.

Jazyky pro popis číslicových systémů

- VHDL
- Verilog
- SystemC

Poznámka: IEEE standardy

VHDL

Vhodné pro složité systémy
Syntaxe odvozena z jazyka Ada

Úrovně popisu (lze kombinovat):

- Popis struktury – propojení hradel
- Popis chování
  - algoritmem – proces
  - data flow – RTL (Register Transfer Level)
    např. `o <= transport i1 + i2 after 10 ns;`

Knihovny prvků

Verilog

Syntaxe odvozena z jazyků Ada a C

- Procedurální popis chování
  - Pro úrovně: algoritmu a RTL
  - Procedury: funkce a úlohy (”tasks”, nenulová doba provádění)
  - Výrazy: zpoždění (delay), podmínky (event expr.)
- Strukturální popis
  - Pro úrovně: logická, ”switch-level”
  - Spínače a obousměrné spoje
  - Možnost modelovat MOS obvody s nábojovými strukturami
  - Síla signálu + různé neurčité hodnoty signálu
- Hierarchické modely
- Modularita

Verilog – příklad

```
module full_adder(a, b, cin, cout, sum);
    input   a, b, cin;
    output  cout, sum;
    wire tmp;
    tmp = a ^ b;
    sum = tmp ^ cin;
    cout = (cin & tmp) | (a & b);
endmodule

wire [3:0] c;
wire [3:0] s;
full_adder fa1(a[0], b[0],  'b0, c[0], s[0]);
full_adder fa2(a[1], b[1], c[0], c[1], s[1]);
full_adder fa3(a[2], b[2], c[1], c[2], s[2]);
full_adder fa4(a[3], b[3], c[2], c[3], s[3]);
```

SystemC

”System description language”, ”Transaction-level modelling”

- Implementováno jako knihovna v C++
- Zahrnuje jazyk i simulátor
- Souběžné procesy implementují chování
- Kanály (Channels: signal, fifo, buffer, mutex, semaphore) slouží pro komunikaci
- Rozhraní, události, porty, moduly
- Datové typy: `sc_int<>`, `sc_bit`, ...

SystemC – příklad

```
#include "systemc.h"
#define WIDTH  4

SC_MODULE(adder) {
  sc_in< sc_uint<WIDTH> >   a, b;
  sc_out< sc_uint<WIDTH> >  sum;

  void do_add() {
    sum.write( a.read() + b.read() );
  }
  SC_CTOR(adder)      {
    SC_METHOD(do_add);
    sensitive << a << b;
  }
}
```

Vizualizace

(Scientific Visualization, * 1987)

- Obecné principy
- Oblasti použití vizualizace:
  - Modelování a simulace
  - GIS, meteorologie,
  - Software
  - Lékařství (Medical Imaging)
  - Architektura, umění, ...
- Poznámky: Animace, Virtuální realita
- Vizualizační nástroje pro simulaci

Typy vizualizace

- Vizualizace informací
- Vizualizace znalostí
- Vizuální komunikace
- Vizualizace ve výuce
- Vizualizace produktů (CAD)

Proces vizualizace

- Simulace – zdroj dat
- Filtr – výběr části dat (zrychluje vizualizaci)
- Mapper – konverze dat na abstraktní vizuální reprezentaci. (Většinou převod na primitivní geometrické útvary: úsečky, trojůhelníky, body, ...)
- Renderer – generování obrazu (uvažuje osvětlení, pozici kamery atd).
- Display – zobrazení

Vizualizační techniky

- Grafy (funkcí, histogramy, ...) 1D-3D
- Grafy (stromy, diagramy)
- Tabulky
- Mapy
- Izočáry (contours) 2D
- Izoplochy (isosurface) 3D
- Zobrazování objemů (volume visualization) 3D
- Vektory 2D, 3D
- Plochy 2D
- Částice 2D, 3D
- Řezy 3D
- Trasování částic 2D, 3D

(pokračování)

- Animace
- Vykreslování os
- Přiřazení barev (barva=$f(data)$)
- Ořezávání
- Popisy (Labels)
- Zvětšování, ...
- Interpolace

Nástroje pro vizualizaci

Stručný přehled, příklady – viz WWW

- VTK – knihovna pro C++
- OpenDX
- Graphviz – automatické kreslení grafů
- Gnuplot – jednoduchý
- ...

Simulace – poznámky

Návrh simulačních experimentů

Volba parametrů simulace

- délka simulačního běhu
- počet běhů
- úpravy omezující vliv náběhu systému
- ...

Analýza výsledků simulace

- Techniky aplikovatelné na diskrétní a MC simulaci
- Problém autokorelace u diskrétní simulace
- Intervalové odhady
- Redukce rozptylu – technika jak zvýšit přesnost bez zvyšování počtu vzorků

Intervalové odhady

Výsledky diskrétní simulace (statistiky) zahrnují neurčitost, kterou můžeme kvantifikovat

- Stanovíme míru spolehlivosti (např 0.95)
- Vypočteme interval ve kterém leží střední hodnota s touto mírou spolehlivosti (používá se Studentovo rozložení)
- Pokud potřebujeme zvýšit přesnost (zúžit interval) při stejné míře spolehlivosti, musíme zvýšit počet vzorků

Poznámka: Míra spolehlivosti $\neq$ pravděpodobnost

Simulační systém Dymola

Dymola (Dynamic Modelling Laboratory) = integrované prostředí pro modelování a simulaci
Modelica = jazyk pro popis modelů

- Objektová orientace, Hierarchické skládání modelů
- Knihovny znovupoužitelných komponent modelů pro různé aplikační oblasti: Modelica Standard Library, MultiBody, Hydraulics, Power Train, ...
- Propojování: konektory, ..., Grafický editor modelů
- Popis rovnicemi, automatické úpravy rovnic
- Zvládá velké a složité modely
- Rychlost - používá symbolické zpracování
- Možnost propojování s jinými systémy (Matlab,...)
- Animace 3D, Real-time simulace, ...

Poznámky: historie, OpenModelica, JModelica

Architektura systému Dymola

- Dymola program
- Import dat z externích programů (CAD,...)
- Editor modelů
- Knihovny modelů
- Symbolické zpracování
- Modelica, překlad do C, "dymosim"
- LAPACK, ...
- Řízení experimentů, zpracování výsledků
- Vizualizace, animace
- Možnost propojení na Matlab/Simulink

Dymola – závěr

- Komerční simulační program – problém ceny licence
- Dokumentace – manuál
- Modelica – existuje volně dostupná implementace
- Modelica Library – volně dostupná
- Celá řada knihoven – komerční i nekomerční
- Numerické metody – efektivita, tuhé systémy
- Používáno v průmyslu
- ...

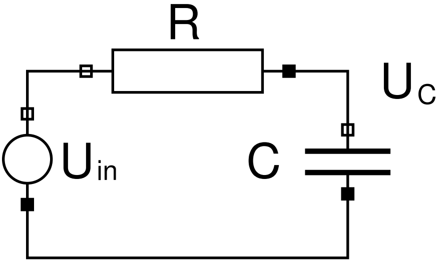
Modelica – úvod

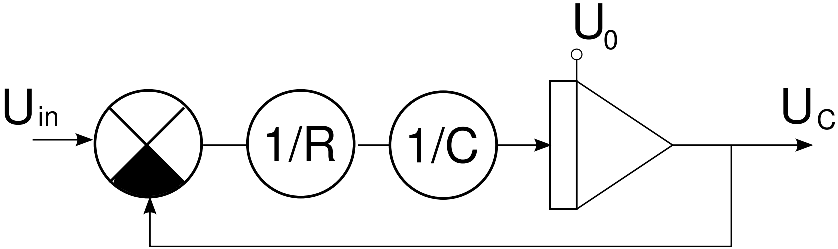
- Objektově orientovaný jazyk
- Umožňuje nekauzální popis kombinovaných modelů
- Symbolické úpravy rovnic
- Numerické řešení
- Knihovny modelů
- Organizace: ”Modelica association” – definuje jazyk

Kauzální a nekauzální modelování

- Nekauzální modelování
  - proměnné
  - rovnice = omezení možných hodnot proměnných
  - modely komunikují přes některé rovnice
  - Příklad – rovnice:
    $$u - Ri = 0$$
    Nedefinuje zda průchodem proudu vzniká napětí, nebo přiložené napětí způsobuje proud.
- Kauzální modelování
  - vstupy, výstupy
  - proměnné, stavové proměnné
  - relace mezi vstupy a proměnnými, derivacemi proměnných, výstupy
  - Příklad – blokové schema: `i := u/R`
    Máme hodnotu napětí `u` a parametru `R` a počítáme výstup – proud `i`.

Kauzální a nekauzální modelování – příklad





Převod rovnic na kauzální

Tarjanův algoritmus pro dif.-alg. rovnice (DAE):

- očíslovat rovnice (libovolně)
- očíslovat neznámé (libovolně) bez stavových proměnných
- vytvořit graf popisující stukturu rovnic - černé hrany spojují rovnice a v nich obsažené neznámé - viz obrázek
- v cyklu dokud je co dělat:
  1. Pro všechny ještě nezpracované rovnice, které mají právě jednu černou hranu, přebarvíme tuto hranu na červenou a všechny ostatní hrany vycházející z připojené neznámé obarvíme modře. Právě zpracované rovnici přidělíme nejnižší volné pořadové číslo počínaje 1.
  2. Pro všechny neznámé proměnné, které mají právě jednu černou hranu, přebarvíme tuto hranu na červenou a všechny hrany vycházející z připojené rovnice obarvíme modře. Připojené rovnici přidělíme nejvyšší volné pořadové číslo počínaje n (n je počet rovnic).

Převod rovnic na kauzální – pokračování

- uspořádáme rovnice podle pořadových čísel
- převedeme rovnice je tak, aby červenou hranou připojené neznámé byly na levé straně

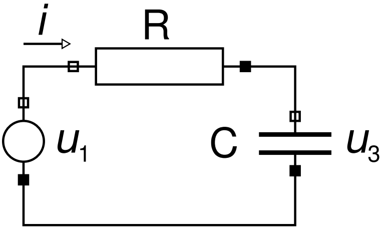
Výsledná soustava je kauzální – všechny neznámé lze postupně vypočítat, symbol $=$ odpovídá operaci přiřazení.

Poznámky:

- Časová složitost algoritmu je $O(N)$, N je počet rovnic.
- Problém, když narazí na rychlé smyčky (řešení: ”tearing”)
- Pantelides algoritmus (”output set assignment”)
- Strukturální singularity a index problému:

  $Index0$ DAE bez alg. smyček a bez strukt. singularit
  $Index1$ DAE obsahuje algebraické smyčky, ne singularity
  $Index > 1$ DAE obsahuje strukturální singularity (”higher index”)
  Pantelides algoritmus snižuje index o 1.

Převod rovnic na kauzální – příklad

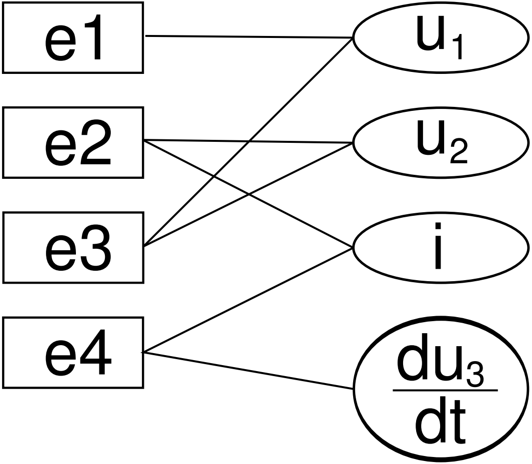


Rovnice popisující elektrický obvod na obrázku:

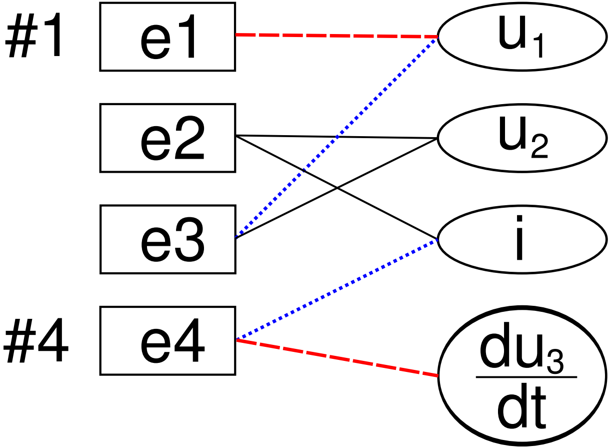
1. $u_1 = f(t)$
2. $u_2 = Ri$
3. $u_1 = u_2 + u_3$
4. $i = C\frac{du_3}{dt}$

Nekauzální popis, symbol $=$ znamená rovnost (ne přiřazení)

Tarjanův algoritmus – postup 1

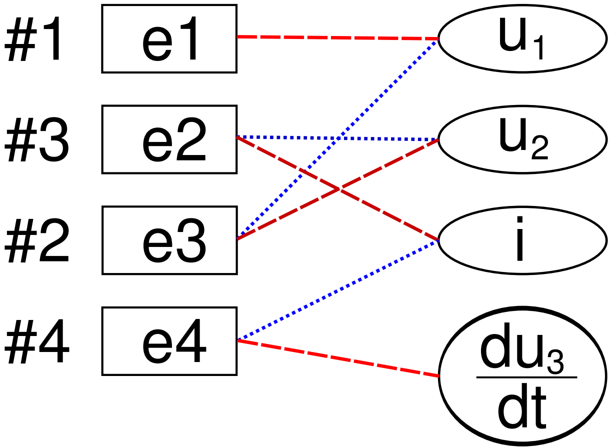


Tarjanův algoritmus – postup 2



(chyba)

Tarjanův algoritmus – postup 3



Tarjanův algoritmus – výsledek a poznámky

1. $u_1 = f(t)$
2. $u_2 = u_1 - u_3$
3. $i = \frac{u_2}{R}$
4. $\frac{du_3}{dt} = \frac{i}{C}$

V grafu platí:

- Kauzální rovnice = 1 červená hrana
- Nekauzální = pouze černé nebo modré hrany
- Známá proměnná = 1 červená hrana
- Neznámá = pouze černé nebo modré hrany
- Nikdy se nemůže vyskytnout více než 1 červená hrana u 1 uzlu

Základní koncepty jazyka Modelica

- Třídy: `class` `record`, `function`, `package`, `connector`, `model`, `block`
- Datové typy: `boolean`, enum, `integer`, `real`, string
- Strukturované: záznam, pole
- Konstruktory:

```
Complex(re=0,im=1)
{ 1, 2, 3 }
{ { 1, 2 }, { 3, 4 } }
zeros(2,3)
fill(3.14, 2, 3, 4)
```

Příklad modelu – kruhový test

```
model ctest "kruhovy test"
  Real x(start=1);
  Real y;
 equation
  der(x) = y;
  der(y) = -x;
end ctest;
```

Složené modely

- Konektory: třída `connector`,
  popis konektoru neobsahuje rovnice (ale používají je propojovací rovnice)
- Proměnné:
  - spojité (`real`),
  - diskrétní (boolean, integer, enumeration,string)
- Rovnice:
  - spojité,
  - propojovací,
  - ”instant”
- Bloky: speciální modely s konektory typu vstup a výstup

Konektory, propojení modelů

- Proměnné typu tok (`flow`)
- Proměnné typu potenciál (bez označení)
- Vstup/výstupní specifikace (`input`, `output`)

```
connector Pin     "elektrický kontakt"
  Real v;             // napětí = potenciál
  flow Real i;        // proud  = tok
end Pin;

block B
  input Real x;       // vstupní proměnná
  output Real y;      // výstupní proměnná
  ...
end B;
```

Proměnné

- Diskrétní: všechny typu `real` je nutné explicitně označit `discrete`
- Spojité: jen typu `real`

```
model M
  Integer i;                 // implicitně diskrétní
  discrete Integer j;        // explicitně
  Real x[20];                // spojité
  discrete Real y[5,3];      // diskrétní
end M;
```

Popis chování (constraints)

- ”instant”: aktivní jen při stavových událostech, popisují změny stavových (a diskrétních) proměnných vyžadují zápis ve tvaru `proměnná = výraz`
- Spojité (”Continuous”): obecný popis vztahů mezi proměnnými
- Propojovací (”Connection”): popis propojení modelů = speciální rovnice
- Rovnice: vztahy, podmínky, ... (nezáleží na pořadí)
- Algoritmy: imperativní – posloupnosti příkazů, přiřazení, cykly, ...

Spojité chování — rovnice

Příklad: Diferenciální a algebraické rovnice

```
equation
  if (time>5) then der(x)=x; else der(x)=-x; end if;
  der(y) =  if time>10 then y else -y;
  a = 5 * b;
  f(a) = 0;
  ...
```

Spojité chování — algoritmy

Příklad: Algoritmy

```
algorithm
  if time>5 then der(x):=x; else der(x):=-x; end if;
  der(y) :=  if time>10 then y else -y;
  if a>b then tmp:=a; a:=b; b:=tmp; end if;
  while a<b loop
      ...
  end while;
```

Deklarační rovnice

Model:

```
model M
  parametr Real a := 1;
  Real x = y + a + 1
end M;
```

je ekvivalentní modelu:

```
model M
  parametr Real a;
  Real x;
algorithm
  a := 1;
equation
  x = y + a + 1
end M;
```

Příklad: blok

```
block Integrator
    input  Real u;
    output Real y;
  protected
    Real x;      // zapouzdření
  equation
    der(x) = u;
    y = x;
end Integrator;
```

Příklad: funkce

```
function f
    input Real x;
    output Real y;
  algorithm
    y = if abs(x) < Modelica.Constants.eps then 1
        else Modelica.Math.sin(x)/x;
end f;
```

Příklady: package

```
package Library
    constant Real k = 0.1;
    type X = Real(min=0);

    model A
      ...
    end A;

    model B
      ...
    end B;

end Library;
```

Použití: `Library.k` nebo `import`

Dědičnost

Lze dědit a modifikovat chování a rozhraní
Příklad:

```
block C "odvozeny blok z B"
  import Library.Types;     // zpřístupní typy
  extends Library.B;        // dědí z B
  ...
 initial equation
   y = y_0;                 // počáteční podmínky
 equation
   der(y) = x;              // rovnice
end C;
```

Stavové podmínky a události, ”instant equations”

- Stavové podmínky a události

```
when aktivační podmínka then
  rovnice nebo algoritmy
else when aktivační podmínka then
  ...
end when;
```

- Když se podmínka stane pravdivou, aktivuje se odpovídající rovnice/algoritmus
- Požadavky na rovnice v sekci when:
  `y = ....` y se nemění mezi událostmi

Stavové podmínky – příklad

```
when time>5 then
  reinit(x,0);         // x:=0
  a = pre(a) + 1;      // pre == stará hodnota
end when;

when a>10 then
  z := z+y;
  c = 2*pre(c) + pre(d);
end when;
```

Poznámka: Závislosti mezi stavovými událostmi

Propojení modelů

- Propojovací rovnice propojují konektory stejného typu
  Propojení:

  ```
  connect(model1.p, model2.n);
  connect(model1.p, model3.n);
  ```

  je ekvivalentní

  ```
  connect(model2.n, model1.p);
  connect(model2.n, model3.n);
  ```

- Propojení definuje graf
- V každém propojení konektorů platí:
  - potenciálové proměnné jsou si rovny
  - suma "flow" proměnných je nulová

Propojení modelů

Je možné použít cyklus:

```
...
  parameter Integer NR=10 "pocet rezistoru";
  Modelica.Electrical.Analog.Basic.Resistor R[NR];
equation
  for i in 1:NR-1 loop
    connect(R[i].p, R[i+1].n); // 9 propojovacích rovnic
  end for;
```

Možnosti cyklu for:

```
for i in 1:10 loop              // i:  1,2,3,...,10
for r in 1.0 : 1.5 : 5.5 loop // r:  1.0, 2.5, 4.0, 5.5
for i in {1,3,6,7} loop         // i:  1, 3, 6, 7
```

Příklady různých forem popisu

Ekvivalentní modely popsané rovnicemi, algoritmy a kombinací

```
model R1
  Pin p,n;
  Real v,i;
  parameter Real R;
equation
  p.v - n.v = v;
  p.i + n.i = 0;
  i = p.i;
  v = R * i;
end R1;
```

```
model R2
  Pin p,n;
  Real v,i;
  parameter Real R;
algorithm
  i := p.i;
  v := R * i;
  n.v := p.v + v;
  n.i := -p.i;
end R1;
```

```
model R3
  Pin p,n;
  Real v,i;
  parameter Real R;
equation
  p.v - n.v = v;
  p.i + n.i = 0;
algorithm
  i := p.i;
  v := R * i;
end R1;
```

Příklad 2

Model: RC článek

```
model Circuit
  Resistor  R1(R=10);
  Capacitor C(C=0.01);
  SineVoltage AC(freqHz=50,V=10,startTime=0);
  Ground    G;
 equation
  connect (AC.p, R1.p);
  connect (R1.n, C.p);
  connect (C.n, AC.n);
  connect (AC.n, G.p);
end Circuit;
```

Poznámka: +Anotace pro grafiku

Poznámky

- kontrola struktury modelu
- single assignment rule (počet rovnic = počet neznámých)
- Model s pouze spojitým popisem definuje systém diferenciálně algebraických rovnic $f(y', y, t) = 0$
- převod rovnic na podobu řešitelnou numerickými algoritmy (DASSL)
- proměnné, které nejsou stavové nazýváme ”algebraické”

Poznámky

- Automatickou detekci změn stavových podmínek a provádění stavových událostí lze zakázat:

```
x = smooth(1, if y>0 then y^2 else -y^2);
x = noEvent(if y>=0 then sqrt(y) else 0);
```

- Modelica nesynchronizuje události = musí se explicitně naprogramovat

Knihovny modelů

- Modelica standard library (open source)
  - Bloky
  - Konstanty
  - Elektrické obvody
  - Matematické funkce
  - Mechanika
  - Jednotky SI
  - Tepelné modely
  - ...
- součást Dymoly
- + komerční knihovny

Modelica – shrnutí

- Jazyk: Modelica standard (verze 3.6/2023)
- Knihovny: Modelica Standard Library (v 4.0.0/2020)
- Existuje několik implementací:
  - Dymola
  - OpenModelica
  - MathModelica
  - JModelica.org
  - ...
- Stále se zdokonaluje (1.0/1997 — 3.5/2021)

Závěr

- Cílem bylo uvedení do různých oblastí modelování a simulace.
- Simulace je významná pro návrh systémů i jejich testování.
- Přehled algoritmů a metod může být inspirací i pro další oblasti použití.
- Přehled témat pro zkoušku — viz WWW