Samokontrola a samodiagnostika na systémové úrovni

Viktor Mashkov, Jiří Fišer

Ukrainian Academic Press
Lviv

2010

Návrh obálky: Pavel Simr
Jazyková korektura: Petra Jelínková

Recenzent: doc. Ing. Jaromír Kukal, Ph.D.

ISBN:

А судьи кто?

«Горе от ума» (1824) А. С. Грибоедов (1795—1829)

Kdo jsou ti soudci?

Hoře z rozumu, A. S. Gribojedov (překl. Bohumil Franěk, 1979)

Obsah

Úvod

Tato kniha je věnována problematice samokontroly a samodiagnostiky složitých systémů na *systémové úrovni*. Systémová úroveň odpovídá takové úrovni abstrakce nebo formalizace, při níž je systém uvažován jako množina komponentů (modulů). Tyto moduly spolu komunikují a především spolupracují, aby splnily funkcionální a další požadavky kladené na systém. Abstraktní model systému nikterak nezohledňuje implementaci jednotlivých modulů. To znamená, že model může být použit pro popis jakéhokoliv druhu systému, a to jak systémů technických tak i sociálních.

Jedním z hlavních problémů, se kterým se setkáváme v oblasti kontroly a diagnostiky systémů, je rozhodování o tom, kdo bude provádět jednotlivé kontroly modulů systému a konečné rozhodování, resp. poskytování výsledků kontroly a diagnostiky. Obecně tyto funkce provádí speciální modul, který se většinou nezúčastní provádění systémových funkcí a je dedikován pouze ke kontrolním účelům. Tento postup při kontrole a diagnostice systému přináší velmi vysoké požadavky na spolehlivost takového speciálního modulu. V reálných systémech však není vždy možno zajistit vysokou spolehlivost a důvěryhodnost speciálního kontrolního modulu, a proto jsou otázky přerozdělení (distribuce) funkcí kontrolního modulu mezi moduly systému velmi aktuální a přitahují velkou pozornost vývojářů.

Kniha shrnuje výsledky výzkumné práce autorů a dalších vývojářů při řešení těchto otázek. Kniha je napsána pro široké spektrum čtenářů. Předpokládá se, že čtenář má základní znalosti v oblasti programovaní a teorie pravděpodobnosti a je alespoň částečně seznámen s matematickými základy teorie spolehlivosti. Matematické vztahy a výpočty uvedené v knize jsou zjednodušeny a zkráceny, aby je mohli využívat i studenti technických směrů. Složité detaily jsou vynechány, což činí knihu dostupnou pro širší okruh čtenářů.

Předpokládáme, že kniha pomůže studentům lépe pochopit obecná specifika kontroly, diagnostiky a dependability složitých systémů, což umožní studentům korektněji uvažovat a zohledňovat problematiku kontroly a diagnostiky při návrhu vlastních systémů v rámci seminárních a závěrečných prací. Vývojáři složitějších systémů mohou v knize nalézt odpovědi na otázky jak decentralizovat kontrolní a rozhodčí funkce pro zlepšení celkové dependability systémů.

Reference na zdroje uvedené v textu umožňují čtenáři detailnější studium problematiky a rozšíření znalostí v rámci daného tématu.

Kniha sestává ze čtyř kapitol:

Kapitola 1 „*Samodiagnostika*“ uvádí základní koncepty, pojmy a definice, které se týkají samodiagnostiky a diagnostického modelu systému. Zavádí se *diagnostický graf systému*. Důležitou částí je klasifikace algoritmů samodiagnostiky a jejich analýza. Každá třída diagnostických algoritmů je vysvětlena pomocí ukázkových příkladů.

Kapitola 2 „*Organizace samokontroly a samodiagnostiky*“ se zabývá různými organizacemi provedení samodiagnostiky ve složitém systému. Zavádí se koncepce diagnostického jádra a uvažují různé druhy diagnostických jader. Hlavní důraz je kladen na mechanismus *putujícího diagnostického jádra*.

Kapitola 3 „*Model spolehlivosti systému*“ rozebírá model spolehlivosti systému pro případ použití samodiagnostiky založené na putujícím diagnostickém jádru. Kapitola je organizována tak, že se model spolehlivosti systému postupně zpřesňuje od modelu zjednodušeného do modelu detailního. Detailní model umožňuje ohodnotit vliv samokontroly a samodiagnostiky na spolehlivost systému. Toto ohodnocení a jeho výsledky jsou uvedeny v závěru kapitoly.

Kapitola 4 „*Samokontrola a samodiagnostika v kontextu spolehlivosti a dependability*“ uvádí přehled hlavních konceptů dependability, popisuje jednotlivé části, ze kterých se dependabilita skládá. Hlavním účelem je ukázat roli a místo samokontroly a samodiagnostiky při zajištění dependability složitého systému. Možnosti využití samokontroly a samodiagnostiky ve výpočetních systémech jsou vysvětleny na konkrétních příkladech.

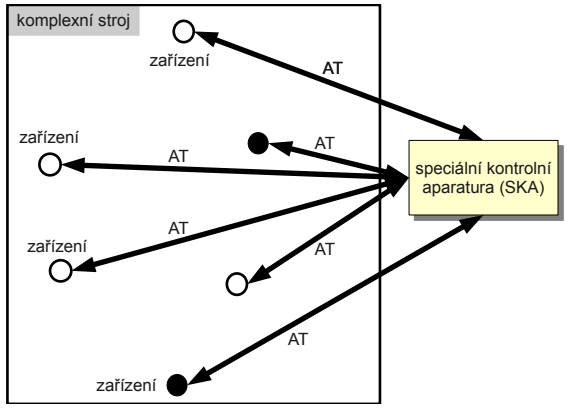
1. Samodiagnostika

1.1. Kontrola složitých systémů

Současný svět je plný složitých technických programovatelných zařízení, které řídí neméně komplexní stroje jako jsou letadla, vlaky a v poslední době i osobní automobily. Tato technická zařízení tvoří hardware a programové vybavení, pro něž je klíčová bezchybná činnost. Důležitou roli proto hraje **kontrola** těchto zařízení.

Technická diagnostika uvažuje dvě fáze kontroly zařízení: před průběhem vlastní činnosti zařízení (*předběžná*) a v jeho průběhu (*průběžná*).

Předběžná kontrola může být prováděna pomocí *speciální kontrolní aparatur*y (SKA), která detailně prověří technický stav daného zařízení (viz 1.1). To je typické především pro předstartovní diagnostiku letadel (testovací aparatura je v tomto případě rozsáhlé externí zařízení, jehož instalace přímo v letounu by byla neefektivní).



Obrázek 1.1.: Speciální kontrolní aparatura

Samotná předběžná kontrola může zahrnovat vícenásobnou výměnu dat mezi SKA a zařízením a jejich následné zpracování kontrolním algoritmem v SKA. Tento kontrolní algoritmus obvykle provádí jednoduché porovnání výstupních dat ze zařízení s daty, která jsou považována za správná (etalonní).

Tato celková a mnohdy komplexní kontrola modulu, zahrnující jak výměnu dat tak provádění kontrolního algoritmu, může být na vyšší úrovni abstrakce interpretována jako jediná **atomická akce resp. kontrola** (AT = *atomic test* nebo *elementary check*). Na vyšší, tj. systémové úrovni abstrakce, jsou detaily kontroly irelevantní, v kontextu této abstrakce tudíž vystačíme s pohledem, že jeden systémový modul kontroluje druhý.

SKA je obvykle považována za bezchybnou, tj. můžeme plně důvěřovat výsledkům kontroly. V praxi je toho dosahováno různými způsoby, přičemž jedním z nejdůležitějších kritérií je úplnost kontroly. Například při kontrole programu s mnoha větvemi toku řízení (např. pro různá vstupní data) je nutno projít všemi větvemi. Pokud by byla některá větev vynechána, nebude kontrola úplná.

Hlavní výhodou SKA je proto vysoká důvěryhodnost výsledků. Naopak k nevýhodám předběžné kontroly za použití SKA patří:

- vysoké (finanční) náklady a náročné použití
- časové a režijní náklady
- nutnost zaškolení personálu (kontrola běžně vyžaduje účast operátora)
- velké prostorové nároky běžných kontrolních mechanismů.

Nyní se zaměříme na zajímavější druhou fázi, to jest **průběžnou kontrolu**. Kontrola je v tomto případě prováděna souběžně s hlavní činností zařízení. V tomto případě nelze běžně použít speciální kontrolní aparaturu (projevují se všechny výše uvedené nevýhody), její náhrada je však komplikovaná.

Jedním z řešení je použití specializovaného modulu umístěného externě, tj. vně kontrolovaného zařízení. Oproti SKA je specializovaný modul výrazně jednodušší a jeho funkce jsou omezené. Tento modul provádí kontrolu a sběr dat ve vymezených časových intervalech.

Hlavní nevýhodou použití kontrolního modulu je obtížné zajištění vysoké spolehlivosti a důvěryhodnosti výsledků kontroly. Navíc v průběhu provádění hlavní činnosti kontrolovaná zařízení navzájem komunikují, což ovlivňuje a ztěžuje jeho kontrolu.

1.2. Samodiagnostika s vnějším pozorovatelem

Možným řešením výše uvedených problémů je **samodiagnostika,** tj. vzájemná kontrola a diagnostika jednotlivých účelových zařízení komplexního stroje. V tomto případě neexistuje žádné dedikované kontrolní zařízení, tj. všechna zařízení vykonávají jak běžnou tak kontrolní činnost.

Atomická kontrola může i v tomto případě zahrnovat:

1. jednoduchou kontrolu přijatých běžných dat (např. kontrolní součty)
2. složitější kontrolu přijatých dat (např. testování etalonu)
3. odesílání speciálních kontrolních dat a přijetí a zpracování odezvy.

Na systémové úrovni abstrakce odpovídá každému zařízení **modul**. Celkový model systému je tak representován grafem, v němž uzly representují moduly tj. jednotlivá dílčí zařízení a hrany atomické kontroly. Tento graf se nazývá **diagnostickým grafem** systému (zkráceně **DG**)

Každé technické zařízení (tj. modul) může být buď ve stavu, kdy poskytuje správná výstupní data (= **bezchybný modul**) resp. kdy jsou jím produkovaná data nesprávná (= modul selhal, **chybný modu**l). Bohužel může kontrolující zařízení chybně vyhodnotit data přijatá ze zařízení kontrolovaného a to v obou směrech (správná označit za chybná resp, chybná za správná) a tak zařízení nesprávně ohodnotit.

Každý modul tak má svá vlastní hodnocení modelů, které zkontroloval a tato hodnocení nemusí být konzistentní (tj. nemusí existovat konsenzus v hodnocení jednotlivých modulů). Navíc pravděpodobnost nekonzistence může být relativně velká.

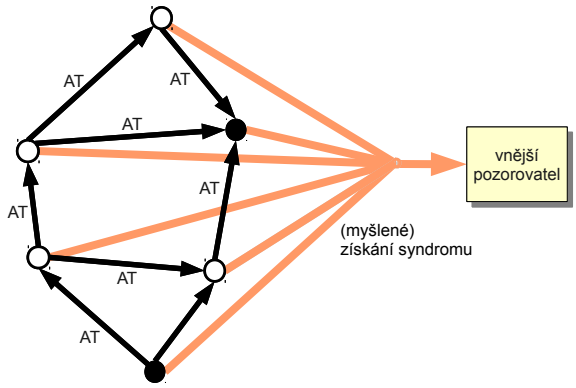
Necháme prozatím stranou problémem nekonzistence v hodnoceních jednotlivých modulů, ale zamyslíme se jak je možno i přes případnou nekonzistentnost využít výsledků jednotlivých kontrol.

Na začátku si pro jednoduchost představíme, že existuje **abstraktní vnější pozorovatel**, který dostane výsledky všech dílčích atomických kontrol (viz obrázek 1.2 na následující straně). Předpokládejme pro jednoduchost, že přitom nedojde k žádnému chybnému přenosu nebo chybné interpretaci obdržených dat. Tento pozorovatel ve skutečnosti neexistuje, neboť kontrola musí být plně autonomní. Jeho zavedení však zjednoduší počáteční model systému.

Lze si tak například položit otázku, zda je tento abstraktní pozorovatel schopen na základě výsledků atomických kontrol určit, které moduly jsou bezchybné a které naopak selhaly. Tento problém je již součástí systémové diagnostiky.

Diagnostika na **systémové úrovni** spočívá v odhalení všech chybných (=selhávajících) modulů. Naopak je nutno zdůraznit, že na této úrovni se neuvažuje konkrétní příčina selhání modulu (tj. co se stalo uvnitř zařízení).

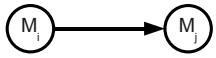
V uvedeném zjednodušeném modelu je jediným vstupem diagnostiky (prováděné abstraktním vnějším pozorovatelem) množina výsledků atomických kontrol. Tato množina je označována jako **syndrom**. Výstupem je seznam chybných modulů, resp. komplementární seznam modulů chybných.



Obrázek 1.2.: Vnější pozorovatel

Výstup, tj. výsledek diagnostiky i jeho důvěryhodnost, závisí na několika předpokladech souvisejících s atomickými kontrolami. Nejdůležitějším je předpoklad o výsledcích kontrol prováděných jak bezchybnými tak selhávajícími moduly.

Uvažujme například atomickou kontrolu modulu M_j provedenou modulem M_i (viz obrázek 1.3). Předpokládejme, že výsledek kontroly bezchybného modulu bezchybným modulem je roven vždy hodnotě **0**. Obvyklá notace má tvar $r_{ij} = 0$.



Obrázek 1.3.: Atomická kontrola

Složitější je situace v případě kontroly selhávajícího modulu (zde tedy např. M_j) modulem bezchybným (zde např. M_i). Většinou se předpokládá (viz např. [1]), že r_{ij} je v tomto případě rovno **1**, tj. bezchybný modul vždy odhalí chybu v modulu selhávajícím.

Nakonec uvážíme situaci, kdy kontrolu provádí selhávající modul. V tomto případě lze předpokládat, že výsledek bude náhodný, tj. může nabývat jak hodnoty 0 tak 1. V nejjednodušším případě budou tyto hodnoty nabývat se stejnou pravděpodobností (rovnou 0,5). Existují však i další modely, například Barsi, Grandoni a Maesstrini [2] nabízejí v tomto případě předpoklad, že výsledek této kontroly je vždy roven hodnotě 1 (tj. kontrolovaný modul bude vždy označen za chybný). V praxi navíc můžeme mít i zpřesňující informace o chování selhávajícího modelu, včetně pravděpodobnosti produkování různých výsledků atomických kontrol, tj. např. zpřesněnou informaci o produkování zavádějících výsledků.

Všechny výše uvedené předpoklady navíc uvažují, že spojení mezi moduly jsou bezchybná (tj. při přenosu informací nedochází k jejich ztrátě). V opačném případě by musely být předpoklady přehodnoceny.

Zde však budeme vycházet pouze z následujících relativně jednoduchých předpokladů (podle Preparata, [1]).

DEFINICE 1.1:

Výsledek atomické kontroly modulu M_j modulem M_i je definován takto:

$$r_{ij} = \begin{cases} 0 & \text{jsou-li oba moduly } M_i \text{ i } M_j \text{ bezchybné} \\ 1 & \text{je-li } M_i \text{ bezchybný a } M_j \text{ selhávající} \\ X(0,1) & \text{je-li } M_i \text{ selhávající} \end{cases} \tag{1.1}$$

□

Výsledek, který poskytuje vnější pozorovatel, je ovlivněn i strukturou atomických kontrol, které tvoří *diagnostický graf*.

DEFINICE 1.2:

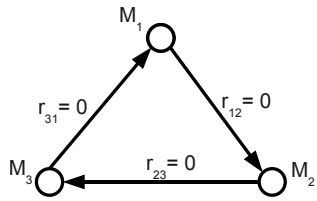
Syndrom R je uspořádaná množina výsledků jednotlivých atomických kontrol, tj.

$$R = \{r_{ij}\} \tag{1.2}$$

Jednotlivé výsledky r_{ij} se označují jako prvky syndromu.

□

Jednotlivé prvky syndromu mohou být v diagnostickém grafu representovány jako ohodnocení hran, kde se hodnota rovná výsledku atomické kontroly. Viz obr. 1.4, kde ohodnocení pro přehlednost obsahuje i označení výsledků.



Obrázek 1.4.: Diagnostický graf systému

Tento diagnostický graf presentuje syndrom v názorné formě. Usnadňuje provedení diagnostické analýzy (z pozice vnějšího pozorovatele). Například syndrom na

obrázku 1.4 umožňuje učinit závěr, že všechny tři moduly jsou bezchybné. Samozřejmě jen tehdy, pokud platí zvolený model ohodnocení atomických kontrol 1.1.

1.3. Hodnocení diagnostiky podle diagnostického grafu

Pokud je k dispozici diagnostický graf, je možno položit si otázku, zda lze systém diagnostikovat, tj. určit chybné moduly, a to bez ohledu na obdržený syndrom.

Lze dokázat, že úspěšnost diagnostiky závisí na počtu chybných, tj. selhávajících modulů. Tento počet se označuje jako t. Pro některé hodnoty t lze vždy vytvořit diagnostický graf, který bude s úplnou jistotou zaručovat správnou diagnostiku bez ohledu na získaný syndrom.

Pro grafy, u nichž je zaručeno diagnostikování t chybných modulů, tzv. **t-diagnostikovatelnost**, platí následující *nutná* podmínka:

V t-diagnostikovatelném grafu musí být modul kontrolován nejméně t dalšími moduly, přičemž v grafu nejsou vícenásobné hrany (= atomické kontroly).

Pokud však počet chybných modulů t překročí jisté t_{max}, pak již není možné takový DG zkonstruovat. Preparata ve své práci [1] dokázal, že pro toto t_{max} platí:

$$t_{max} = \left\lfloor \frac{N-1}{2} \right\rfloor, \tag{1.3}$$

kde N je počet uzlů resp. modulů.

t-diagnostikovatelných grafů (pro $t \leq t_{max}$) však může existovat i více. Zajímavé jsou však především ty s malým či dokonce nejmenším počtem atomických kontrol (větší počet kontrol prodražuje a komplikuje diagnostiku).

DEFINICE 1.3:

Diagnostický graf je **t-optimální**, pokud obsahuje minimální počet hran, které stačí pro zajištění určité hodnoty t. Pro hodnotu $t = t_{max}$ je možno graf stručně označit jako **optimální**.

□

Počet hran *t-optimálního grafu* lze snadno vypočítat podle následujícího vztahu:

$$l = t \cdot N, \tag{1.4}$$

jenž lze v případě *optimálního diagnostického grafu* dále upravit na:

$$l = t_{max} \cdot N = \left\lfloor \frac{N-1}{2} \right\rfloor \cdot N, \tag{1.5}$$

kde

N = počet uzlů (modulů)

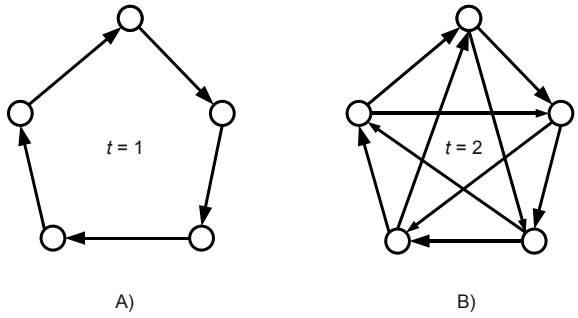
t_{max} = maximální hodnota parametru t

l = počet hran optimálního DG.

Libovolný graf, který obsahuje více než l hran, je podle této definice považován za nadbytečný.

Vztah lze použít i v opačném směru a pro danou hodnotu parametru t vytvářet instance struktur DG, které zajišťují určité diagnostické vlastnosti, například schopnost odhalit určitý počet chybných modulů.

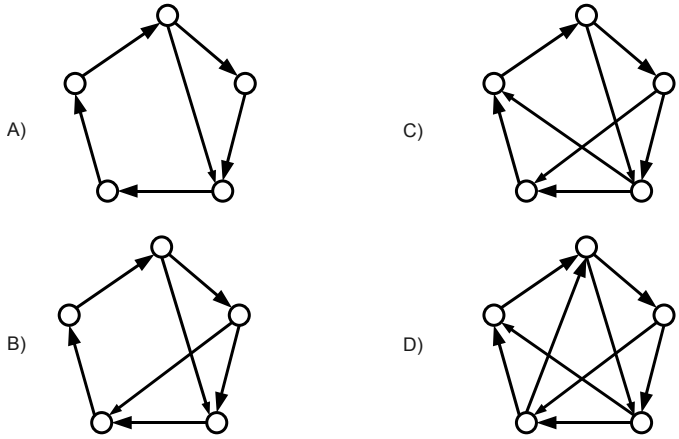
Například diagnostický graf na obr. 1.5(A) je t-optimální pro $t = 1$, neboť zajišťuje detekci pouze jednoho chybného modulu a počet hran je pro dané t minimální. DG na obrázku 1.5(B) má $t = 2$ a zajišťuje tak již detekci dvou chybných modulů (a je navíc optimální, neboť $t = t_{max}$ a počet hran je minimální).



Obrázek 1.5.: Optimální diagnostické grafy

Všechny diagnostické grafy zobrazené na obr. 1.6 na následující straně zajišťují detekci stejného počtu chybných modulů ($t = 1$), ale mají různý počet hran. Metoda hodnocení DG navržená Preparatou neumožňuje porovnat tyto diagnostické struktury a definovat přesněji jejich diagnostické vlastnosti. Zohledňuje se pouze dosažená hodnota t resp. t-optimálnost.

Kromě parametru t však existují také další *kritéria* pro hodnocení diagnostických vlastností grafu, které umožňují porovnání a ohodnocení grafů na obrázku 1.6. Například každému DG může být přiřazena hodnota pravděpodobnosti, která bude



Obrázek 1.6.: Diagnostické grafy *s t*=1

odrážet diagnostické schopnosti grafu. Konkrétně je to pravděpodobnost, že syndrom odpovídající určitému DG umožní správně diagnostikovat stav všech modulů.

Tuto pravděpodobnost si vysvětlíme pomocí jednoduchého příkladu.

Nechť má například DG strukturu zobrazenou na obrázku 1.7 na následující straně. Z obrázku je zřejmé, že když bude bezchybný jen modul M_1, pak obdržený syndrom umožňuje správně diagnostikovat stavy ostatních modulů (tj, že jsou oba chybné). Pravděpodobnost této situace můžeme spočítat pomocí obecného vztahu:

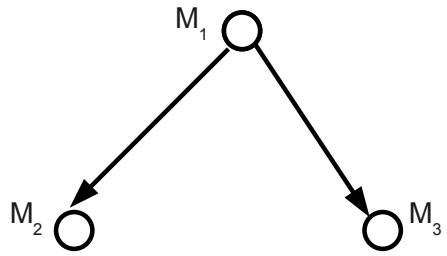
$$P(A_K) = C_K^N \cdot (1-P_M)^K \cdot P_M^{N-K} \cdot \frac{C_K}{C_K^N} = (1-P_M)^K \cdot P_M^{N-K} \cdot C_K \tag{1.6}$$

kde

- P_M = pravděpodobnost, že modul je v chybném stavu (selže). Pravděpodobnost je u všech modulů stejná.
- K = počet bezchybných modulů (zde 1)
- C_K = počet možností výběru podgrafu tvořeného K uzly, ze kterého jsou všechny ostatní uzly přímo dosažitelné
- C_K^N = počet kombinací k-té třídy z n prvků $\binom{n}{k}$

V případě jednoprvkové množiny uzlů ($K = 1$) je pravděpodobnost rovna:

$$P(A_1) = (1-P_M) \cdot P_M^2 \cdot C_1$$



Obrázek 1.7.: Ukázkový DG pro vysvětlení pravděpodobnosti P_{SD}

Pro DG na obrázku 1.7 se číslo C_1 rovná 1, protože existuje pouze jeden výběr jednoprvkové množiny uzlů, z nichž jsou ostatní uzly přímo dosažitelné (množina $\{M_1\}$).

Správnou diagnostiku získáme také v případě, že jsou správné pouze dva moduly ze tří, a to buď $\{M_1, M_2\}$nebo $\{M_1, M_3\}$. Pravděpodobnost této události je podle vztahu 1.6 rovna:

$$P(A_2) = (1 - P_M)^2 \cdot P_M \cdot C_2$$

Pro uvažovaný diagnostický graf je C_2 rovno 2, neboť existují pouze dva podgrafy s dvěma uzly, z nichž jsou dosažitelné všechny ostatní uzly ($\{M_1, M_2\}$, $\{M_1, M_3\}$).

To však ještě není vše, neboť správný výsledek přirozeně dostaneme i v případě, že jsou správné všechny tři moduly. Pravděpodobnost takovéto události je:

$$P(A_2) = (1 - P_M)^3 \cdot P_M^0 \cdot C_3 = (1 - P_M)^3$$

C_3 je v tomto případě vždy rovno 1, a to bez ohledu na strukturu atomických kontrol.

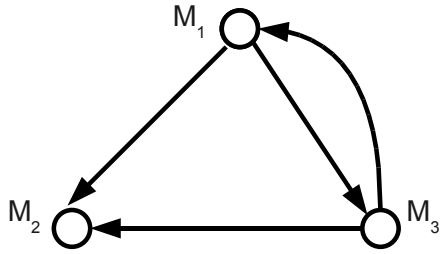
Výsledek diagnostiky systému je správný, pokud nastane *alespoň jedna* ze situací A_K, $k = 1, \ldots, n$. Z toho vyplývá, že pravděpodobnost P_{SD} získání správného výsledku diagnostiky systému na základě syndromu, který odpovídá DG, je rovna:

$$P_{SD} = \sum_{K=1}^{N} P(A_K) = \sum_{K=1}^{N} (1 - P_M)^K \cdot P_M^{N-K} \cdot C_K \tag{1.7}$$

Nyní již můžeme vypočítat pravděpodobnost P_{SD} pro diagnostický graf na obr. 1.7.

Například pro $P_M = 0.1$:

$$P_{SD} = (1-P_M)\cdot P_M^2 \cdot 1 + (1-P_M)^2 \cdot P_M \cdot 2 + (1-P_M)^3 = 0.9$$



Obrázek 1.8.: Rozšířený ukázkový DG (přidány dvě hrany)

Nyní uvážíme nový mírně rozšířený diagnostický graf z obrázku 1.8. Zde byly přidány dvě hrany. Modul M_3 nyní kontroluje ostatní moduly (M_1a M_2).

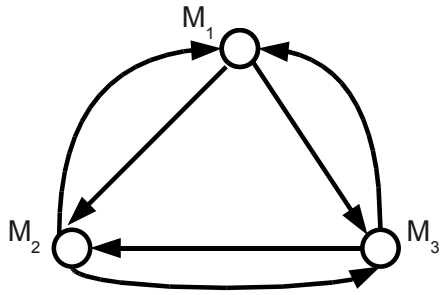
Pro tento DG se proto změní čísla C_1a C_2:

$C_1 = 2$ výběry : $\{M_1, M_2\}$

$C_2 = 3$ výběry: $\{M_1, M_2\}$a $\{M_1, M_3\}$ a $\{M_2, M_3\}$

Pravděpodobnost P_{SD} pro tento DG a shodnou hodnotu $P_M = 0.1$ je rovna $(1-P_M)\cdot P_M^2 \cdot 2 + (1-P_M)^2 \cdot P_M \cdot 3 + (1-P_M)^3 = 0.99$.

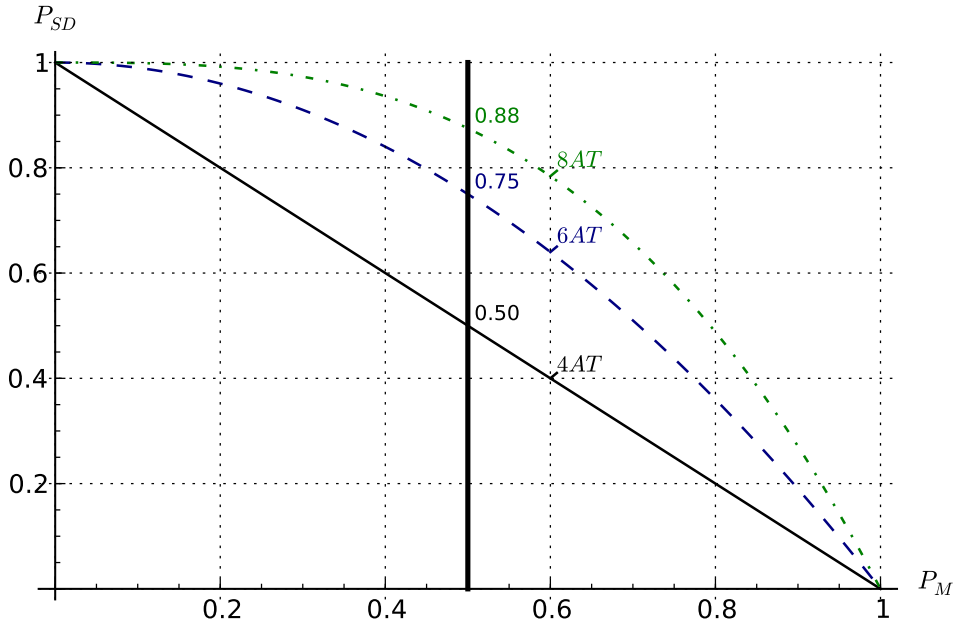
Nakonec uvážíme diagnostický graf z obrázku 1.9. Zde byly přidány další dvě atomické kontroly z modulu M_2. Číslo C_1 se tak zvýší na 3 (z každého uzlu lze přímo dosáhnout ostatní), číslo C_2 zůstává na hodnotě 3. Pravděpodobnost $P_{SD} = (1-P_M)\cdot P_M^2 \cdot 3 + (1-P_M)^2 \cdot P_M \cdot 3 + (1-P_M)^3$ se opět mírně zvýší a to na 0.993.



Obrázek 1.9.: Rozšířený ukázkový DG (přidány čtyři hrany)

Přidání hran zde tedy nevede ke zvýšení t, neboť to je rovno t_{max}. Zvýší se však pravděpodobnost získání správného výsledku diagnostiky. To lze ještě lépe vidět

z grafu závislosti P_{SD} na P_M na obrázku 1.10. S rostoucí chybovostí modulů P_M pravděpodobnost správné diagnostiky systému P_{SD} klesá, ale u DG s větším počtem atomických testů je pokles méně výrazný. Jednotlivé křivky odpovídají DG na obrázcích 1.7 (dole) 1.8 (uprostřed) a 1.9 (nahoře).



Obrázek 1.10.: Funkční závislost P_{SD} na P_M a počtu AT

Při výpočtu pravděpodobnosti správného výsledku diagnostiky P_{SD} hrají klíčovou roli čísla C_K, která odrážejí strukturu diagnostického grafu, a tudíž i strukturu atomických kontrol. Tato čísla se označují jako *čísla charakteristická.*

DEFINICE 1.4:

Charakteristické číslo $C_{K,}$ $k = 1,2,\ldots,n$ je *počet výběru K uzlů* (podgrafů) z diagnostického grafu, ze kterých jsou všechny *ostatní uzly přímo dosažitelné.*

□

U jednoduchých grafů lze charakteristická čísla zjistit snadno z nákresu diagnostického grafu. U komplexnějších diagnostických grafů je však nutno využít automatizovaného výpočtu nad modifikovanou maticí sousednosti.

Modifikovaná matice sousednosti je odvozena z běžné matice sousednosti nastavením hodnoty 1 u všech prvků na diagonále (tj. je zohledněn fakt, že uzel je dostupný ze sebe sama).

Algoritmus musí otestovat všechny kombinace výběrů, tj. jde o vyčerpávající prohledání, a vypočítat charakteristické číslo podle následujícího vztahu:

$$C_K = \frac{1}{k!} \underbrace{\sum_{i_k=1}^{n} \sum_{i_{k-1}=1}^{n} \cdots \sum_{i_1=1}^{n}}_{i_k \neq i_{k-1} \neq \cdots \neq i_1} \left[\prod_{j=1}^{n} (a_{i_k j} \vee a_{i_{k-1} j} \vee \ldots \vee a_{i_1 j}) \right] \tag{1.8}$$

Použití vztahu (1.8) si ukážeme na modifikované matici sousednosti grafu z obrázku 1.8 na straně 18.

$a_{11} = 1$	$a_{12} = 1$	$a_{13} = 1$
$a_{21} = 0$	$a_{22} = 1$	$a_{23} = 1$
$a_{31} = 1$	$a_{32} = 0$	$a_{33} = 1$

Při výpočtu charakteristického čísla C_i se vypočítává suma přes všechny řádka matice, přičemž pro každý řádek je počítán produkt přes sloupce.

$$C_i = \sum_{i=1}^{3} \left[\prod_{j=1}^{3} a_{ij} \right], \text{ kde j=číslo sloupce}$$

Hodnota produktu pro první řádek je rovna 1, neboť $a_{11} \cdot a_{12} \cdot a_{13} = 1$. Produkt u ostatních řádků je roven 0 ($a_{21} \cdot a_{22} \cdot a_{23} = 0$, $a_{31} \cdot a_{32} \cdot a_{33} = 0$). Suma pro C_1 je tedy rovna 1.

V případě čísla C_2 je nutné otestovat všechny uspořádané dvojice řádků, tj. variace bez opakování (1,2), (1,3), (2,1), (2,3), (3,1), (3,2).

$$C_2 = \frac{1}{2!} \sum_{i_2=1}^{3} \sum_{i_1=1}^{3} \left[\prod_{j=1}^{3} (a_{i_2 j} \vee a_{i_1 j}) \right]$$

Proměnné i_2 a $i_1$určují jednotlivé dvojice řádků (prochází se všechny kombinace dvojic). Poté se v jednotlivých sloupcích spočítá logický součet hodnot, a to vždy jen v řádcích i_1 a i_2, a z výsledných hodnot je vypočítán produkt.

Pro první dvojici (1,2) je například potřeba spočítat následující produkt:

$$\prod_{j=1}^{3} (a_{2j} \vee a_{1j}) = 1$$

Podobně se vypočítají i další dvojice.

Program pro výpočet charakteristického čísla C_n pro konkrétní n je relativně snadný (má $n+2$ cyklů). Obecný výpočet je složitější, neboť vyžaduje generování velkého počtu kombinací. Následující ukázka je v jazyce Haskell:

```
1  import Data.List (delete, transpose)
2
3  permutations :: Eq a => Int -> [a] -> [[a]]  -- permutations
4  permutations 0 _  = [[]]
5  permutations n xs = [x:xs | x<-xs, xs <- permutations (n - 1) (x `delete` xs)]
6
7  ladd :: Int -> Int -> Int -- logical or
8      -- (logical OR i.e. || is defined only for boolean values)
9  ladd 0 0 = 0
10 ladd _ _ = 1 --for all other = 1
11
12 getRows :: [Int] -> [a] -> [a] --get row with indices
13 getRows indices matrix = [ row | (i, row) <- (zip [0..] matrix),
14                                  i `elem` indices]
15
16 cnum :: Int -> [[Int]] -> Int --characteristic number
17 cnum n matrix =
18   (sum [ product ( map (foldr1 ladd) (transpose (getRows i matrix)))
19            | i <- permutations n [0..v]]) `div` f
20        where v = length matrix - 1
21              f = product [1..n] --factorial
```

Program definuje tři pomocné funkce. Funkce *combinations* produkuje všechny n-prvkové kombinace v podobě seznamu seznamů. Funkce *ladd* definuje logický součet nad celými čísly (resp. nad podmnožinou $\{0,1\}$). Funkce *getRow* umožňuje výběr řádků z matice (representované jako seznam seznamů). Vyjímané řádky jsou dány seznamem indexů (parametr *indices*).

Vlastní výpočet charakteristického čísla (funkce *cnum*) je již relativně přímočarý. Je zde počítána suma seznamu, který je získán aplikováním dílčího výrazu na všechny n-prvkové kombinace řádku. Je použita tzv. seznamová komprehenze, v níž je na proměnnou i, která postupně odkazuje na jednotlivé kombinace, aplikován výraz před svislítkem. V rámci výrazu se nejdříve provede výběr řádku podle aktuální kombinace indexů (*getRows*) a výsledek jenž je opět opět maticí, je transponován. Původní sloupce se tak stávají řádky a tak může být proveden výpočet logických součinů všech hodnot v řádcích pomocí mapování funkcionálu *foldr1* na jednotlivé (souvislé) řádky. *Foldr1* aplikuje binární funkci, která je uvedena jako první parametr (zde logický součet *ladd*), mezi všemi hodnotami daného řádku (tj. provede výpočet $a_{i_k j} \vee a_{i_{k-1} j} \vee \ldots \vee a_{i_1 j}$ pro dané j). Následně je vypočítán produkt ze seznamu výsledků této funkce pro všechny řádky (= původní sloupce).

Vyčerpávající prohledávání všech n-tic je pomalé a neefektivní, neboť výpočetní složitost je exponenciální. Částečně jej lze urychlit využitím již vypočítaných dílčích hodnot. Mnohé hodnoty jako např. řádkové součiny resp. dílčí sloupcové počty jsou v průběhu výpočtu využívány vícenásobně. Ani toto urychlení však nemusí

být dostatečné v situacích, kdy je graf rozsáhlý a doba výpočtu je limitována, neboť výsledek musí být k dispozici "okamžitě". Proto se stále hledají nové metody výpočtu charakteristických čísel. Pro tyto účely lze využít různé invariantní charakteristiky diagnostického grafu (např. spektrum grafu, viz [3]).

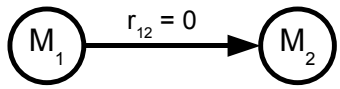
1.4. Hodnocení diagnostického grafu s ohledem na vlastnosti atomických kontrol

Všechny výše uvedené vztahy a závěry vycházely ze zjednodušujícího předpokladu, že správný modul vždy odhalí po atomické kontrole modul chybný (viz vztah 1.1 na straně 13). Skutečnost je však jiná, neboť vždy existuje určitá pravděpodobnost, že chyba modulu nebude odhalena. Pravděpodobnost odhalení chybného modulu v rámci atomické kontroly je označována jako **důvěryhodnost** atomické kontroly P_{AT}.

Zohlednění důvěryhodnosti jednotlivých atomických kontrol změní pohled na diagnostický graf a jeho syndrom. Například získání nulového syndromu negarantuje bezchybnost všech modulů.

Z tohoto důvodu je nutné rozhodnout, do jaké míry syndrom odráží skutečný stav modulů, tj. do jaké míry mu můžeme důvěřovat, resp. tuto míru přesně kvantifikovat. To zajišťuje tzv. **aposteriorní pravděpodobnost správnosti systému**, kterou lze přiřadit každému diagnostickému grafu. Pravděpodobnost je *aposteriorní*, neboť hodnotí systém až po (= a posteriori) provedení atomických kontrol.

Výpočet aposteriorní pravděpodobnosti si ukážeme na jednoduchém příkladě systému, jehož diagnostický graf je na na obrázku 1.11.



Obrázek 1.11.: Ukázkový diagnostický graf pro výpočet aposteriorní pravděpodobnosti

Modul M_1 zde kontroluje modul M_2, výsledek atomické kontroly r_{12} je roven 0.

Pro výpočet aposteriorní pravděpodobnosti vycházíme z dílčích pravděpodobností výsledků atomických kontrol pro jednotlivé stavy systému. Tyto pravděpodobnosti jsou přehledně znázorněny v tabulce 1.1 na následující straně.

výsledek atomické kontroly r_{12}		Modul M_1	
		správný stav	chybný stav
Modul M_2	správný stav	$r_{12} = 0$	$r_{12} = 1$ P_A
			$r_{12} = 0$ $1 - P_A$
	chybný stav	$r_{12} = 1$ P_{AT}	$r_{12} = 1$ P_B
		$r_{12} = 0$ $1 - P_{AT}$	$r_{12} = 0$ $1 - P_B$

Tabulka 1.1.: Pravděpodobnosti výsledků atomických kontrol

Pravděpodobnost P_A odpovídá situaci, kdy výsledkem kontroly správného modulu chybným je **1** (chybný výsledek). P_B je pravděpodobnost výsledku 1 u kontroly chybného modulu chybným modulem (správný výsledek). P_{AT} je pravděpodobnost výsledku, že správný modul zkontroluje chybný modul opět s výsledkem 1 (správný výsledek).

Hodnoty pravděpodobností $P_{A,}, P_B, P_{AT}$ je možno získat buď na základě statistických dat z testů systému (tj. získaných po určité době pozorování systému), nebo z technických specifikací uvedených v dokumentaci systému.

Aposteriorní pravděpodobnost správnosti systému lze na základě známých hodnot $P_{A,}, P_B, P_{AT}$ vypočítat pomocí Bayesova vztahu pro různé hypotézy.

V našem ukázkovém systému lze definovat následující hypotézy H_i:

H_1: M_1 je správný M_2 je správný
H_2: M_1 je správný M_2 je chybný
H_3: M_1 je chybný M_2 je správný
H_4: M_1 je chybný M_2 je chybný

Apriorní pravděpodobnosti těchto hypotéz jsou rovny:

$$P(H_1) = P_{M_1} \cdot P_{M_2}$$

$$P(H_2) = P_{M_1} \cdot (1 - P_{M_2})$$

$$P(H_3) = (1 - P_{M_1}) \cdot P_{M_2}$$

$$P(H_4) = (1 - P_{M_1}) \cdot (1 - P_{M_2}),$$

kde P_{M_1} je apriorní pravděpodobnost, že modul M_1 je správný, a P_{M_2} je pravděpodobnost správnosti modulu M_2. Apriorní pravděpodobnosti lze zjistit v dokumentaci ke konkretním modulům cílového systému. Pro účely obecného ohodnocení diagnostických grafů lze volit shodné apriorní pravděpodobnosti u všech modulů, navíc lze tyto pravděpodobnosti odhadnout, neboť tato pravděpodobnost se ve většině případů blíží hodnotě 1.

Podmíněné pravděpodobnosti $P(R/H_i)$ jsou pravděpodobnosti získání syndromu R po provedení atomických kontrol za situace, kdy stav systému odpovídá hypotéze H_i.

Pro uvažovaný příklad je $R = \{r_{12} = 0\}$. Za předpokladu, že $P_A = P_B = 0.5$, jsou podmíněné pravděpodobnosti následující (viz tabulka 1.1):

$P(R/H_1) = 1$

$P(R/H_2) = 1 - P_{AT}$

$P(R/H_3) = 1 - P_A = 0.5$

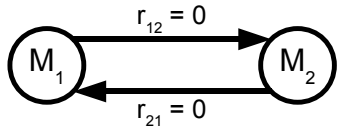
$P(R/H_4) = 1 - P_B = 0.5$

Aposteriorní pravděpodobnosti hypotézy H_i a obdrženém syndromu R, kde $i = 1,\ldots,n$, jsou určeny následujícím Bayesovým vztahem:

$$P(H_i/R) = \frac{P(H_i) \cdot P(R/H_i)}{\sum_{i=1}^{n}(P(H_i) \cdot P(R/H_i))} \tag{1.9}$$

Za předpokladu, že apriorní pravděpodobnosti jsou rovny $P_{M_1} = P_{M_2} = 0.9$, je aposteriorní pravděpodobnost správnosti systému na obrázku 1.11, tj. pravděpodobnost P(H_1/R), rovna 0.92.

Podobně lze vypočítat aposteriorní pravděpodobnost správnosti systému pro libovolný graf, pokud jsou známy dílčí pravděpodobnosti. Je například zřejmé, že pro graf na obrázku 1.12 je aposteriorní pravděpodobnost správnosti systému větší než u grafu s jednou kontrolou (viz 1.11). Ověřte jako cvičení toto tvrzení alespoň pro dílčí pravděpodobnosti $P_A = P_B = 0.5$ a $P_{M_1} = P_{M_2} = 0.9$.



Obrázek 1.12.: Cvičný DG pro výpočet aposteriorní pravděpodobnosti

Kromě toho, že uvedené aposteriorní pravděpodobnosti umožňují ohodnotit a porovnat různé diagnostické grafy, lze je využít i pro zhodnocení příspěvku každé jednotlivé atomické kontroly.

V našem případě můžeme například ohodnotit příspěvek atomické kontroly $r_{12} = 0$. Apriorní pravděpodobnost, že jsou oba moduly správné, je bez provedení atomické kontroly rovna (dílčí pravděpodobnosti jsou stále stejné):

$P(H_1) = P_{M_1} \cdot P_{M_2} = 0.9 \cdot 0.9 = 0.81$

Po provedení atomické kontroly s výsledkem $r_{12} = 0$ vzroste jistota správnosti obou modulů na hodnotu $P(H_0/R) = P(H_0/r_{12} = 0) = 0.92$. Pozitivním efektem provedení dodatečné atomické kontroly je zvýšení naší jistoty o správnosti systému o $\triangle = 0.11$ tj. o 11%.

Za zmínku stojí, že použití Beyesova vztahu (1.9) je pro složité diagnostické grafy s velkým počtem modulů velmi náročné.

Možnou alternativou je proto metoda založená na přímém využití jednoduchých invariantů grafů. Podle této metody bude mezi diagnostickými grafy s n uzly a k hranami testy optimální ten, pro nějž pro každé $i = 1, \ldots, n$ platí:

$$\alpha_i^+ = \alpha_i^- = n/k, \tag{1.10}$$

kde α_i^- je počet vstupních hran uzlu i
α_i^+ je počet výstupních hran uzlu i.

Kritérium hodnocení diagnostických grafů zde zůstává stejné: pravděpodobnost, že výsledek samokontroly odpovídá skutečnému stavu systému.

V případě, že podíl n/k není celočíselný, nemusí tato metoda poskytovat zcela přesné výsledky, přičemž záleží na konkrétních hodnotách n a k a jejich vztahu.

Uvažované metody hodnocení diagnostických grafů, jmenovitě:

- použití charakteristických čísel (vztah 1.6)
- aposteriorní pravděpodobnosti (vztah 1.9)
- jednoduché invarianty grafu (vztah 1.10),

mají své výhody i nevýhody.

Z tohoto důvodu je nutno v každém konkrétním případě provést analýzu požadavků a zvolit nejvhodnější resp. nejefektivnější metodu.

1.5. Pořadí provedení atomických kontrol

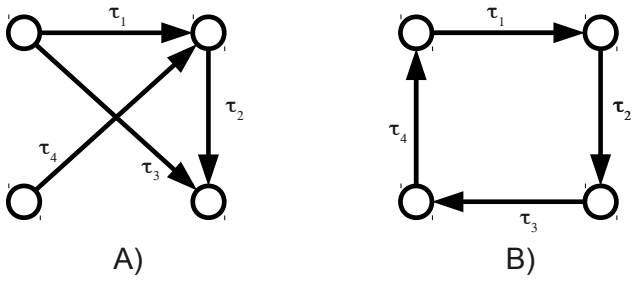
Předchozí podkapitoly se soustředily na strukturální, tj. prostorové aspekty samodiagnostikujících se systémů. V praxi však nelze pominout ani časový aspekt, to jest čas provedení jednotlivých atomických kontrol vytvářejících syndrom a jejich pořadí.

Pro zjednodušení předpokládejme následující dvě omezení:

1. čas vykonávání všech atomických kontrol je stejný a konstantní (tj. $\forall i \; t_i = t_{AT}$, kde t_i je čas provedení i-té atomické kontroly)
2. modul je schopen provádět v daném časovém okamžiku pouze jedinou atomickou kontrolu. Není dokonce schopen souběžně jednu kontrolu iniciovat (na DG šipka z uzlu) a zároveň na druhou reagovat (šipka do uzlu).

Tato omezení se u široké třídy systémů blíží realitě. Samodiagnostikující systémy jsou běžně homogenní (tj. homogenní jsou i komunikační kanály včetně doby odezvy) a moduly jsou jednoduché (tj. paralelismus je možný pouze na úrovni systémů, nikoliv na úrovni modulů).

Uvažujme následující dva grafy na obrázku 1.13.



Obrázek 1.13.: Pořadí provedení atomických kontrol

Oba grafy obsahují stejný počet hran, resp. atomických kontrol. Lze však snadno ukázat, že atomické kontroly v případě grafu B mohou být provedeny rychleji než v případě grafu A. V situacích, kdy je čas provedení atomických kontrol prioritní, bude proto dána přednost grafu B před A, a to i tehdy, jestliže jsou jeho strukturální kritéria horší.

Celkový čas provedení atomických kontrol je kromě jejich počtu determinován i pořadím jejich vykonávání a především možností paralelního vykonávání více atomických kontrol (paralelních v rámci celého systému!). Hlavním cílem je tudíž najít takové uspořádání AT a takovou míru paralelismu, aby byl celkový čas provádění diagnostický kontrol (dále označován jako T_Σ) minimální.

Například pro diagnostický graf na obrázku 1.13 (B) lze volit pořadí $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4$, v němž jsou jednotlivé atomické kontroly prováděny sekvenčně jedna po druhé a $T_\Sigma = 4t_{AT}$. Lze však zvolit i pořadí, v němž jsou dvě kontroly prováděny současně. Při zohlednění omezujících podmínek existuje jen jedno takové pořadí: $\tau_1, \tau_3 \rightarrow \tau_2, \tau_4$ (AT oddělené čárkou jsou prováděny souběžně). Celkový čas provedení $T_\Sigma = 2t_{AT}$ je dvakrát kratší. U grafu 1.13 (A) lze paralelismus využít pouze pro dvě AT (τ_3, τ_4). Paralelní provedení jakékoliv jiné dvojice atomických kontrol není kvůli výše uvedeným omezení možné. Modul by byl nucen vykonat paralelně vyslání dvou požadavků na AT, resp. odpovědět na dva požadavky o AT, nebo zároveň vyslat požadavek a odpovědět na žádost jiného modulu.

Jak vyplývá z příkladů, je pro určení minimální hodnoty T_Σ rozhodující především stupeň jednotlivých uzlů v DG a to bez ohledu na orientaci jednotlivých hran, tj. stupeň určený v odpovídajícím neorientovaném grafu. Tento lokální stupeň pro uzel i označíme α_i.

Celkový počet hran DG je pak roven $Q = \frac{1}{2}\sum \alpha_i$.

Diagnostika je prováděna v jednotlivých následných krocích, z nichž každý trvá t_{AT}. Vzhledem k tomu, že v každé atomické kontrole jsou angažovány dva moduly, může být v každém kroku provedeno maximálně $d = \frac{N}{2}$ kontrol.

Celkový počet kroků nutných pro provedení všech AT nemůže být menší než $\left\lceil \frac{Q}{d} \right\rceil$ a zároveň nemůže být menší než největší stupeň uzlů v grafu. Minimální počet kroků je tak roven:

$$K_{min} = \max\left(\left\lceil \frac{Q}{d} \right\rceil, \max\{\alpha_i\} \right), \text{ pro } \forall i \in 1..N$$

Minimální čas nutný pro provedení série atomických kontrol je proto roven:

$$T_{min} = K_{min} \cdot t_{AT}$$

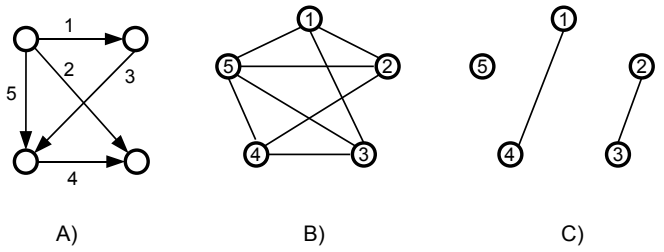
Cílem optimalizace pořadí jednotlivých atomických kontrol je dosažení tohoto minimálního času.

DEFINICE 1.5:

Pořadí provedení atomických kontrol je pro daný diagnostický graf *optimální*, pokud zajišťuje minimální čas provedení atomických kontrol T_{min}.
□

Otázkou však zůstává, jak optimální pořadí získat, neboť vhled je možný pouze u jednoduchých grafů. Pro tento účel lze využít následující grafově orientovaný algoritmus, který lze provést nad maticí sousednosti DG a maticemi odvozenými.

Algoritmus nalezení optimálního pořadí



Obrázek 1.14.: Grafy algoritmu nalezení optimálního pořadí AT

1. V původním diagnostickém grafu $G(V,E)$ očíslujeme jednotlivé hrany od 1 do Q. Příkladem je graf 1.14(A).
2. Pro graf $G(V,E)$ vytvoříme odpovídající graf sousednosti uzlů $I(G)$ (angl. *adjacent graph*) viz 1.14(B). Uzly tohoto grafu odpovídají hranám původního grafu G (tj. jsou označeny $1\ldots Q$) a odpovídají tak jednotlivým atomickým kontrolám. Tyto uzly jsou spojeny hranami v případě, že příslušné hrany sdílejí v původním grafu uzel (bez ohledu na orientaci).
3. Pro graf sousednosti uzlů vytvoříme doplňkový graf $\bar{I}(G)$, tj. graf, v němž jsou uzly spojeny hranou jen tehdy, pokud tomu tak nebylo v grafu $I(G)$. Jinak řečeno množina uzlů zůstává stejná, množina hran je doplňkem (komplementem) množiny hran grafu $I(G)$. Univerzální množinou je v tomto případě množina všech možných hran, tj. množina hran úplného grafu. Příklad viz 1.14(C).

Výsledný graf $\bar{I}(G)$ lze již přímo použít pro návrh pořadí provedení atomických kontrol. Jsou-li totiž uzly i a j ($i \neq j$), jenž odpovídají jednotlivým atomickým kontrolám τ_i a τ_j, spojeny v doplňkovém grafu hranou, pak lze příslušné atomické kontroly provádět souběžně.

V našem případě tak lze navrhnout např. pořadí $5 \rightarrow 1,4 \rightarrow 2,3$, resp. pořadí $2,3 \rightarrow 1,4 \rightarrow 5$ (existují i další kombinace). Všechna tato pořadí jsou optimální, a čas provedení je tudíž roven $3t_{AT}$.

V praxi se všechny kroky výše uvedeného algoritmu provádějí nad maticemi sousednosti jednotlivých grafů. Výsledkem je v tomto případě matice sousednosti doplňkového grafu sousednosti uzlů tj. $M[\bar{I}(G)]$.

Matice sousednosti pro graf $\bar{I}(G)$ (viz 1.14-C) je následující:

	1	2	3	4	5
1	-	0	0	1	0
2	0	-	1	0	0
3	0	1	-	0	0
4	1	0	0	-	0
5	0	0	0	0	-

Z matice sousednosti $M(\bar{I}(G))$ je zřejmé, jaké atomické kontroly mohou být provedeny souběžně (na průsečíku je jednička), ale pořadí provedení už nemusí být tak snadno odvoditelné, a to především u grafů s větším počtem atomických kontrol. Zde mohou pomoci triviální záměny uzlů v grafu $\bar{I}(G)$(tj. odpovídajících sloupců i řádků v matici sousednosti), které nemění graf, pouze jeho maticovou representaci. Pomocí série záměn lze dosáhnout stavu, kdy jsou jedničková pole soustředěna těsně podél diagonály. V našem případě lze tohoto stavu dosáhnout po vzájemné záměně druhého a čtvrtého uzlu, resp. druhého a čtvrtého řádku a zároveň i druhého a čtvrtého sloupce.

	1	**4**	3	**2**	5
1	-	**1**	0	0	0
4	**1**	-	0	0	0
3	0	0	-	**1**	0
2	0	0	**1**	-	0
5	0	0	0	0	-

Pokud dosáhneme tohoto stavu, lze pořadí vykonávání AT přečíst přímo ze záhlaví tabulky, jež je tvořena transformovanou posloupností označení uzlů. V našem případě je výsledkem záměn posloupnost 1, 4, 3, 2, 5, která určuje jedno z optimálních řešení. Stačí pouze zohlednit všechny přípustné souběhy atomických kontrol, tj. v daném případě přepsat posloupnost do tvaru $1,4 \rightarrow 3,2 \rightarrow 5$. Důkaz, že výše popsaná transformace vede k přeuspořádání záhlaví, které odpovídá optimální posloupnosti AT je snadný (ověření proveďte sami).

Algoritmus nalezení optimálního pořadí AT je obdobou algoritmu nalezení hamiltonovské kružnice (nad $\bar{I}(G)$), neboť i zde je omezení, že nalezená cesta prochází každý uzel právě jednou. Z tohoto lze odvodit, že stejně jako u hamiltonovských kružnic neexistuje žádný efektivní a obecný algoritmus pro nalezení optimálního pořadí (problém je NP-úplný). U systémů s větším počtem atomických kontrol tak

může být nalezení optimálního pořadí extrémně náročné na čas, neboť je nutné testovat všechny alternativy záměn[1].

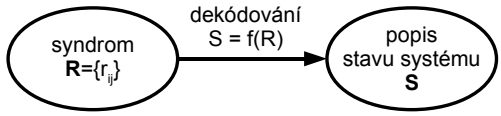
1.6. Diagnostika na základě výsledků atomických kontrol

Provedení atomických kontrol a získání syndromu není konečným cílem. Hlavním cílem je zjištění stavu systému, tj. nalezení všech chybných resp. správných modulů. Po provedení atomických kontrol a získání syndromu nastupuje další fáze, vlastní **diagnostika systému**. Tato fáze je nezbytná, pokud je výsledkem jedné nebo více atomických kontrol hodnota „1“, neboť to svědčí o přítomnosti chybných modulů v systému.

V uvažovaném modelu se předpokládá, že diagnostiku bude provádět myšlený externí pozorovatel, který získává syndrom ze systému. Externí pozorovatel není součástí systému, je sám bezchybný a bezchybný je i přenos údajů ze systému k vnějšímu pozorovateli.

Cílem samodiagnostiky je zjištění, jaký modul, resp. jaké moduly selhaly, respektive zpřesnění informace o druhu chyby. V některých případech není možné určit konkrétní chybné moduly, ale je možné pouze stanovit podmnožinu modulů, v níž jsou chybné moduly soustředěny, nicméně však může obsahovat i moduly bezchybné.

Libovolnou diagnostiku tak lze popsat jako funkci převádějící získaný syndrom na popis stavu systému $S = \mathrm{f}(R)$, respektive jako dekódování syndromu na popis stavu (viz obrázek 1.15).



Obrázek 1.15.: Podstata diagnostiky

Toto dekódování je dále závislé na různých předběžných informacích o stavu systému, včetně apriorních předpokladů o chování systému v různých stavech, o vlastnostech jednotlivých atomických kontrol, nebo o režimech selhání modulů systému apod.

[1] příkladem složitého, ale v rozumném čase řešitelného hamiltonovského problému je nalezení nejkratší cyklické letecké trasy mezi hlavními městy států USA.

Algoritmy samodiagnostiky, které se používají v praxi, zohledňují především následující dodatečné informace:

- předpoklad o výsledcích atomických kontrol prováděných jak správnými tak i selhávajícími moduly
- pořadí provádění množiny atomických kontrol
- spolehlivost jednotlivých modulů systému a spolehlivost spojení mezi moduly
- předpoklad o maximálním počtu chybných modulů. Tento předpoklad určuje mez, za kterou mohou být výsledky samodiagnostiky s určitou pravděpodobností považovány za chybné. Respektive lze na základě požadované jistoty určit mez, v jejímž rozsahu lze s danou pravděpodobností předpokládat, že zjištěný stav systému odpovídá skutečnosti.
- předpoklad o režimech selhání jednotlivých modulů systému. Většinou je uvažováno, zda jsou selhání jednotlivých modulů řízená či nikoliv. Selhání modulů mohou být například stálá, přechodná nebo nahodilá.
- možností obnovení systému (tj. nahrazení chybného modulu nebo jeho průběžné odstranění).

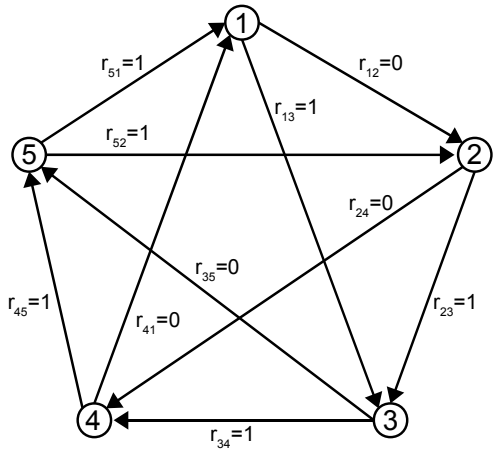
První obecné algoritmy samodiagnostiky začaly být navrhovány počínaje rokem 1967, kdy byl publikován Preparatův článek [1]. Od té doby byla navržena celá řada dalších algoritmů samodiagnostiky. Následující přehled se věnuje těm nejzákladnějším a nejužitečnějším.

Algoritmy založené na tabulce syndromu

Tabulkové algoritmy pracují s **tabulkou syndromu**, což je maticová representace syndromu. Tabulka syndromu $M_R[r_{ij}]$ je čtvercová matice o rozměru $N \times N$, kde N je počet modulů. Pokud je součástí syndromu výsledek atomické kontroly, kterou provádí i-tý modul na modulu j-tém, to jest hodnota r_{ij}, pak tabulka syndromu obsahuje tuto hodnotu v i-tém řádku a j-tém sloupci. Položka v tomto případě obsahuje hodnotu nula nebo jedna.

Pokud není atomická kontrola mezi určitými dvěma moduly systému provedena (tj. není v syndromu k dispozici), pak je tato situace graficky representována pomlčkou v průsečíku příslušného sloupce a řádku. Při representaci tabulky syndromu v počítači lze použít například hodnotu -1.

Obrázek 1.16 na následující straně znázorňuje dvě možné representace syndromu. Vlevo representaci pomocí ohodnoceného diagnostického grafu, vpravo pomocí ekvivalentní tabulky (matice) syndromu.



$M_R =$

	1	2	3	4	5	
	-	0	1	-	-	1
	-	-	1	0	-	2
	-	-	-	1	0	3
	0	-	-	-	1	4
	1	1	-	-	-	5

Obrázek 1.16.: Dvě varianty representace syndromu

Pokud je graf t-diagnostikovatelný, lze pomocí tabulkových algoritmů identifikovat všechny chybné moduly, ale samozřejmě pouze v případě, že jejich počet nepřekročí hodnotu t. V opačném případě bude algoritmus s velkou pravděpodobností schopen tuto situaci detekovat, ale chybné moduly nemohou být identifikovány (tj. program může nanejvýše signalizovat, že počet chybných modulů je příliš velký). Výsledkem však může být i zcela zmatečná identifikace chybných modulů.

Přípravným krokem tabulkových algoritmů je proto určení hodnoty t z diagnostického grafu. Běžnější je však využití ad hoc navrženého diagnostického grafu se zaručenou hodnotou t, která je obvykle zároveň optimální, tj. je rovno $t_{max} = \lfloor \frac{N-1}{2} \rfloor$.

Před volbou a použitím tabulkového algoritmu je navíc nutné zohlednit vlastnosti atomických kontrol. Většina tabulkových algoritmů pracuje s vlastnostmi AT podle definice 1.1 na straně 13 .

Při volbě konkrétního tabulkového algoritmu je rozhodující počet modulů v systému, neboť tyto algoritmy jsou na počtu modulů silně závislé. Větší počet modulů může algoritmus výrazně zkomplikovat, a tak může být čas provedení diagnostiky pro větší počet modulů neakceptovatelný (např. s exponenciální časovou složitostí), resp. výrazně závislý na konkrétním syndromu. Další vývoj se proto zaměřuje na návrh tabulkových algoritmů s akceptovatelnou a predikovatelnou časovou složitostí.

Pro účely vysvětlení principů byl zvolen relativně jednoduchý algoritmus, jenž je efektivní pro malé systémy (byl primárně vytvořen pro výukové účely). Pro větší systémy ($N > 10$) je také použitelný, ale v určitých situacích může být časově náročný. Pravděpodobnost tohoto chování je však u reálných systémů velmi malá.

Navíc tento algoritmus používá mnohé přístupy, které lze nalézt i v komplexnějších algoritmech.

vstupní data algoritmu:

1. syndrom $R = \{r_{ij}\}$ v podobě tabulky M_R
2. hodnota t, typicky je to zároveň optimální hodnota t_{max} (je tomu i v příkladu z obrázku 1.16, kde $t_{max} = 2$)

ALGORITMUS

1.krok

Spočítání celkového počtu jedniček v každém řádku a sloupci. Celkový počet jedniček v řádku s indexem x_i je roven $S_{x_i} = \sum_{j=1}^{n} r_{ij}$, počet jedniček ve sloupci s indexem y_j je roven $S_{y_j} = \sum_{i=1}^{n} r_{ij}$.

2.krok

Sumy pro řádek a sloupec se stejným indexem se sečtou do jediného součtu, tj. $S_i = S_{x_i} + S_{y_i}$ a to pro každé $i = 1 \ldots N$. Výsledkem tohoto kroku je tudíž vektor hodnot $\langle S_1, S_2, \ldots, S_N \rangle$.

3. krok

Každá vypočtená hodnota S_i je porovnána s hodnotou t. Mohou nastat tři situace:

A) $S_i > t$

B) $S_i = t$

C) $S_i < t$

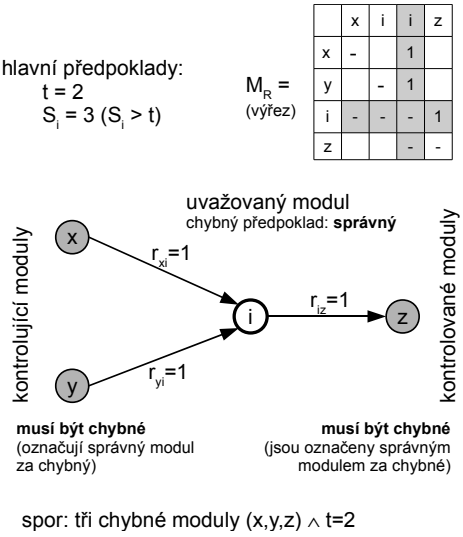
Další krok záleží na tom, jaký je výsledek porovnání. Algoritmus se v tomto bodě dělí do tří větví podle hodnoty S_i.

Nejjednodušší je použití větve **A**, neboť matice je bezprostředně redukována na jednodušší. Je proto výhodné nejdříve řešit sloupce a řádky, pro něž je $S_i > t$. Teprve ve druhé fázi jsou vyřešeny řádky a sloupce, u nichž $S_i = t$ (podle větve **B**), a zcela nakonec nejsložitější případ $S_i < t$ (pokud nějaký takový nevyřešený řádek-sloupec zbude).

Větev A) $S_i > t$

Modul i je určitě chybný.

Důkaz lze provést sporem (viz obrázek 1.17). Obrázek je vytvořen pro konkrétní t a S_i, ale lze jej snadno zobecnit.



Obrázek 1.17.: Tabulkový algoritmus, důkaz předpokladu větve A

krok A1: Je nutno odstranit modul i ze syndromu, tj. eliminovat i-tý řádek a i-tý sloupec.

krok A2: Algoritmus dále pokračuje na redukované tabulce syndromu počínaje prvním krokem.

Větev B) $S_i = t$

Nelze přímo určit, zda je daný modul i chybný nebo nikoliv.

krok B1: Nejdříve **předpokládejme**, že modul je správný (bezchybný).

krok B2: Na základě tohoto předpokladu je možno dále předpokládat, že chybné jsou i ty moduly, které při AT označily daný modul za chybný, nebo jsou naopak daným modulem označeny za chybné.

Takže modul j je chybný, pokud platí: $r_{ij} = 1 \vee r_{ji} = 1$.

krok B3: Dále lze na základě tohoto předpokladu identifikovat další potenciálně správné moduly, a to rekurzivně. Správný modul musí být (podle přijatých předpokladů) korektně identifikován jiným správným modelem (tj. atomická kontrola musí vrátit 0). Rekurzivní procházení začíná v daném modulu (u něhož se správnost předpokládá) a postupně se rozšiřuje na další moduly. Během procházení lze identifikovat i další potenciálně chybné moduly podobně jako v kroku B2. Ty jsou při kontrole správným modulem označeny jako chybné, nicméně od chybného modulu však dále procházet nejde.

krok B4: Další krok závisí na tom, zda došlo při rekurzivním procházení podle kroku B3 k rozporu či nikoliv. Rozpor vznikne, pokud dva předpokládaně správné moduly otestují jiný modul nekonzistentně (tj. jeden jej ohodnotí jako správný, druhý jako chybný), resp. pokud (předpokládaně) správný modul označí moduly získané v kroku B2 jako správné (původní předpoklad je, že jsou chybné).

Pokud není rozpor nalezen, pak je předpoklad v kroku **B1 potvrzen** a potvrzeny jsou předpoklady učiněné v krocích B2 a B3. Pokud ještě zůstane nějaký modul bez potvrzeného odhadu, je možno pokračovat krokem 1.

Je-li rozpor nalezen, pak je původní předpoklad v kroku **B1 nesprávný**. Daný modul je označen za chybný a je eliminován z tabulky (jako v kroku A1). Algoritmus pokračuje krokem 1 na redukované tabulce.

Větev C) $S_i < t$

Pokud nastane tato situace, závisí další pokračování na celkovém počtu modulů v systému (N).

Je-li $N < 13$, stačí nalézt sloupec obsahující samé nuly. Tento sloupec odpovídá modulu, jež lze považovat za správný. S touto informací lze snadno rozhodnout stavy všech modulů (viz krok B2, B3).

Je-li $N \geq 13$, je možno dokázat, že v diagnostickém grafu existuje cyklus délky $t+1$, jehož všechny hrany jsou ohodnoceny nulou. Tento cyklus lze zjistit přímo z matice M_R. Všechny moduly, které jsou representovány v grafu uzly nalezeného cyklu, jsou správné. Stav ostatních modulů lze odvodit stejně jako v krocích B2 a B3.

KONEC ALGORITMU

Aplikace výše uvedeného algoritmu na diagnostický graf a syndrom uvedený na obr. 1.16 na straně 32 vede k závěru, že chybné jsou moduly M_3 a M_5, přičemž algoritmus může využít větve A.

Na závěr popisu algoritmu ještě malé zamyšlení nad jeho efektivností, resp. obecnou výpočetní náročností algoritmů nad tabulkou syndromů. Zpracování tabulky je velmi výhodné, pokud lze jít větví $S_i > t$. Její provedení je velmi rychlé (v konstantním čase), neboť vyjmutí sloupce a řádku nemusí být provedeno přímo (fyzicky), ale postačí pouze označit modul za vymazaný, což se musí zohlednit při dalším zpracování na redukované tabulce. Algoritmus v ostatních větvích je časově výrazně složitější, i když jeho časovou složitost nelze jednoduše vyjádřit. Navíc o něco složitější je i kód, což může být problém, pokud diagnostiku provádí jednoduché zařízení.

Pravděpodobnost, že v dané tabulce syndromů je pro alespoň jeden modul i, $i \in 1..N$ splněna podmínka $S_i > t$, lze relativně snadno vypočítat. Tato pravděpodobnost závisí nejen na t a tím i nepřímo na počtu modulů N (předpokládáme-li $t = t_{max}$), ale také na pravděpodobnosti, že chybný modul označí jakýkoliv jiný modul za chybný, tj. na pravděpodobnosti, že náhodná veličina X z definice 1.1 na straně 13 nabude hodnoty jedna (označení $P(X = 1)$).

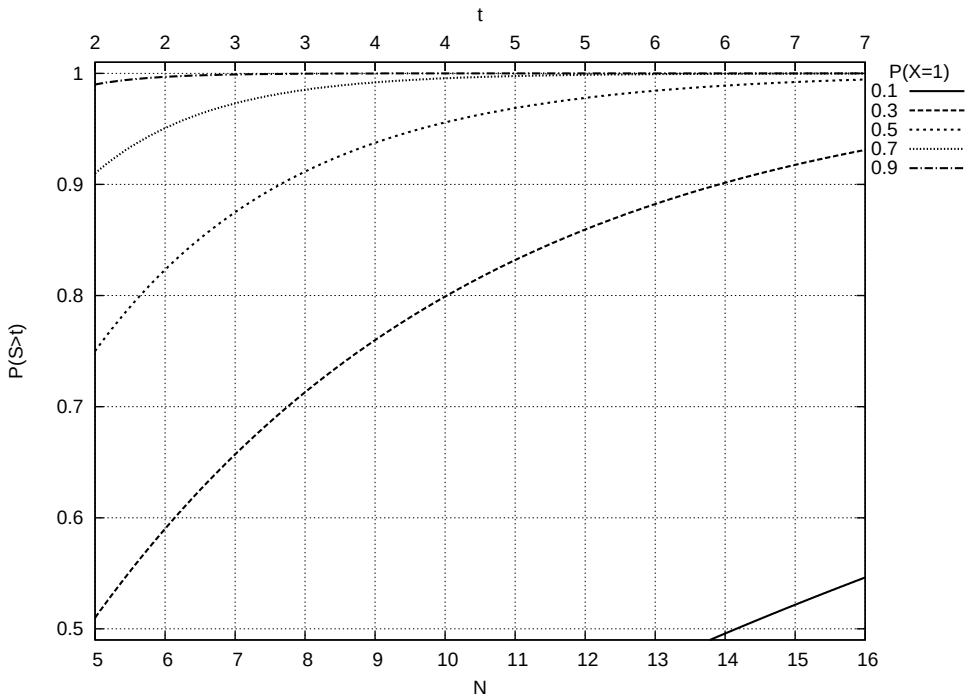
Tato závislost je pro pět vybraných hodnot pravděpodobnosti $P(X = 1)$ uvedena na obrázku 1.18 na následující straně. Křivky jsou nakresleny jako spojité, ve skutečnosti jsou však definovány jen pro celá čísla a funkční hodnoty se mění jen v při změně t_{max} (lépe je však vidět trend). Na horním okraji je pomocná osa x zobrazující hodnoty t_{max} odpovídající počtu modulů v systému (dolní osa).

Z grafu lze snadno vidět, že pro běžně používanou hodnotu P(X=1) = 0.5 odpovídající zcela náhodnému chování modulu je pravděpodobnost rychlého nalezení alespoň jednoho chybného modulu vysoká (pro $N > 5$ větší než 0.9). V případě pesimistického chování chybných modulů (chybný modul označuje většinu ostatních zařízení jako chybná, typické například u velmi jednoduchých zařízení) je tato pravděpodobnost téměř stoprocentní už i pro malé N. Naopak u optimisticky chybných modulů je tato pravděpodobnost výrazně nižší (pro P(X=1) = 0.1 je poloviční i u grafů s 15 moduly).

Algoritmy založené na tabulce potenciálních syndromů

Dalším příkladem tabulkových algoritmů jsou algoritmy založené na *tabulce potenciálních syndromů*. Jeden z prvních algoritmů tohoto typu byl navržen Vedeshenkovem [4].

Tabulka potenciálních syndromů se vytváří před začátkem diagnostické procedury. Podkladem pro vytvoření tabulky je matice sousednosti diagnostického grafu



Obrázek 1.18.: Pravděpodobnost $P(S_i > t)$

a reprezentace vlastností atomické kontroly (většinou se volí opět reprezentace podle definice 1.1 na straně 13).

Tabulka potenciálních syndromů zahrnuje všechny syndromy, které mohou být obdrženy pro různé přípustné kombinace stavů modulů. Zpravidla se opět předpokládá, že počet chybných modulů v systému nepřekročí hodnotu t. V tomto případě stačí uvažovat jen ty kombinace stavů, v nichž je počet chybných modulů menší než t. Počet stavů a tudíž i potenciálních syndromů je v tomto případě roven:

$$Q = \sum_{i=1}^{t} \binom{n}{i}$$

Tak například pro systém, jehož diagnostický graf je zobrazen na 1.16 na straně 32, budou uvažovány pouze následující situace:

S_1: M_1 je chybný
S_2: M_2 je chybný
S_3: M_3 je chybný
S_4: M_4 je chybný
S_5: M_5 je chybný
S_6: M_1 a M_2 jsou chybné
S_7: M_1 a M_3 jsou chybné
S_8: M_1 a M_4 jsou chybné
S_9: M_1 a M_5 jsou chybné
S_{10}: M_2 a M_3 jsou chybné
S_{11}: M_2 a M_4 jsou chybné
S_{12}: M_2 a M_5 jsou chybné
S_{13}: M_3 a M_4 jsou chybné
S_{14}: M_3 a M_5 jsou chybné
S_{15}: M_4 a M_5 jsou chybné

Když skutečný stav systému neodpovídá žádné z uvažovaných situací (například v případě, když je počet chybných modulů větší než dva) je výsledek diagnostiky nesprávný nebo dokonce zavádějící. Diagnostické algoritmy, které používají tabulky potenciálních syndromů, proto mají jen omezenou důvěryhodnost. Tuto důvěryhodnost však lze předem spočítat.

Tabulka potenciálních syndromů má následující sloupce:

- označení uvažované situace (S_i)
- čísla chybný modulů $\{M_j\}$, $j = 1 \ldots n$ pro danou situace S_i
- označení potenciálního syndromu (R_p^i) pro danou situaci S_i
- jednotlivý prvky syndromu $\{r_{ij}\}$ pro danou situaci, tyto prvky tvoří l sloupců, kde l je počet atomických kontrol (resp. hran).

Tabulka potenciálních syndromů obsahuje tolik řádků, kolik je uvažovaných situací. Hodnoty jednotlivých prvků potenciálního syndromu $\{r_{ij}\}$ jsou stanoveny podle zvolené reprezentace atomické kontroly. V případě Preparatovy representace mohou jednotlivé prvky nabývat hodnot 0,1 nebo X v závislosti na stavech modulů M_i a M_j. Hodnota označovaná jako X vyjadřuje náhodný výsledek kontroly prováděné chybným modulem. Ve skutečném syndromu může nabývat hodnoty 0 nebo 1 s určitým náhodným rozdělením. Pravděpodobnostní charakteristiky atomických kontrol nejsou pro tento algoritmus podstatné.

Pro systém s diagnostickým grafem zobrazeným na obrázku 1.16 na straně 32 je tabulka potenciálních syndromů následující:

Jak lze z tabulky snadno vidět, kterékoli dva potenciální syndromy (tj. kterékoli dva řádky v tabulce) jsou odlišné alespoň v jednom prvku. Tato odlišnost syndromů je klíčová pro diagnostiku na základě potenciálních syndromů.

Vlastní algoritmus je prováděn po skončení běhu atomických kontrol, to jest po získání skutečného syndromu. Algoritmus spočívá v porovnání skutečného syndromu se syndromy potenciálními. Cílem je nalézt potenciální syndrom, který od-

S_i	chybné	potenciál.	atomické kontroly									
	moduly	syndrom	r_{12}	r_{13}	r_{23}	r_{24}	r_{34}	r_{35}	r_{45}	r_{41}	r_{51}	r_{52}
S_1	M_1	${r_p}^1$	x	x	0	0	0	0	0	1	1	0
S_2	M_2	${r_p}^2$	1	0	x	x	0	0	0	0	0	1
S_3	M_3	${r_p}^3$	0	1	1	0	x	x	0	0	0	0
S_4	M_4	${r_p}^4$	0	0	0	1	1	0	x	x	0	0
S_5	M_5	${r_p}^5$	0	0	0	0	0	1	1	0	x	x
S_6	M_1, M_2	${r_p}^6$	x	x	x	x	0	0	0	1	1	1
S_7	M_1, M_3	${r_p}^7$	x	x	1	0	x	x	0	1	1	0
S_8	M_1, M_4	${r_p}^8$	x	x	0	1	1	0	x	x	1	0
S_9	M_1, M_5	${r_p}^9$	x	x	0	0	0	1	1	1	x	x
S_{10}	M_2, M_3	${r_p}^{10}$	1	1	x	x	x	x	0	0	0	1
S_{11}	M_2, M_4	${r_p}^{11}$	1	0	x	x	1	0	x	x	0	1
S_{12}	M_2, M_5	${r_p}^{12}$	1	0	x	x	0	1	1	0	x	x
S_{13}	M_3, M_4	${r_p}^{13}$	0	1	1	1	x	x	x	x	0	0
S_{14}	M_3, M_5	${r_p}^{14}$	0	1	1	0	x	x	1	0	x	x
S_{15}	M_4, M_5	${r_p}^{15}$	0	0	0	1	1	1	x	x	x	x

Tabulka 1.2.: Tabulka potenciálních syndromů

povídá syndromu reálnému, což umožní identifikovat chybné moduly (jsou uvedeny v druhém sloupci tabulky). Při porovnání se musí shodovat všechny výsledky atomických kontrol, nejednoznačný výsledek u potenciálního syndromu (označený jako „x“) se shoduje s libovolným výsledkem reálné kontroly (žolíkové porovnávání).

Pro porovnávání existuje několik strategií. Základní a nejjednodušší strategie, jež spočívá v postupném porovnávání reálného syndromu s jednotlivými řádky tabulky, je neefektivní, neboť vyžaduje největší počet porovnání (maximálně až $Q \cdot l$).

V našem ukázkovém případě, v němž je reálný syndrom roven $R_A = \{r_{12}=0, r_{13}=1, r_{23}=1, r_{24}=0, r_{34}=1, r_{35}=0, , r_{45}=1, , r_{41}=0, , r_{51}=1, r_{52}=1\}$, lze i při použití základní strategie snadno nalézt shodující se potenciální syndrom ${r_p}^{14}$ a tím diagnostikovat stav modulů v systému (chybné jsou moduly M_3, M_5, správné M_1, M_2, M_4).

I při ručním prohledávání tabulky se však jako výhodnější jeví postupný předvýběr řádků pomocí několika prvních hodnot syndromu (například, pokud zohledníme jen první prvek syndromu, omezí se výběr na 11 potenciálních řádků).

Tento algoritmus předvýběrů lze snadno rozšířit a implementovat. Nejdříve jsou vybrány řádky, u nichž se shoduje první prvek s reálným syndromem (jsou to řádky 1, 3, 4, 5, 6, 7, 8, 9, 13, 14, 15). V druhém kroku se zaměříme jen na vybrané řádky a testujeme shodu u druhého prvku syndromu. Výběr se opět omezí na řádky (1, 3, 6, 7, 8, 9, 13, 14). Postupný výběr pokračuje a končí v sedmém kroku, v němž se rozhodne mezi variantami potenciálních syndromů ${r_p}^3$ a ${r_p}^{14}$. Počet porovnání je v tomto případě výrazně menší[2].

Na závěr této sekce shrneme některé přednosti a nevýhody tabulkových algoritmů plynoucí z jejich návrhu i použití:

výhody:

- potřebují pouze základní informace o systému (ty máme zpravidla vždy k dispozici)
- tabulky mohou být snadno vytvořeny a jsou názorné, což snižuje chybovost při jejich zpracování

nevýhody:

- tabulkové algoritmy nezohledňují spolehlivost jednotlivých modulů systému, což snižuje důvěryhodnost výsledku

Pravděpodobnostní algoritmy

Pravděpodobnostní algoritmy samodiagnostiky jsou zaměřeny na výpočet aposteriorní pravděpodobností stavů jednotlivých modulů systému. Po zjištění daných pravděpodobností může být učiněno rozhodnutí buď o stavech všech modulů v systému, nebo alespoň o některých modulech (v případě nedostatku informací). Důvěryhodnost výsledků může být navíc zvýšena zohledněním předběžných informacích o spolehlivosti jednotlivých modulů. Z tohoto důvodu nejsou pravděpodobnostní algoritmy omezeny jen na t-diagnostikovatelné grafy s maximálně t chybnými moduly, ale mohou poskytovat relevantní informace i při nižším počtu atomických kontrol, nebo při větším počtu chybných modulů.

K návrhu pravděpodobnostních algoritmů přispěli především H. Fujiwara a K. Kinoshita, kteří již v roce 1981 navrhli jednoduchý a efektivní algoritmus [7].

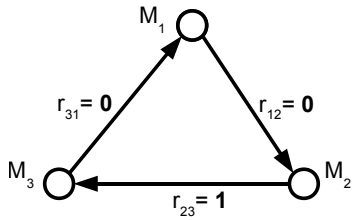
Nejdříve si připomeňme, že pro zjištění stavu modulu můžeme použít jak *apriorní* tak *aposteriorní* pravděpodobnost. *Apriori* znamená doslova „před“. Proto přívlastek **apriorní** vyjadřuje pravděpodobnost určitého stavu modulu ještě před provedením atomických kontrol. Tato pravděpodobnost je ve většině případů stanovena

[2] to však ještě neznamená, že je vždy výhodnější, při použití proudových bitových operací v paralelních systémech mohou být výhodnější některé varianty přímého porovnávání.

na základě dodatečných informací, například informace o spolehlivosti modulu. Tato informace bývá uváděna v dokumentaci k modulu. Běžně to bývá např. parametr λ, tj. intenzita exponenciálního rozdělení selhání. Apriorní pravděpodobnosti se mohou u jednotlivých modulů lišit. Tím se pravděpodobnostní přístup k diagnostice liší od přístupu tabulkového, který vychází z předpokladu, že všechny moduly mají stejnou (apriorní) pravděpodobnost selhání. Zahrnutím specifického chování jednotlivých modulů se může zvýšit důvěryhodnost diagnostiky, neboť ta již například nemusí být omezena hodnotou t.

Aposteriorní pravděpodobnost naproti tomu vyjadřuje pravděpodobnost určitého stavu modulu po provedení jeho kontroly. Je zřejmé, že aposteriorní pravděpodobnost správného stavu u správného modulu je vyšší než apriorní, neboť výsledky kontroly přidávají další informace o stavu modulů, čímž zvyšují naši jistotu o stavu systému.

Základem algoritmu je výpočet aposteriorní pravděpodobnosti jednotlivých stavů (správný/chybný) u všech modulů v systému. Pro tento účel využijeme jednoduchý příklad diagnostického grafu s třemi moduly (M_1, M_2, M_3) a třemi atomickými kontrolami. Diagnostický graf je znázorněn na obrázku 1.19.



Obrázek 1.19.: Ukázkový DG a syndrom pro popis pravděpodobnostního algoritmu

Výpočet aposteriorních pravděpodobností stavů modulů začíná výpočtem aposteriorní pravděpodobností hypotéz všech možných stavů modulů, který již byl uveden v kapitole 1.4. Zde je pouze poněkud zjednodušen model apriorních pravděpodobností a tím zkrácena symbolika.

Předpokládejme, že apriorní pravděpodobnosti bezchybného stavu modulů jsou známé a nabývají hodnoty P_1,P_2 a P_3. Symboly q_1, q_2 a q_3 označují apriorní pravděpodobnost, že moduly jsou v chybném stavu. Je zřejmé, že $q_1 = 1-P_1$, $q_2 = 1-P_2$ a $q_3 = 1-P_3$. Dále předpokládejme syndrom podle obrázku 1.19.

1. určení všech možných hypotéz ohledně stavu modulů. V našem případě se všechny moduly mohou nacházet jak ve stavu správném tak chybném. Je tak nutno uvažovat osm hypotéz:

H_1:	123	M_1 je správný	M_2 je správný	M_3 je správný
H_2:	$12\overline{3}$	M_1 je správný	M_2 je správný	M_3 je chybný
H_3:	$1\overline{2}3$	M_1 je správný	M_2 je chybný	M_3 je správný
H_4:	$1\overline{23}$	M_1 je správný	M_2 je chybný	M_3 je chybný
H_5:	$\overline{1}23$	M_1 je chybný	M_2 je správný	M_3 je správný
H_6:	$\overline{1}2\overline{3}$	M_1 je chybný	M_2 je správný	M_3 je chybný
H_7:	$\overline{12}3$	M_1 je chybný	M_2 je chybný	M_3 je správný
H_8:	$\overline{123}$	M_1 je chybný	M_2 je chybný	M_3 je chybný

2. výpočet pravděpodobnosti všech hypotéz

$P(H_1) = P_1 P_2 P_3$
$P(H_2) = P_1 P_2 q_3$
$P(H_3) = P_1 q_2 P_3$
$P(H_4) = P_1 q_2 q_3$
$P(H_5) = q_1 P_2 P_3$
$P(H_6) = q_1 P_2 q_3$
$P(H_7) = q_1 q_2 P_3$
$P(H_8) = q_1 q_2 q_3$

Protože hypotézy $H_1 \dots H_8$ popisují všechny možné situace, je jejich celková pravděpodobnost rovna jedné.

3. určení podmíněných pravděpodobností

Výpočet provádíme pro událost, v níž atomické kontroly $\tau_{12}, \tau_{23}, \tau_{31}$ skončí s výsledkem (syndromem) podle obrázku 1.19. Proto musíme nejdříve vypočítat pravděpodobnost získání tohoto syndromu za podmínky, že stavy modulů odpovídají určité hypotéze. Jednotlivé podmíněné pravděpodobnosti i zde závisí na representaci výsledků atomických kontrol. Pokud využijeme klasickou representaci Preparatovu a navíc budeme předpokládat, že pravděpodobnost jedničkového výsledku je vždy $P_r = P\{X = 1\}$ (tj. výsledek kontroly prováděné chybným modulem nezávisí na stavu kontrolovaného modulu[3]), získáme následující pravděpodobnosti:

$P(R/H_1) = 0$ $\qquad$ $P(R/H_5) = 0$
$P(R/H_2) = 1 - P_r$ $\qquad$ $P(R/H_6) = (1 - P_r)^2$
$P(R/H_3) = 0$ $\qquad$ $P(R/H_7) = 0$
$P(R/H_4) = 0$ $\qquad$ $P(R/H_1) = (1 - P_r)^2 P_r$

[3] v definici ohodnocení diagnostického grafu (kapitola 1.4) byl užit složitější model. Pravděpodobnosti byly závislé na stavu kontrolovaného modulu: při kontrole správného = P_A a při kontrole chybného = P_B . Zde jednoduše platí $P_r = P\{X = 1\} = P_A = P_B$.

Nulové pravděpodobnosti jsou u hypotetických stavů, které nemohou daný syndrom produkovat[4]. Například, pokud by byly všechny moduly správné (hypotéza H_1), pak by nemohl správný modul M_2 označit za chybný modul M_3 (výsledek atomické kontroly r_{23} je jedna). Hypotéza H_6 je naproti tomu slučitelná se syndromem a pravděpodobnost je součinem tří nezávislých pravděpodobností: $P(R : r_{12} = 0/H_6)$, což je pravděpodobnost, že chybný modul M_1 (viz hypotéza) provede kontrolu modulu M_2 s výsledkem „1" $= (1 - P_r)$; $P(R : r_{23} = 1/H_6) = 1$, neboť správný modul vždy odhalí chybný (podle definice 1.1 [5]); a nakonec $P(R : r_{31} = 0/H_6) = (1 - P_r)$ ze stejných důvodů jako výše (hypotéza předpokládá, že M_3 je chybný). Pravděpodobnosti hypotéz H_2 a H_8 lze vyjádřit obdobným způsobem.

4. určení podmíněné pravděpodobnosti hypotéz při daném syndromu

Pro výpočet jednotlivých podmíněných pravděpodobností $P(H_i/R)$ lze využít Bayesův vztah 1.9 na straně 24. Jmenovatel zlomku vyjadřuje pravděpodobnost syndromu R při daných apriorních pravděpodobnostech, tj. $P(R) = \sum_{i=1}^{\ell} P(H_i)P(R/H_i)$. Pro náš příklad je $P(R)$ rovno $(1-P_r)P_1P_2q_3 + (1-P_r)^2q_1P_2q_3 + (1-P_r)^2P_rq_1q_2q_3$.

Pravděpodobnosti jednotlivých hypotéz za podmínky získání daného syndromu mají následující tvar (nulové jsou vynechány):

$$P(H_2/R) = \frac{(1-P_r)P_1P_2q_3}{P(R)}, \quad P(H_6/R) = \frac{(1-P_r)^2q_1P_2q_3}{P(R)},$$

$$P(H_8/R) = \frac{(1-P_r)^2P_rq_1q_2q_3}{P(R)}$$

5. určení aposteriorní pravděpodobnosti správnosti jednotlivých modulů

Aposteriorní pravděpodobnost, že je určitý modul správný, lze získat z podmíněných pravděpodobností jednotlivých hypotéz. Tato pravděpodobnost je totiž rovna podmíněné pravděpodobnosti události, v níž stav systému odpovídá libovolné hypotéze, která daný modul považuje za správný. Například v našem případě je modul M_2 považován za správný v hypotézách H_1, H_2, H_5 a H_6 .

Podmíněnou pravděpodobnost sjednocení hypotéz lze získat součtem podmíněných pravděpodobností těchto hypotéz (podmínka je ve všech případech stejná, po provedení atomické kontroly je získán určitý syndrom). Když aposteriorní pravděpodobnost správnosti modulu M_i označíme symbolem P_i^*, pak můžeme napsat:

[4] nulové pravděpodobnosti jsou možné, neboť v použitém (zjednodušeném) modelu je $P_{AT} = 1$ (správný modul vždy odhalí chybný). V realitě se nulové hodnotě pouze blíží.

[5] při hodnocení DG jsme používali obecnější předpoklad, že tato pravděpodobnost nemusí být jedna, ale obecně P_{AT}, viz tabulka 1.1 na straně 23

$$P_i^* = \sum_{H \in U} P(H/R), \text{ kde } U = \{H_j : M \text{ je správné v } H_j\} \tag{1.11}$$

V našem případě má pro modul M_2 vztah tuto konkrétní podobu:

$$P_2^* = P(H_1/R) + P(H_2/R) + P(H_5/R) + P(H_6/R) = \frac{(1-P_r)P_1P_2q_3 + (1-P_r)^2q_1P_2q_3}{(1-P_r)P_1P_2q_3 + (1-P_r)^2q_1P_2q_3 + (1-P_r)^2P_rq_1q_2q_3}$$

Pro názornost můžeme aposteriorní pravděpodobnost vyčíslit pro konkrétní hodnoty apriorních pravděpodobností například pro $P_1 = P_2 = 0.8$. Dále předpokládejme, že pravděpodobnost P_r je rovna 0.5 (chybný modul vrací při kontrole ostatních modulů se stejnou pravděpodobností buď hodnotu „0“ nebo „1“).

Podle 1.11 se aposteriorní pravděpodobnost správnosti modulu M_2 rovná 0.986.

Jak lze vidět, po provedení trojice atomických kontrol se zvýšila naše jistota o správnosti modulu z 0.8 (apriorní pravděpodobnost) na téměř 0.99, neboť atomické kontroly byly v souladu s apriorním předpokladem.

Dílčí pravděpodobnosti a výsledky pro všechny moduly ukazuje obrázek 1.20 na následující straně (získaný z tabulkového kalkulátoru *OpenOffice Calc*).

Výpočet aposteriorních pravděpodobností správnosti jednotlivých modulů je však pouze podkladem vlastní diagnostiky. Hlavním cílem je stejně jako u výše uvedených diagnostických algoritmů identifikace správných i chybných modulů.

V některých případech je identifikace správných a chybných modulů zřejmá. Například v uvažovaném případě je aposteriorní pravděpodobnost správnosti modulů M_1 a M_2 vysoká a větší než pravděpodobnost apriorní. Naopak aposteriorní pravděpodobnost správnosti modulu M_3 je nulová. Je tedy zřejmé, které moduly jsou správné a které chybné.

Jako cvičení pro ověření nabytých znalostí nabízíme čtenáři složitější systém s pěti moduly, jehož graf je uveden na obrázku 1.21 na následující straně. Pokud předpokládáme apriorní pravděpodobnost $P_i = 0.8$ ($i = 1 \ldots 5$) a $P_r = 0.5$, pak jsou aposteriorní pravděpodobnosti rovny:

$P_1^* = 0.883, \quad P_2^* = 0.939, \quad P_3^* = 0.941, \quad P_4^* = 0.055, \quad P_5^* = 0.059$

I zde je rozdělení zřejmé, neboť moduly se rozdělují do dvou oddělených skupin. Moduly M_1, M_2, M_3 jsou s vysokou jistotou správné, moduly M_4 a M_5 nesprávné.

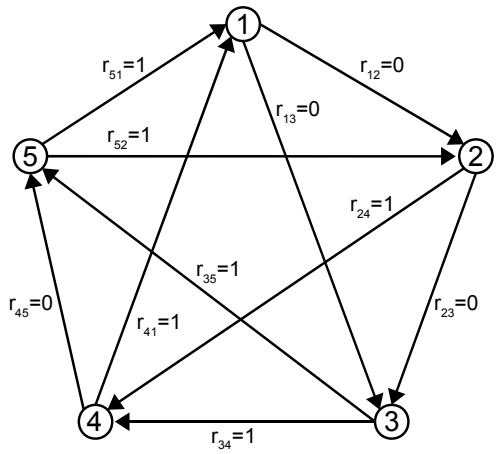
P_r	0,50
$1-P_r$	0,50

	P_i	q_i	P_i^*
M_1	0,80	0,20	**0,877**
M_2	0,80	0,20	**0,986**
M_3	0,80	0,20	**0,000**

		$P(H_i)$	$P(R/H_i)$	$P(H_i/R)$
H_1	123	0,512	0,000	0,000
H_2	12**3**	0,128	0,500	0,877
H_3	1**2**3	0,128	0,000	0,000
H_4	1**23**	0,032	0,000	0,000
H_5	**1**23	0,128	0,000	0,000
H_6	**12**3	0,032	0,250	0,110
H_7	**12**3	0,032	0,000	0,000
H_8	**123**	0,008	0,125	0,014

P(R)	0,073

Obrázek 1.20.: Výpočet aposteriorní pravděpodobnosti správnosti modulů



$M_R =$

1	2	3	4	5	
-	0	0	-	-	1
-	-	0	1	-	2
-	-	-	1	1	3
1	-	-	-	0	4
1	1	-	-	-	5

Obrázek 1.21.: Cvičný syndrom pro testování pravděpodobnostního algoritmu

Implementace pravděpodobnostních algoritmů musí zohledňovat různé modely ohodnocení atomických kontrol i různé representace diagnostických grafů. Generický model (viz [8]) vytvořený pro prostředí *Sage* (= *Python* rozšířený o velké množství matematicky orientovaných knihoven a externích nástrojů) nabízí dostatečnou flexibilitu. Jeho jádrem je objektový model systému a získaného syndromu.

```
from itertools import product, izip, repeat
import probability
from fractions import Fraction

class Module:   #system module
  def __init__(self, apriori, name = None):
    self.apriori = apriori   #apriori probability
    self.system = None
    self.index = None

  def setSystem(self, system):
    self.system = system

class System:
  def __init__(self):
    self.modules = []

  def __getitem__(self, index):   #read only index operator
    return self.modules[index]

  def addModule(self, module):
    module.setSystem(self)
    self.modules.append(module)
    index = self.size() - 1       #module index: n + 1
    module.index = index
    return index

  def size(self):
    return len(self.modules)

  def moduleIterator(self):
    return iter(self.modules)

  @staticmethod
  def fromAprioriProbabilities(apriIterator):   #auxiliary constructor
    system = System()
    for apriProbability in apriIterator:
        module = Module(apriProbability)
        system.addModule(module)
    return system

class Ec:   #elementary check
  def __init__(self, checking, checked, result):
    self.checkingModule = checking #Mi
    self.checkedModule = checked   #Mj
    self.result = result

class Syndrome: #base abstract class for syndrome
  def __init__(self):
    raise NotImplementedError("Abstract class")
```

```
51    def addEc(self, checkingModule, checkedModule, result):
52      raise NotImplementedError("Abstract class")
53    def ecResult(self, checkingModule, checkedModule):
54      return None
55    def ecIterator(self):
56      raise NotImplementedError("Abstract class")
57
58  class SparseSyndrome(Syndrome): #sparse representation of syndrome
59    def __init__(self, system):
60      self.ecs = dict()            #hash table
61      self.system = system
62
63    def addEc(self, ec):
64      assert ec.result in [0,1], "Undefined result"
65      self.ecs[(ec.checkingModule, ec.checkedModule)] = ec.result
66
67    def ecResult(self, checkingModule, checkedModule):
68      if (checkingModule, checkedModule) in self.ecs:
69        return self.ecs[(checkingModule, checkedModule)]
70      else:
71        return super.ecResult(checkingModule, checkedModule)
72
73    def ecIterator(self):  #generator of syndrome sequence
74      for checking,checked in self.ecs.iterkeys():
75        yield Ec(checking, checked, self.ecResult(checking,checked))
76
77    def conditionalProbabilityForHypothesis(self, hypothesis, ecProbabilityFn):
78      assert (self.system == hypothesis.system,
79          "The system of syndrom and system of hypothesis are different")
80      return probability.joint( #P(R/Hn)
81      ecProbabilityFn(hypothesis.moduleState(ec.checkingModule),
82                      hypothesis.moduleState(ec.checkedModule), ec.result)
83          for ec in self.ecIterator()
84      )
85
86    @staticmethod
87    def fromSyndromeTable(system, table, nonEc = None): #auxiliary constructor
88      syndrome = SparseSyndrome(system)
89      for i,line in enumerate(table):
90        for j,item in enumerate(line):
91          if item != nonEc:
92            syndrome.addEc(Ec(i, j, table[i][j]))
93      return syndrome
94
95  class Hypothesis: #representation of hypothesis
96    def __init__(self, system, moduleStates):
97      self.system = system
98      self.moduleStates = moduleStates
99
100   @staticmethod
101   def IteratorOverAllHypothesis(system): #lazy iterator over hypothesis
102     for combination in product([True, False], repeat=system.size()):
103       yield Hypothesis(system, combination)
104
105   def __str__(self):
106     return str(self.moduleStates)
107
```

```
108   def aprioriProbability(self): #P(Hn)
109     return probability.joint(
110       probability.ifElseComplement(state, module.apriori)
111         for module, state in izip(self.system.moduleIterator(),
112                                   self.moduleStates)
113       )
114
115   def moduleState(self, module):
116     return self.moduleStates[module]
117
118
119 def PreparatECPFGenerator(pr): #return closure for Preparata evaluation function
120   def PreparatECPF(checkingOK, checkedOK, result):
121     if (checkingOK):
122       return probability.certainIf(checkedOK and result == 0
123                                    or not checkedOK and result == 1)
124     else:
125       return probability.ifElseComplement(result == 1, pr)
126   return PreparatECPF
127
128 def aposterioriMapping(hypothesis, syndrom, ecProbabilityFn):
129   tp = (hypothesis.aprioriProbability() *
130     syndrom.conditionalProbabilityForHypothesis(hypothesis, ecProbabilityFn))
131   mps = [probability.onlyIf(state, tp) for state in hypothesis.moduleStates]
132   return (tp, mps)
133
134
135 def aposterioriPriorityOfModules(system, syndrom, ecProbabilityFunction):
136   sumtp,summps = reduce(
137     lambda p1, p2: (p1[0] + p2[0], [ x + y for x, y in zip(p1[1], p2[1])]),
138     (aposterioriMapping(hypothesis, syndrom, ecProbabilityFunction)
139         for hypothesis in Hypothesis.IteratorOverAllHypothesis(system))
140     )
141   return [ summp / sumtp for summp in summps]
```

Klíčové jsou především dvě funkce v samém závěru výpisu, které prostřednictvím seznamových komprehenzí iterují přes lenivý iterátor hypotéz a počítají hodnoty $P(R/H_i)$. Určitou komplikací jsou zde pouze výpočty všech dílčích hodnot v jediném průchodu cyklu, což je nezbytné z důvodů efektivity.

Výsledný seznam aposteriorních pravděpodobností lze využít i pro další výpočty, a to jak numerické tak symbolické (*Sage* podporuje i symbolické výpočty), resp. pro grafické znázornění funkčních závislostí.

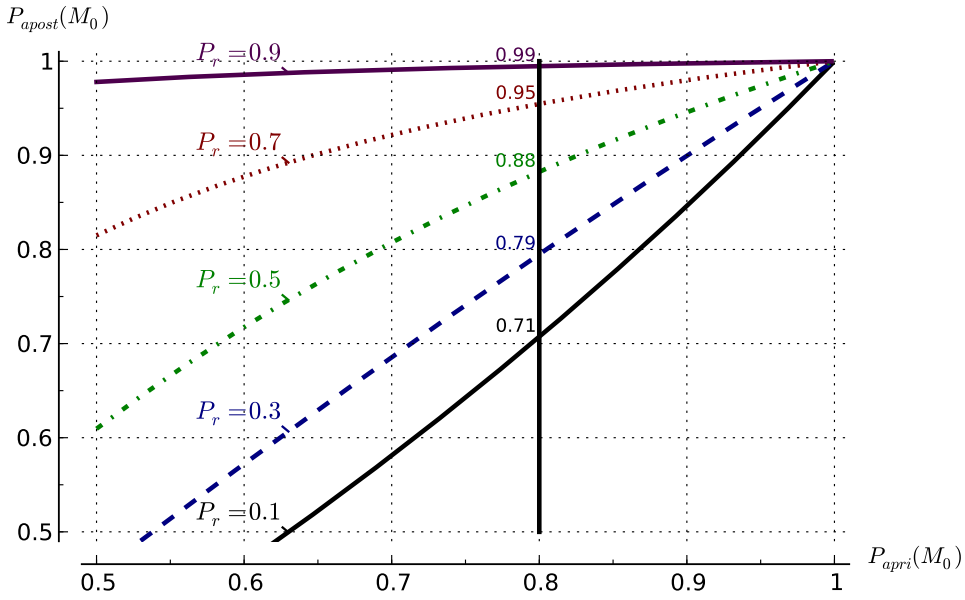
Jako příklad lze uvést program vytvářející graf funkční závislosti aposteriorních pravděpodobností správnosti modulu M_0 (viz DG na obrázku 1.21 na straně 45) na pravděpodobnosti apriorní (osa x) a na hodnotě pravděpodobnosti P_r. Jedinou netriviální částí programu je třída *TestSystem*, která váže jednotlivé pevné parametry modelu (systém modulů a model ohodnocení AT) a poskytuje funkci (přesněji uzávěr), jenž je parametrizován pouze apriorní prioritou, a lze jej tudíž použít pro vykreslení 2D grafu.

```
from aposteriori import *
from graph import Graph, Function

syndromTable = [
      [None,    0,    0, None, None],
      [None, None,    0,    1, None],
      [None, None, None,    1,    1],
      [   1, None, None, None,    0],
      [   1,    1, None, None, None]
    ]


class TestSystem:
    def __init__(self, PreparatP, syndromeTable):
        self.PreparatP = PreparatP
        self.syndromeTable = syndromeTable

    def getAposterioriP(self, apriori):
        system = System.fromAprioriProbabilities(repeat(apriori, 5))
        syndrome = SparseSyndrome.fromSyndromeTable(system, self.syndromeTable)
        return aposterioriPriorityOfModules(system, syndrome,
                                            PreparatECPFGenerator(self.PreparatP))[0]


g = Graph(0.5, 1.0, ymin=0.5, ymax = 1.0, x="P_{apri}(M_0)",
                    y="P_{apost}(M_0)", notesize=1.1)

for pr in [0.1, 0.3, 0.5, 0.7, 0.9]:
    s = TestSystem(pr, syndromTable)
    g.add_function(Function(s.getAposterioriP, "P_r=%.1g" % pr, width=2),
                   xnote=0.63
                  )

g.add_cut(0.8)

g.show("aposteriori")
```

Výsledný graf je presentován na obrázku 1.22 na následující straně.

Zajímavá situace nastane v případě, pokud obdrženému syndromu R nebude odpovídat žádná hypotéza, tj. pokud budou všechny aposteriorní pravděpodobnosti nulové. V tomto případě se evidentně jedná o konfliktní situaci, jejíž příčinou je neadekvátní předpoklad o režimu selhání, tj. předpoklad neodpovídající skutečnosti.

Prozatím jsme předpokládali, že všechna selhání chybných modulů jsou permanentní. To však neplatí u tzv. intermitentních selhání, u nichž se chybný stav modulu projevuje navenek jen občas, tj. modul někdy selhává (atomické kontroly jej odhalí jako chybný), v jiných okamžicích však nikoliv (atomické kontroly jej označí za správný). Podrobnější popis intermitentních selhání viz kapitola 1.9 na straně 55.



Obrázek 1.22.: Ukázkový výstup modelu pro testovací syndrom

1.7. t/(n-1)-diagnostika

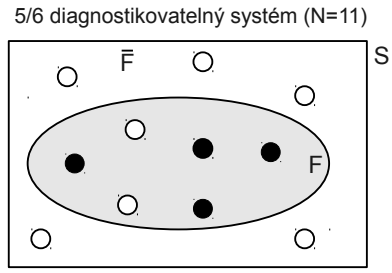
Hlavním cílem diagnostiky v předchozích sekcích byla identifikace stavu všech modulů v systému. Tento cíl však může být příliš ambiciozní. Pro jeho dosažení je nutné provést relativně velký počet atomických kontrol a především paměťově i časově náročnou diagnostiku. Počet atomických kontrol roste u t_{max}-diagnostikovatelných systémů oproti počtu uzlů N kvadraticky (viz vztah 1.5 na straně 15) a podobná je i výpočetní náročnost algoritmů ($O(N^2)$). Jednotlivé moduly jsou však ve většině složitých systémů velmi jednoduché. Tyto moduly nejsou schopny provádět komplexnější algoritmy, a pokud ano, tak v neakceptovatelném čase.

V některých případech však postačuje pouze částečná identifikace stavů modulů, kterou lze specifikovat jako:

1. zjištění omezeného počtu chybných, resp. správných modulů (často postačuje nalezení jediného)
2. nalezení podmnožiny modulů, která obsahuje všechny chybné moduly.

Například Friedmann v článku [5] definoval *t/s-diagnostikovatelné systémy*, tj. systémy, v nichž lze při diagnostice najít takovou s-prvkovou množinu F, ve které jsou obsaženy všechny chybné moduly ($s < N$). Tato množina však může obsahovat i

moduly správné. Konkrétní stav jednotlivých modulů v množině F nelze po provedení diagnostiky jednoznačně určit. U doplňku množiny F (o velikosti $N-s$) je situace jiná, neboť je zřejmé, že obsahuje pouze správné moduly. I když je tedy diagnostika u t/s-diagnostikovatelných systémů pouze částečná, je výsledkem konkrétní identifikace alespoň $(N-s)$ správných modulů. Lze dokázat, že diagnostické grafy zaručující *t/s-diagnostikovatelnost* mohou obsahovat menší počet atomických kontrol, než je tomu u t-diagnostikovatelných systémů.



Obrázek 1.23.: t/s-diagnostika

Speciálním případem t/s-diagnostikovatelných systémů jsou **t/(n-1) diagnostikovatelné systémy**, u nichž $s = n-1$ (viz Xu [6]).

DEFINICE 1.6:

Systém S je **t/(n-1)-diagnostikovatelný** (na základě získaného syndromu) právě tehdy, když všechny chybné moduly mohou být izolovány uvnitř množiny F s velikostí nanejvýš $(N-1)$ modulů pod podmínkou že počet závadných modulů nepřekračuje hodnotu t.

□

Hlavním výsledkem diagnostiky u těchto systémů je identifikace alespoň jednoho správného modulu (doplněk $\overline{F}$ je v tomto případě jednoprvkový). Počet nutných atomických kontrol je však relativně malý. Lze dokázat, že v případě malých systémů ($N \leq 10$) roste počet atomických kontrol lineárně vzhledem k N. Lineární složitost má i diagnostický algoritmus.

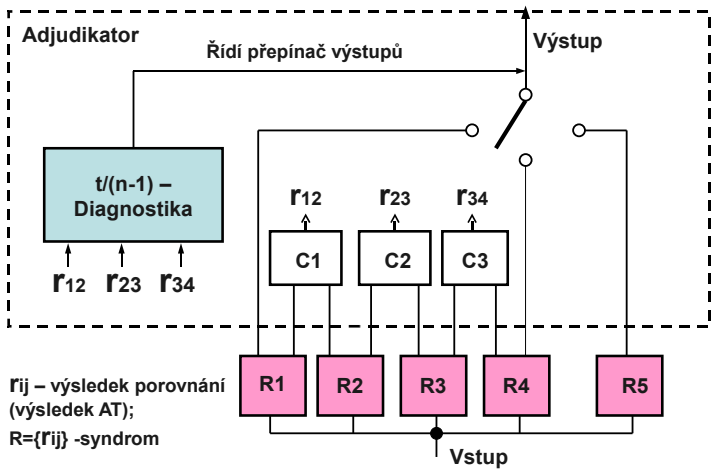
Konkrétně například u systému se třemi moduly ($t_{max} = 1$) postačuje jediná atomická kontrola . U systému s pěti moduly ($t_{max} = 2$) stačí jen tři atomické kontroly oproti deseti AT u t-diagnostiky.

Diagnostika t/(n-1)-diagnostikovatelných systémů je využívána i u systémů zajišťujících odolnost proti softwarovým závadám. U žádného komplexnějšího programového kódu nelze zaručit, že chyby, které zůstaly po ladění, nezpůsobí jeho selhání, např. celkovou nefunkčnost. Odolnost softwarového systému však lze zvý-

šit vykonáním několika nezávislých implementací (resp. variant) jednotlivých rutin. Tyto rutiny by v případě bezchybné implementace měly vracet stejné výsledné hodnoty. Realita však může být jiná, neboť výsledky jednotlivých implementací se mohou lišit (resp. některé implementace nemusí výpočet vůbec dokončit). Systém pro zajištění odolnosti proti závadám však může ve většině případů rozhodnout, který z variantních výsledků je správný, a to i v případě, že nemá žádné dodatečné informace (samozřejmě jen za předpokladu, že chybných rutin není příliš mnoho). Jedním z možných řešení je použití principů samodiagnostiky. Roli modulů hrají v tomto případě jednotlivé softwarové rutiny, atomické kontroly jsou representovány porovnáním výsledků dvou rutin. Vnějším pozorovatelem je samotný systém pro zajištění odolnosti proti závadám. Tento pozorovatel je v tomto případě reálný a je označován jako adjudikátor.

Cílem je nalezení právě jednoho správného modulu, tj. zde rutiny, která poskytne správný výsledek. I zde se předpokládá, že počet chybných rutin je roven nejvýše t_{max}. Je zřejmé, že se lze v tomto případě omezit na t/(n-1)-diagnostiku, neboť stačí identifikovat pouze jedinou bezchybnou rutinu, a tím identifikovat správný výsledek. Navíc stačí k tomuto účelu provést jen řádově jednotky porovnání, neboť počet variantních rutin není v praxi příliš velký.

Obrázek 1.24 ukazuje systém, v němž existuje pět variantních implementací R_i jedné softwarové rutiny. Může to být například rutina pro získání údajů z databáze (každá varianta pracuje nad jinou databází), nebo rutina pro výpočet hodnoty simulované fyzikální veličiny. Každá rutina by měla používat jiný algoritmus nebo být alespoň vytvořena jiným programátorem.



Obrázek 1.24.: t/(n-1) adjudikátor

Jádrem adjudikátoru je jednoduchý algoritmus, který musí na základě porovnání identifikovat rutinu poskytující správný výsledek, a to za předpokladu, že alespoň tři rutiny takový výsledek poskytnou, neboť $t_{max} = 2$. Adjudikátor používá t/(n-1)-diagnostiku (v daném případě 2/4-diagnostiku), tj. postačují tři porovnávače, z nichž každý porovnává výsledky dvou rutin. Porovnávače (C_i) produkují syndrom R = {r_{12}, r_{23}, r_{34}}.

Adjudikátor následně za použití elementární logiky určí, která rutina poskytne konečný výsledek. Navíc nemusí vybírat z výsledků všech rutin, ale stačí pouze z předem určitelné množiny k rutin, kde k je rovno $N-t$. To usnadní návrh adjudikátoru, neboť přepínač výstupů (softwarový nebo hardwarový) může být jednodušší.

Algoritmus správného modulu lze popsat pomocí tabulky. Pro náš konkrétní případ je to tabulka 1.3. Množinu správných rutin pro každý syndrom lze získat na základě jeho rozboru při zohlednění předpokladu, že přípustné jsou pouze dva chybné moduly. Z tabulky lze navíc určit i k-prvkovou množinu modulů, z níž může být vybrán poskytovatel správného výsledku. V daném příkladě to jsou moduly R_1, R_4 a R_5 (v tabulce označeny tučně, lze tudíž snadno vidět, že alespoň jeden modul z této množiny je označen jako správný pro libovolný syndrom).

r_{12}	r_{23}	r_{34}	správné rutiny
0	0	0	R_1,R_2, R_3,$\mathbf{R_4}$
0	0	1	$\mathbf{R_1}$, R_2, R_3
0	1	0	$\mathbf{R_5}$
0	1	1	$\mathbf{R_1}$, R_2
1	0	0	R_2,R_3,$\mathbf{R_4}$
1	0	1	$\mathbf{R_5}$
1	1	0	R_3,$\mathbf{R_4}$
1	1	1	$\mathbf{R_5}$

Tabulka 1.3.: Tabulka algoritmu t/(n-1) diagnostiky

1.8. Sekvenční diagnostika

Provedení diagnostiky (tj. dekódování syndromu) může obecně poskytovat následující výsledky:

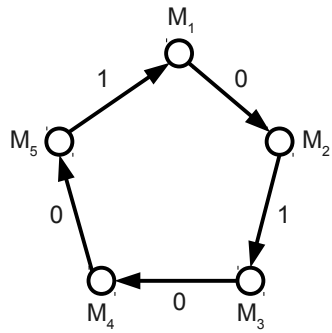
1. všechny chybné moduly jsou správně identifikovány (lokalizovány) – úplná diagnostika (t-diagnostika)

2. je identifikována podmnožina modulů, která zaručeně obsahuje všechny chybné moduly – omezená diagnostika
3. je identifikován alespoň jeden modul - částečná diagnostika.

První dva případy diagnostiky jsou typické pro tzv. jednokrokové, resp. paralelní diagnostiky. Třetí případ je základem tzv. sekvenční diagnostiky, která pro identifikaci všech chybných modulů vyžaduje několik postupných kroků. V rámci každého kroku je identifikován alespoň jeden chybný modul a tento modul je následně odstraněn ze systému. Z tohoto důvodů se sekvenční diagnostika označuje jako diagnostika s opravou [9]. Po odstranění chybného modulu se provádí další krok, v rámci něhož jsou provedeny další atomické kontroly v redukovaném systému a je získán nový syndrom.

Je zřejmé, že identifikace jediného chybného modulu vyžaduje méně rozsáhlý syndrom, a tudíž i menší počet provedených atomických kontrol. Proto je struktura atomických kontrol včetně jejich celkového počtu v každém postupném kroku výrazně zredukována.

Uvažujme například systém, jehož diagnostický graf je znázorněn na obrázku 1.25.



Obrázek 1.25.: Diagnostický graf sekvenčně 2-diagnostikovatelného systému

V případě provedení jednokrokové diagnostiky se počet chybných modulů, při kterém je zaručena jejich přesná identifikace, rovná 1 (to jest parametru t). Malá hodnota parametru t odpovídá velmi jednoduché struktuře atomických kontrol (pouze 5 atomických kontrol).

V případě použití vícekrokové sekvenční diagnostiky umožňuje systém přesně identifikovat až dva chybné moduly. To znamená, že systém, jehož diagnostický graf je znázorněn na obr. 1.25, je sekvenčně 2-diagnostikovatelný. Při identifikaci je však nutno provést dva kroky, přičemž v každém kroku je odhalen jeden chybný modul.

Nechť je pro uvažovaný systém získám syndrom uvedený na obrázku 1.25, tj. $S = \{r_{12} = 0, r_{23} = 1, r_{34} = 0, r_{45} = 0, r_{51} = 1\}$. Za předpokladu, že je využívána represen-

tace podle definice 1.1, musí být modul M_1 označen za chybný. V opačném případě by musely být chybné moduly M_3 a M_5 a také jeden z modulů M_2 a M_3, tj. celkem tři. To je v rozporu s předpokladem o maximálně $t_{max} = 2$ chybných modulech. Je nutné zdůraznit, že získaný syndrom neumožňuje odhalit zbývající chybný modul. Podezřelé jsou moduly M_2 a M_3, neboť situace, kdy jsou chybné moduly M_1a M_2 může produkovat stejný symbol jako situace, kdy jsou chybné moduly M_1a M_3 (ověřte).

Pro upřesnění tak musí být proveden druhý krok. Předtím (tj. po ukončení prvního kroku) je ze systému odstraněn modul M_1. V druhém kroku stačí provést pouze jedinou atomickou kontrolu a to buď modulu M_2 nebo M_3, například τ_{52}. Lze vidět typický rys sekvenční diagnostiky, závislost jednotlivých kroků na krocích předchozích.

Obecné zjištění horní meze parametru t pro zadaný diagnostický graf je v případě sekvenční diagnostiky složitý problém, stejně jako zjištění potřebného počtu kroků. Pozornost se soustřeďuje především na speciální typy grafů, např. na grafy se symetrickou strukturou, tj. regulární grafy, v němž mají všechny uzly stejný vstupní a výstupní stupeň. Pro grafy s kruhovou strukturou lze například parametr t vyčíslit ze vztahu: $\lceil (t^2-1)/4 \rceil + t + 2 \leq N$.

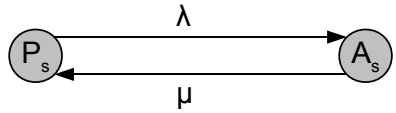
Je k dispozici velké množství článků, které se věnují složitějším regulárním strukturám DG, jako například pravidelným síťovým strukturám (trojúhelníkovým, čtvercovým, šesti- a osmiúhelníkovým) a strukturám hyperkrychlovým [10]. Například S. Khanna a K. Fuchs [11] prokázali, že sekvenční diagnostika může být přesná a kompletní v systému, jehož graf má podobu d-rozměrné čtvercové sítě, jen tehdy, když celkový počet chybných modulů nepřekračuje mez $O(N^{d/(d+1)})$, což je pro libovolný rozměr d lepší než $O(N)$.

Je nutno poznamenat, že sekvenční diagnostika je možná pouze tehdy, pokud systém nevykonává svou základní činnost. Obvykle se tato diagnostika provádí před nasazením systému, kdy čas diagnostiky nehraje velkou roli a jsou k dispozici náhradní moduly. V tomto případě může být v každém kroku algoritmu nahrazen chybný modul správným a diagnostický algoritmus bude opakován.

1.9. Diagnostika intermitentních selhání

Intermitentní selhání modulů má na rozdíl od selhání permanentních, tj. trvalých, občasný resp. přerušovaný charakter. Důvody takového chování mohou být různé a představují specifickou oblast výzkumu. Zde pouze poznamenejme, že jednou z příčin může být například vliv radiace.

Pro naše účely jsou mnohem zajímavější matematické modely intermitentně selhávajících modulů. Jeden z takových modelů byl navržen Mallelem a Massonem [12]. Tento model uvažuje dva stavy intermitentního selhání, a to stav pasivní P_s a stav aktivní A_s. Dále používá dva číselné parametry λ a μ určující přechody mezi těmito stavy (viz obr. 1.26).



Obrázek 1.26.: Model intermitentních selhání

Tento i další modely se používají k modelování a testování metod diagnostiky intermitentních selhání.

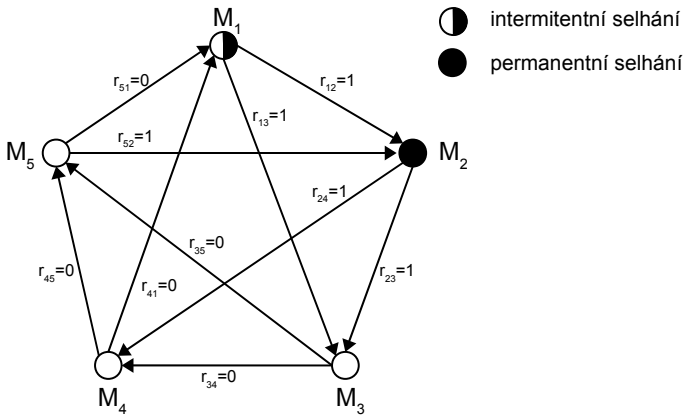
Pro diagnostiku intermitentních selhání je možno použít metody používané u stálých selhání (viz výše), je však nutno uvažovat tři stavy modulů: správný, permanentní selhání a intermitentní selhání. U pravděpodobnostního algoritmu je tak např. nutno uvažovat 3^n hypotéz, což může být výpočetně velmi náročné (a to i v případě malých systémů, neboť např. 3^{10} je řádově rovno 10^{10}). Kromě toho může použití pravděpodobnostních algoritmů vést k situaci, kdy budou mít dvě hypotézy o stavu modulu stejnou resp. podobnou aposteriorní pravděpodobnost, což by vyžadovalo další kritéria pro rozhodnutí. Tato situace může vzniknout především v případě stejných apriorních pravděpodobností u jednotlivých modulů.

V případě tabulkových algoritmů je situace ještě složitější, neboť intermitentní chování porušuje základní předpoklad Preparatovy representace: bezpodmínečné odhalení chybného modulu modulem správným. To může vést k nesprávnému nebo dokonce zcela zmatečnému výsledku diagnostiky. Tato situace je ilustrována pomocí obrázku 1.27, jenž popisuje systém s pěti moduly. V systému předpokládáme jeden modul s permanentním selháním a jeden se selháním intermitentním.

Získaný syndrom je kompatibilní s výše uvedeným předpokladem o stavu modulů. Pokud však máme k dispozici pouze tento syndrom, nelze učinit rozhodnutí, který z modulů M_1 a M_3 je správný a který má intermitentní selhání.

Nalezení obecného intermitentního selhání je velmi obtížné, neboť charakter selhání (vyjádřitelný například pomocí parametrů λ a μ) se může výrazně měnit. Pro některé druhy intermitentních selhání však existují speciální metody, které nalezení a identifikaci chybných modulů výrazně usnadňují.

Navíc je nutno zdůraznit, že v případě intermitentních selhání je důležitá nejen vlastní identifikace selhávajícího modulu, ale i stanovení dalšího postupu pro zacházení s tímto modulem. I zde existují specifické třídy selhání, včetně intermi-



Obrázek 1.27.: Systém s intermitentními selháními

tentní selhání, u nichž je vysoká pravděpodobnost, že modul může být používán i nadále bez jakéhokoliv druhu opravy.

Na základě modelování intermitentních selhání modulů s parametry λ a μ a se zohledněním časových okamžiků atomických kontrol t_{AT} lze intermitentní selhání rozdělit do následujících tří druhů:

1.druh zahrnuje intermitentní selhání, která mohou být odhalena při několika málo (2 až M) opakovaných atomických kontrolách[6]

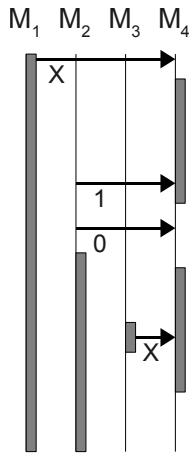
2.druh intermitentní selhání, která sice mohou být odhalena po opakovaných kontrolách, avšak těchto kontrol však musí být relativně velký počet (až např. v řádu 10^6)

3.druh intermitentní selhání, u nichž je možnost zachycení velmi nepravděpodobná, resp. zachycení nastává nejvýše jedenkrát během diagnostiky.

Souběh intermitentních selhání a atomických kontrol lze nejlépe znázornit na modifikovaném **sekvenčním diagramu**, jenž je znám například z modelovacího jazyka UML. Ukázkový diagram je na obrázku 1.28 na následující straně.

Jednotlivé moduly jsou v tomto diagramu zobrazeny jako svislé čáry. Ve směru ze shora dolů roste čas. Atomické kontroly jsou znázorněny jako vodorovné čáry se šipkami. Jejich vertikální poloha (osa y) určuje čas provedení atomické kontroly, přičemž atomická kontrola ležící výše je provedena před atomickou kontrolu ležící níže. Počátek a konec šipky (v horizontálním směru) určuje modul provádějící kontrolu (počátek) a modul kontrolovaný (konec). Pod šipkou je zapsán obdržený syndrom (syndromy na obrázku odpovídají definice 1.1, kde X je náhodná veličina).

[6] M je dáno časovými i výpočetními možnostmi systému, je však typicky malé, nejvýše v řádu desítek



Obrázek 1.28.: Sekvenční diagram intermitentního systému

Pro příklad, první (nejčasnější) atomická kontrola je prováděna modulem M_1 a kontrolován je modul M_4. Výsledkem může být jednička nebo nula.

Selhání je v grafu znázorněno tmavě šedým obdélníkem u svislice daného modulu, jenž pokrývá časové období, v němž modul selhává. Modul M_1 je permanentně chybný po celou dobu diagnostiky systému, u modulu M_2 se také jedná o selhání permanentní, jenž vzniká až v průběhu diagnostiky systému a diagnostiku nijak neovlivňuje, neboť modul se po selhání již neúčastní atomických kontrol. Modul M_3 selže jen na velmi krátký okamžik a pouze jednou. Jedná se tedy o intermitentní selhání třetího druhu (je zachyceno jedenkrát). U modul M_4 se opět jedná o intermitentní selhání, které se však opakuje a výrazněji ovlivňuje diagnostiku. Díky malému počtu atomických kontrol nelze stanovit, zda se jedná o selhání prvého nebo druhého druhu.

Pro diagnostiku intermitentních selhání prvého druhu byly navrženy metody [13] založené na sumárním syndromu R_Σ, jenž může být získán po M-násobně opakovaném provedení množiny atomických kontrol.

Sumární syndrom spočítáme takto:

$$R_\Sigma = \{r^*_{ij}\}, \quad r^*_{ij} = \bigvee_l r^l_{ij}, \text{ kde } r^l_{ij} \in R_l(\text{syndrom obdržený při } l\text{-tém opakování})$$

Operace „$\vee$“ je zobecněný logický součet (podobně jako v (1.8)). Jinak řečeno, pokud bude v jednom opakování AT zjištěn u atomické kontroly výsledek „0“ a ve

druhém opakování „1“, je sumární výsledek roven $0 \vee 1 = 1$ a modul je označen za chybný.

Diagnostika může být provedena pouze v případě, že bude splněna následující podmínka:

$$R_\Sigma \in R_o \tag{1.12}$$

kde R_o je množina sumárních syndromů, které mohou být obdrženy v případě, že jsou možná pouze trvalá selhání modulů, za podmínky, že počet nesprávných modulů nepřekročí hodnotu t.

Pokud je podmínka (1.12) splněna, je provedena běžná diagnostika např,. pomocí tabulkové metody, ale vychází se ze sumárního syndromu. Moduly označené za chybné mají buď permanentní selhání, nebo se jedná o intermitentní selhání prvého druhu.

Podmínka (1.12) není splněna, je-li výsledkem sumární syndrom, který je nekonzistentní tj. obsahuje výsledky, které jsou konfliktní. Výsledek je konfliktní, pokud je v sumárním pohledu některý modul jedním správným modulem označen za správný a druhým taktéž správným za chybný. V tomto případě se obvykle další diagnostika neprovádí a výsledkem je jednoduché oznámení, že systém nemůže být správně diagnostikován. Tento případ nastává v případě, kdy v systému existují moduly s intermitentními selháními druhého a třetího druhu.

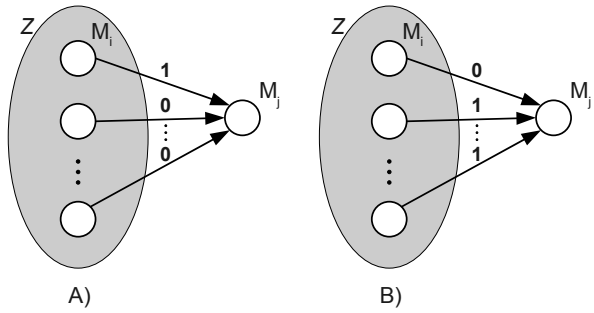
Pokud jsou však k dispozici další prostředky (časové a režijní), může procedura diagnostiky dále pokračovat, čímž lze získat další informace. Nejjednodušší možností rozšíření je zvýšení počtu opakování množiny atomických kontrol. Poté mohou být některé intermitentní moduly odhaleny jako selhávající, a to i z pohledu modulů, které tuto skutečnost ještě neodhalily, což vede k odstranění nekonzistencí a zařazení těchto modulů mezi moduly se selháním prvého druhu.

Druhá možnost je o něco zajímavější, neboť sice vyžaduje složitější prozkoumání sumárního syndromu a přináší i určitý risk (tj. pravděpodobnost nesprávného výsledku diagnostiky), nevyžaduje však provádění dalších kontrol.

Základem této metody je předpoklad, že **všechna zbývající neodhalená intermitentní selhání jsou třetího druhu** Pravděpodobnost chybného výsledku je rovna pravděpodobnosti nesplnění tohoto předpokladu. Tento předpoklad je odůvodněný, neboť v reálných složitých systémech se tento typ intermitentních selhání vyskytuje mnohem častěji než ostatní druhy selhání.

V tomto případě (tj. pokud není splněna podmínka (1.12)), je prvním krokem stanovení podmnožiny Z, do níž patří všechny moduly, které mohou být na základě

sumárního syndromu R_Σ označeny jako správné. V druhém kroku je nutno ověřit konzistentnost všech výsledků atomických kontrol, jež jsou prováděny moduly z podmnožiny Z. Je tedy nutno zjistit, zda moduly z této podmnožiny stejně hodnotí moduly z podmnožiny doplňkové ($\bar{Z}$) nebo nikoliv. V průběhu zjišťování může nastat jedna ze situací znázorněných na obrázku 1.29.



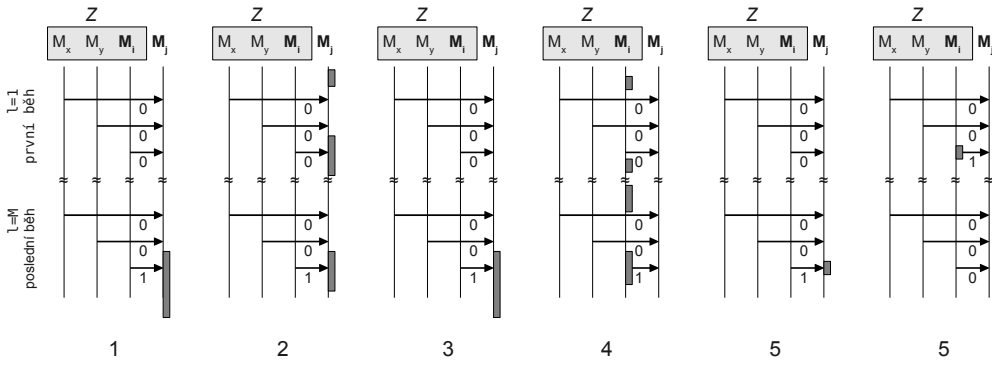
Obrázek 1.29.: Situace způsobené intermitentním selháním třetího druhu

Situace A znázorněná na obrázku 1.29 může vzniknout z následujících příčin:

1. modul M_j selhal v okamžiku těsně před provedením poslední atomické kontroly v posledním opakování AT (zde je to kontrola provedená modulem M_i tj. τ_{ij})
2. modul M_j má intermitentní selhání druhého druhu a modul M_i je jediný z modulů, který jej zaregistroval
3. modul M_i permanentně selhal. Atomická kontrola τ_{ij} je první kontrola, která byla tímto selháním dotčena
4. modulu M_i má intermitentní selhání. Selhání bylo zachyceno pouze atomickou kontrolou τ_{ij}.
5. buď modul M_i nebo M_j má intermitentní selhání třetího druhu. Toto selhání se projevilo v průběhu atomické kontroly τ_{ij}.

Současné intermitentní selhání obou modulů nebudeme uvažovat, neboť pravděpodobnost takovéto události je velmi nízká.

Jednotlivé možné příčiny konfliktů jsou přehledně znázorněny v sekvenčních diagramech na obrázku 1.30 na následující straně. V případech 2, 4, 5 existuje více možných souběhů selhání a atomických kontrol. Znázorněn je však vždy pouze jeden, resp. u pátého případu výjimečně dva.



Obrázek 1.30.: Souběhy selhání a atomických kontrol v situaci A

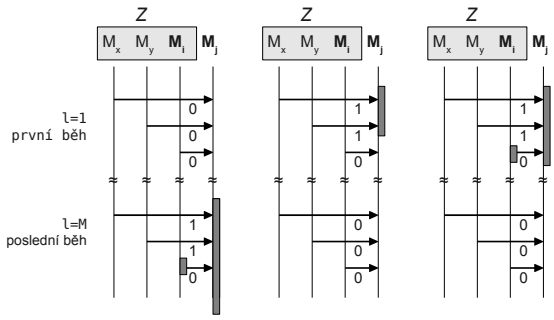
V souladu s přijatým předpokladem budeme dále uvažovat pouze intermitentní selhání třetího druhu (viz příčina 5).

Můžeme proto tvrdit, že buď modul M_i nebo M_j má krátkodobé intermitentní selhání. Takové selhání se s vysokou pravděpodobností nebude opakovat a modul bude nadále fungovat bez problémů. Proto není pro další úvahy důležité, který z obou modelů selhal. Jediným cílem je odstranění nekonzistencí v syndromu. Řešení je v tomto případě snadné, stačí zaměnit výsledek kontroly τ_{ij} z hodnoty „1“ na hodnotu „0“ .

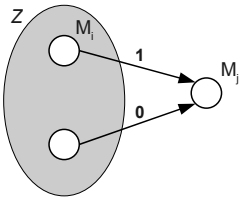
Situace B z obrázku 1.29 je snadněji řešitelná, neboť může nastat pouze v případě, že modul M_j má intermitentní selhání, které odhalily všechny moduly z podmnožiny Z vyjma modulu M_i. Ukázky možných souběhů jsou na obrázku 1.31. Řešení je jednoznačné, stačí zaměnit výsledek kontroly τ_{ij} z hodnoty „0“ na „1“; a tím zajistit konzistentní stav sumárního syndromu.

Komplikovanější situace vzniká v případě nekonzistence u dvouprvkové podmnožiny správných modulů Z (viz obrázek 1.32 na následující straně).

Obdržený výsledek lze interpretovat buď jako situaci A (a řešit ji změnou jednoho ohodnocení na „0“) nebo jako situaci B (v tomto případě jsou ohodnocení sjednocena na „1“). Abychom mohli učinit rozhodnutí a přiklonit se k jednomu z řešení, musíme porovnat pravděpodobnosti obou situací. V případě interpretace podle situace A by se jednalo o intermitentní selhání třetího druhu modulu M_i nebo M_j. Podle druhé interpretace (situace B) musí mít modul M_i selhání druhého druhu. Protože pravděpodobnost vzniku selhání tohoto druhu je menší, je vhodnější zvolit řešení podle situace A, tj. sjednotit syndrom na hodnotu „0“ ($\tau_{ij} = 0$, volíme pozitivnější řešení).



Obrázek 1.31.: Souběhy selhání a atomických kontrol v situaci B



Obrázek 1.32.: Situace způsobená intermit. selháním třetího druhu (spec. případ)

Na konci sekce ještě shrneme specifické rysy systémů s intermitentními selháními:

I. Hlavním cílem diagnostiky není identifikace modulu s intermitentním selháním, ale řešení konfliktních situací, které vznikají z důvodů specifického charakteru těchto selhání.

II. Samotná diagnostika zahrnuje několik dílčích kroků:

Krok 1: vícenásobné opakování množiny atomických kontrol a získání sumárního syndromu R_Σ.
Krok 2: ověření, zda obdržený syndrom odpovídá podmínce $R_\Sigma \in R_o$. Pokud je tato podmínka splněna, pak diagnostika probíhá stejně jako v případě permanentních selhání. V opačném případě se přechází k dalšímu kroku.
Krok 3: učení množiny Z, jenž obsahuje moduly, které lze na základě sumárního syndromu jednoznačně považovat za správné.
Krok 4: kontrola konzistentnosti výsledků atomických kontrol prováděných moduly z množiny Z.
Krok 5: vyřešení konfliktních situací.

III. Intermitentní selhání je možno rozdělit do tří druhů podle počtu opakování množiny atomických kontrol nutných pro zachycení selhání modulu.

Intermitentní selhání prvého druhu jsou odhalena již v kroku 2. Selhání druhého druhu však mohou být odhalena až po provedení kroku 3 (a to pouze některá). Selhání třetího druhu jsou tolerována. To znamená, že systém je schopen dalšího provozu bez vnějšího zásahu. Konfliktní situace způsobené intermitentními selháními třetího druhu jsou řešeny v kroku 5.

IV. Nevýhodou diagnostiky intermitentních selhání je výrazně vyšší časová složitost resp. režie systému. Proto může být velmi náročné, ne-li nemožné provádět tuto diagnostiku v průběhu provozu reálných systémů.

2. Organizace samodiagnostiky a samokontroly

2.1. Samodiagnostika a samokontrola bez vnějšího pozorovatele

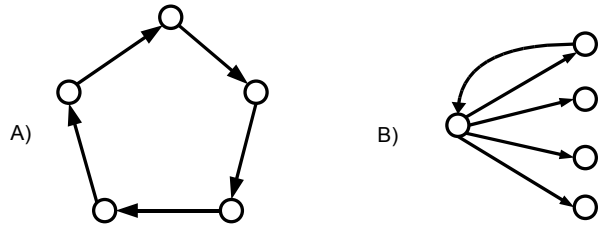
Samodiagnostika (angl. *selfdiagnotics*) je v souladu s použitím předpony „samo“ charakterizována tím, že kontroly i následná diagnostika stavu systému jsou prováděny přímo moduly daného systému, tj. nepoužívá se žádný externí modul nebo zařízení. Často se používá i obdobný termín samokontrola resp. autokontrola[1], jehož definice se však poněkud liší, a je proto nutné oba termíny rozlišovat.

Samokontrola je proces, jehož cílem je rozlišení dvou stavů systému: správný (bezchybný) resp. nesprávný (chybný). Výsledek samokontroly neukazuje, který z modulů v systému selhal. Samokontrola vyžaduje pouze minimální počet atomických kontrol, stačí pouze zajistit, že každý modul je alespoň jednou zkontrolován, tj. postačuje pouze N atomických kontrol (viz příklady na obrázku 2.1 na následující straně). Při samokontrole není navíc nutné vytvářet syndrom, tudíž odpadá i fáze jeho analýzy. Postačuje totiž pouze signalizace chybného stavu systému modulem, který provedl alespoň jednu atomickou kontrolu s výsledkem „1“. Signalizace se děje do prostředí systému, resp. do jeho okolí.

Samokontrola bývá často prováděna samostatně před případnou samodiagnostikou, není to však nutné. Samokontrola totiž může být v mnoha případech obsažena (vnořena) přímo v samodiagnostice. Atomické kontroly nutné pro samokontrolu jsou v tomto případě podmnožinou atomických kontrol samodiagnostiky.

Na rozdíl od samokontroly provádí **samodiagnostika** lokalizaci zdrojů chyb v systému. Je možné zjistit konkrétní modul, resp. moduly, které selhaly, nebo podmnožinu modulů, která obsahuje všechny chybné moduly. Úroveň lokalizace závisí na struktuře a počtu atomických kontrol.

[1] termín *autokontrola* je v češtině běžnější, bohužel odpovídající termín *autodiagnostika* je užíván především pro diagnostiku automobilů. Z tohoto důvodu je v rámci celé knihy preferován jednoznačnější prefix „samo“



Obrázek 2.1.: Příklady diagnostických grafů samokontroly

Samodiagnostika však již vyžaduje provedení většího počtu atomických kontrol, nutností je i vytvoření a zpracování syndromu. Tímto procesem jsme se zabývali v předchozí kapitole, situaci jsme si však zjednodušili předpokladem *imaginárního externího pozorovatele*, který přebírá syndrom, a na základě tohoto syndromu provádí diagnostiku stavů jednotlivých modulů.

Ve skutečnosti však musí být zpracování syndromu prováděno přímo v systému prostřednictvím tzv. *diagnostického jádra*.

Diagnostické jádro je modul nebo množina modulů (hardwarových nebo softwarových), které provádějí následující činnosti:

- analýza syndromu, tj. výsledků množiny atomických kontrol
- rozhodování o stavu systému
- signalizace výsledků diagnostiky do prostředí systému

Obecně není nutné, aby se diagnostické jádro zúčastnilo provádění atomických kontrol. Diagnostické jádro buď nemusí vůbec provádět atomické kontroly, respektive může provádět všechny atomické kontroly, nebo provádí pouze některé atomické kontroly. Diagnostické jádro se navíc může připojit k provedení atomických kontrol v libovolný okamžik v průběhu samokontroly nebo samodiagnostiky.

Výběr modulu resp. modulů, kterým mohou být předány výsledky atomických kontrol, může být proveden různými způsoby, což určuje různé organizace samodiagnostiky. Pod *organizací samodiagnostiky* rozumíme strukturu provedení atomických kontrol a předání jejich výsledků mezi moduly systému.

V současnosti je známo několik různých mechanismů formování diagnostického jádra, které zohledňují různá časová například režijní omezení, nebo disponibilní hardwarové a softwarové prostředky. Navíc existují různá kriteria, na jejichž základě můžeme diagnostická jádra klasifikovat. Například bylo navrženo stanovit jako hlavní kriterium okamžik formace diagnostického jádra a zároveň jeho závislost na výsledcích jednotlivých atomických kontrol [14]. Podle tohoto kriteria lze odlišit tři základní typy diagnostických jader:

1. přidělené, tj. předem stanovené jádro
2. formující se jádro
3. putující jádro.

Podle druhu použitého jádra lze popisovat a klasifikovat i různé typy organizace samodiagnostiky. Základní typy v současnosti používaných organizací samodiagnostiky (a tudíž i základní druhy jader) jsou shrnuty v následující podkapitole.

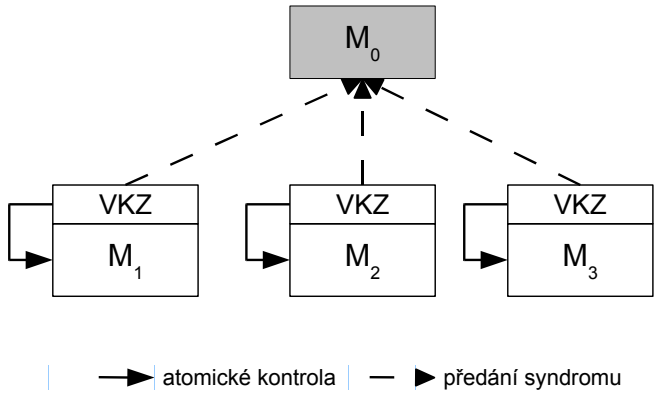
2.2. Organizace samodiagnostiky a diagnostické jádro

Popis jednotlivých typů organizací začneme u několika přístupů, které se příliš neliší od klasických diagnostickým modelů s použitím externího kontrolního zařízení (SKA, viz strana 9), nebo externího pozorovatele (viz strana 12). Funkci pozorovatele, resp. kontrolního zařízení, však přejímá interní modul, který je součástí systému. Tento modul je buď vyhrazený (má pouze diagnostické funkce) nebo může současně vykonávat i běžnou činnost. Z hlediska diagnostických funkcí však má tento modul speciální postavení, které je předem určeno a v průběhu činnosti systému se nemění.

První typ organizace je znázorněn na obrázku 2.2 na následující straně. Diagnostické jádro je v tomto případě representované *vyhrazeným* modulem M_0. Diagnostické jádro je určeno předem, jinými slovy organizace samodiagnostiky má *přidělené diagnostické jádro*. Modul M_0 shromažďuje výsledky atomických kontrol ostatních modulů systému a následně provádí jejich analýzu. Samotné atomické kontroly jsou prováděny *vestavěnými kontrolními zařízeními* (VKZ), která jsou vestavěna v každém modulu. Tyto kontroly se provádí jen v rámci daného modulu. Modul M_0 nemusí mít VKZ, avšak může být zkontrolován předem pomocí spolehlivého kontrolního zařízení. Předpokládá se, že modul M_0 má výrazně vyšší spolehlivost než ostatní moduly systému.

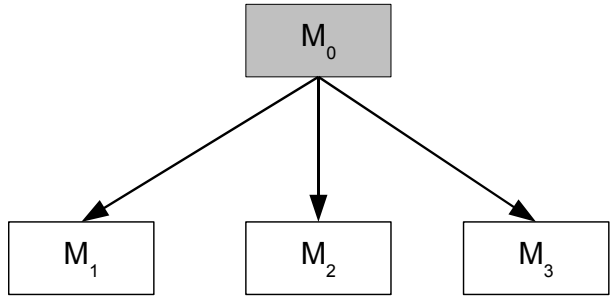
Na rozdíl od externího diagnostického zařízení (SKA) je diagnostický modul částí systému (např. je součástí určitého dílčího systému letadla), bývá výrazně jednodušší a specializovanější (je určen jen pro daný konkrétní systém).

I v druhém typu organizace samodiagnostiky (obr. 2.3 na následující straně) je diagnostické jádro reprezentováno jediným modulem (M_0), jenž je určen předem (organizace má přidělené diagnostické jádro). Specifickým rysem této organizace je omezení provádění atomických kontrol na diagnostické jádro. Žádný z modulů kromě modulu M_0 neprovádí atomické kontroly, tj. může být jen kontrolován. Tato



Obrázek 2.2.: Diagnostické jádro a moduly se neúčastní provádění AT

organizace nevyžaduje předání výsledků atomických kontrol mezi moduly systému, což lze považovat za výhodu této organizace.

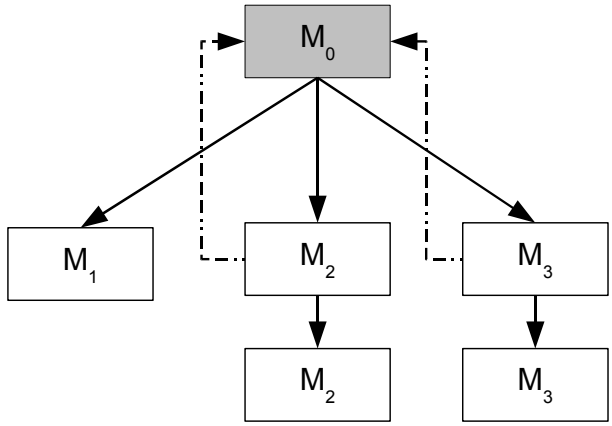


Obrázek 2.3.: Diagnostické jádro provádí všechny AT

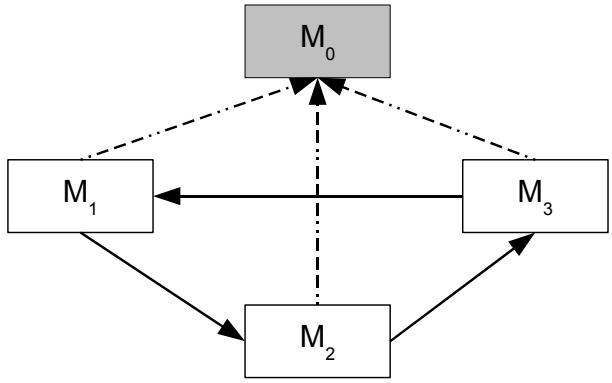
Obě výše uvedené organizace lze dále mírně modifikovat.

Organizace samodiagnostiky na obrázku 2.4 na následující straně stále využívá přidělené diagnostické jádro (modul M_0), atomické kontroly však mohou být prováděny i dalšími moduly. Výsledky jsou pak předávány diagnostickému jádru. Moduly jsou v tomto případě funkčně rovnocennější, neboť provádějí jak běžné, tak kontrolní funkce.

Mírnou modifikací je organizace samodiagnostiky (obr. 2.5), v níž atomické kontroly neprovádí přidělené diagnostické jádro, ale ostatní moduly. Diagnostické jádro pouze přijímá výsledky atomických kontrol a zpracovává je (je tak obdobou první organizace, ale zde již bez VKZ). Tato organizace vyžaduje předávání výsledků jednotlivých AT, poněkud však vyrovnává zatížení jednotlivých modulů.



Obrázek 2.4.: AT jsou prováděny jak jádrem tak i moduly systému



Obrázek 2.5.: Jádro se neúčastní provádění AT

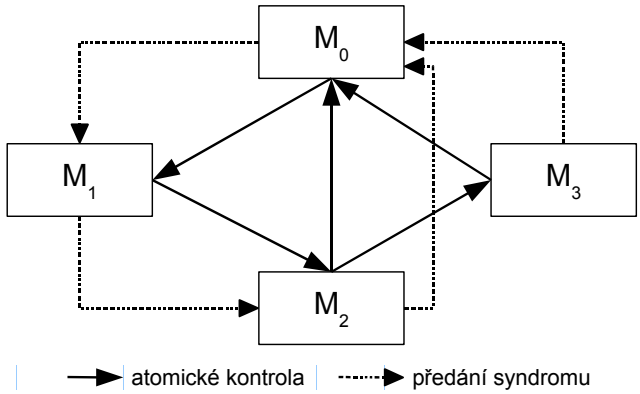
Všechny výše uvedené organizace však sdílejí zcela **zásadní nevýhody**:

1. modul, který vystupuje v roli diagnostického jádra, je silně zatěžován diagnostickými funkcemi, tj. je omezen ve vykonávání běžných funkcí systému, resp. vykonává jen činnost diagnostickou
2. je nutno zajistit značnou spolehlivost modulu M_0, což může být velmi obtížné.

Řešením těchto problémů je systém, v němž je distribuce kontrolních a diagnostických funkcí na moduly rovnoměrnější a volba diagnostického jádra je dynamičtější.

Z tohoto důvodu není v organizaci samodiagnostiky znázorněné na obr. 2.6 pře-

dem stanoveno diagnostické jádro. Dokonce ani není možno předpovědět, který z modulů bude provádět rozbor syndromu, tj. bude vystupovat v roli diagnostického jádra. Naopak se předpokládá, že moduly mají přibližně stejnou spolehlivost a všichni mohou vykonávat funkce diagnostického jádra.



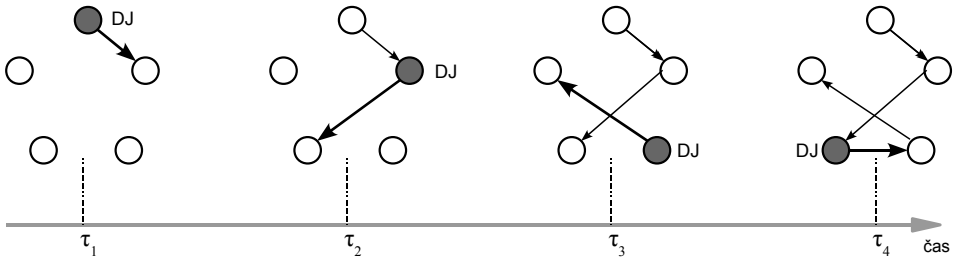
Obrázek 2.6.: Formace jádra na základě výsledků AT

Atomické kontroly mohou být (a běžně jsou) prováděny všemi moduly systému, a to jak v průběhu normálního provozu systému, tak i před, resp. po vykonání přidělených úkolů. Struktura provedení množiny atomických kontrol a pořadí provedení jednotlivých atomických kontrol jsou však *stanoveny předem*.

Po ukončení provedení všech atomických kontrol jsou jejich výsledky (dílčí syndromy) předávány mezi moduly. Existují různé návrhy organizace předávání výsledků atomických kontrol mezi moduly systému a formování diagnostického jádra. Jedním z možných východisek je strategie předávání výsledků všem modulům, které zdrojový modul považuje za správné. Tato strategie předání výsledků eliminuje možnost, že chybný modul bude vystupovat v roli diagnostického jádra. Může se tak stát (např. když všechny moduly jsou správné), že několik modulů bude současně provádět rozbor syndromu a každý z nich bude následně signalizovat do prostředí systému výsledek diagnostiky. Jinak řečeno, více modulů bude vystupovat v roli nezávislých diagnostických jader. Prostředí systému musí být na takovou situaci připraveno.

Charakteristickou vlastností této organizace samodiagnostiky je skutečnost, že diagnostické jádro je zformováno až po ukončení všech atomických kontrol a na základě jejich výsledků.

Zpřesněním a rozvinutím této dynamické organizace samodiagnostiky je ještě dynamičtější organizace založená na tzv. „putujícím jádře“ (angl. *wandering kernel*) [14], [15].



Obrázek 2.7.: Putující jádro

I zde mohou být atomické kontroly prováděny všemi moduly systému a moduly si předávají výsledky kontrol mezi sebou. Každý modul shromažďuje výsledky atomických kontrol (svých i cizích), formuje syndrom a bezprostředně ověřuje, je-li syndrom postačující pro určení stavů ostatních modulů systému. Rozhodnutí o dostatečnosti syndromu je provedeno na základě požadavků důvěryhodnosti celkového výsledku diagnostiky. Můžeme říci, že každý modul „se snaží“ stát diagnostickým jádrem (tj. zformovat dostačující syndrom, provést jeho rozbor a signalizovat do prostředí systému výsledky diagnostiky). Pro danou organizaci je podstatné, že diagnostické jádro může vzniknout v kterýkoli okamžik, přičemž tento okamžik závisí na požadavcích na důvěryhodnost výsledků diagnostiky. Čím vyšší jsou tyto požadavky, tím déle trvá proces vzniku diagnostického jádra. Obrázek 2.7 vysvětluje, proč takto vytvořené diagnostické jádro (DJ) dostalo název „putující“.

Jak je vidět z obrázku, každý modul, který provádí atomickou kontrolu τ_i, kde $i = 1 \ldots 4$, se snaží vytvořit diagnostické jádro. V různé okamžiky je diagnostické jádro vytvořeno různými moduly, tj. diagnostické jádro se jakoby přemisťuje od jednoho modulu ke druhému. Jestliže není předem stanoveno pořadí provedení atomických kontrol, což je typické pro danou organizaci samodiagnostiky, je cesta diagnostického jádra od jednoho modulu ke druhému nahodilá a připomíná cestu vandrovníka. Tato asociace se stala podnětem pro výběr názvu diagnostického jádra i celé organizace.

2.3. Putující diagnostické jádro

Organizace samodiagnostiky a samokontroly, které nevyužívají předem stanovené diagnostické jádro, přinášejí několik podstatných výhod.

Za prvé: umožňují dosáhnout vysoké úrovně důvěryhodnosti výsledku celkového systému i při použití množiny výsledků atomických kontrol s nízkou důvěryhod-

ností. Strategie, která umožňuje z nespolehlivých a nedůvěryhodných elementů vytvořit spolehlivý resp. důvěryhodný celkový systém, je známa již relativně dlouho. Například již I. von Neuman, E. F. Moore a C. E. Shannon [16] navrhli teorii maskující redundance (nadbytečnosti) pro vytváření spolehlivých systémů z méně spolehlivých komponent. V našem případě tudíž nepotřebujeme zajistit vysokou spolehlivost u modulů, které provádí kontrolu a diagnostiku, což může být v mnoha situacích velmi obtížné.

Za druhé: mají takové organizace vysokou odolnost proti závadám jednotlivých modulů systému. To znamená, že v případě současného selhání několika modulů provádějících atomické kontroly je možno zajistit správný výsledek celosystémové kontroly a diagnostiky.

Za třetí: tato organizace nevyžaduje žádné vnější zařízení, což zajišťuje jejich dlouhodobé autonomní fungování a zároveň nepřetržitou kontrolu a diagnostiku.

Bohužel však organizace bez přiděleného diagnostického jádra vede i k určitým problémům, a to především v systémech reálného času. Vzhledem k obtížnosti praktické realizace samodiagnostiky bez předběžně přiděleného jádra bylo vynaloženo mnoho úsilí modernizovat teorii samodiagnostiky za účelem stanovení takového postupu, jenž by umožnil efektivně provádět samodiagnostiku v systémech reálného času. Výsledkem této výzkumné práce bylo vyvinutí nové organizace : organizace s putujícím diagnostickým jádrem.

Tato nová organizace samodiagnostiky se liší od předchozích v následujících bodech:

- v organizaci provedení atomických kontrol;
- ve způsobu, jakým bude vybrán modul pro vykonání roli diagnostického jádra;
- v možnosti ukončení procedury kontroly a diagnostiky v kterýkoli okamžik včetně poskytnutí odpovídajícího výsledku diagnostiky.

Organizace vychází ze skutečnosti, že **atomické kontroly jsou prováděny v průběhu fungování systému**, tj. nelze předem stanovit, jaké moduly budou v určitý okamžik nečinné (tj. neprovádí systémové úkoly) a budou se schopny zúčastnit atomické kontroly. Z toho vyplývá, že nejen dvojice modulů, jež provádí atomickou kontrolu, ale i čas provedení kontroly jsou nahodilé. Nahodilý je také počet atomických kontrol provedených v systému za určitý čas a též i samotná struktura AT. Na rozdíl od organizace formujícího se diagnostického jádra, kde procedura samokontroly je vnořena v samodiagnostice (tj. atomické kontroly, které zajišťují samokontrolu jsou zároveň použity i pro samodiagnostiku), jsou v případě putujícího diagnostického jádra **procedury samokontroly a samodiagnostiky oddělené** a mají vlastní strategii provedení atomických kontrol.

Na začátku zjišťování stavu systému se nejdříve provádí **procedura samokontroly**. Doba trvání samokontroly závisí na požadavcích kladených na důvěryhodnost výsledku samokontroly. Pokud se v průběhu samokontroly neobjeví žádný jednotlivý výsledek atomické kontroly, který by svědčil o existenci chybného modulu (tj. *všechny atomické kontroly* skončí s výsledkem **„0“**)[2], je procedura samokontroly ukončena a příslušný výsledek je signalizován do prostředí systému. Procedura samokontroly a signalizace výsledků pak může být vykonávána opakovaně v určitých intervalech, dokud je systém v provozu.

V opačném případě, tj. pokud se objeví výsledek AT svědčící o existenci chybného modulu *(alespoň jedna atomická kontrola* s výsledkem **1**), je procedura samodiagnostiky ihned ukončena a je zahájena procedura samodiagnostiky, která má za cíl odhalit chybný modul nebo moduly.

Jak ukazuje výzkum, je jedním z nejdůležitějších a nejobtížnějších úkolů stanovit časový rozsah procedury samokontroly v případě, kdy všechny výsledky atomických kontrol svědčí o správném stavu systému. Nejdříve však zavedeme klíčový pojem:

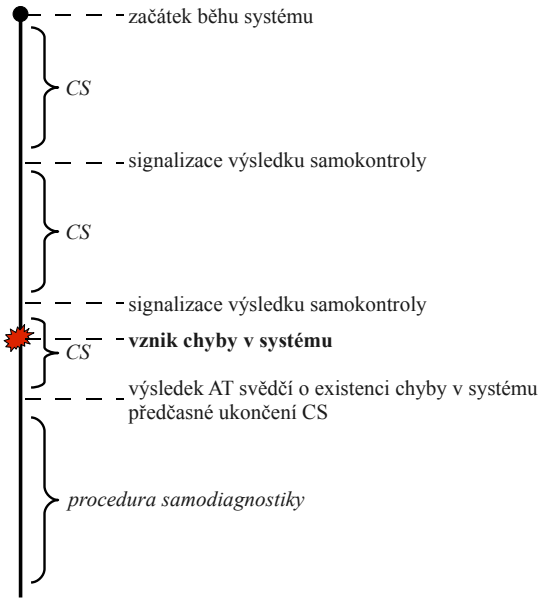
DEFINICE 2.1:

Cyklus samokontroly je interval mezi dvěma za sebou bezprostředně následujícími signalizacemi výsledků samokontroly.
□

Je nutno zdůraznit, že cyklus samokontroly *nezahrnuje* proceduru samodiagnostiky. Na obrázku 2.8 je znázorněn cyklus samokontroly (CS) a případná následující procedura samodiagnostiky. Obrázek 2.8 také ilustruje další důležitý rys samokontroly. Z obrázku lze vidět, že vznik chyby v systému nevede k bezprostřednímu ukončení samokontroly. Samokontrola ještě zpravidla určitý čas běží, dokud není chyba zachycena jednou z atomických kontrol.

Po ukončení každého cyklu samokontroly je signalizován výsledek, jenž vždy určuje, že systém je správný. Pouze v případě mimořádného ukončení cyklu samokontroly (viz obr. 2.8) není do prostředí systému poskytován žádný výsledek. Z toho vyplývá, že signalizovaný výsledek je vždy stejný. Lze tudíž uvažovat o systému, v němž výsledek samokontroly nemusí být vůbec signalizován, neboť postačuje signalizace výsledku následné samodiagnostiky (pokud nějaký vznikne). V tomto případě může prostředí systému interpretovat nepřítomnost výsledku jako náznak toho, že systém je správný. Tento návrh však není dostatečně prozkoumán jak z hlediska teorie, tak v praktické aplikaci. Nicméně může být tento návrh interpretován i v rámci původního přístupu, uvažujeme-li dobu trvání cyklu jako limitně se blížící k nekonečnu.

[2] nulová hodnota všech atomických kontrol nezaručuje bezchybnost modulů (viz strana 13)



Obrázek 2.8.: Cykly samokontroly a vznik chyby

Pro organizaci cyklu samokontroly (tj. především pro stanovení délky cyklu) bylo navrženo několik přístupů. V zásadě lze dobu provádění cyklu stanovit:

- *s pevným časem* – doba cyklu je konstantní a je stanovena předem (dále je označována jako t_c)
- *s pevným počtem atomických kontrol* – doba je definována určitým počtem atomických kontrol, tj. cyklus samokontroly běží, dokud není proveden předem stanovený počet atomických kontrol. Čas provedení se může v tomto případě lišit
- *s cílovým diagnostickým grafem* — cyklus je definován určitou strukturou atomických kontrol. Cyklus běží, dokud není zformována předem stanovená struktura atomických kontrol (resp. jedna z předem stanovené podmnožiny struktur). I v tomto případě je doba provedení cyklu nahodilá.

Přístupy, v nichž je doba trvání cyklu samokontroly definována fixním časem nebo počtem atomických kontrol, mohou být charakterizovány i z dalšího pohledu: zdali je provedena analýza dosažené struktury atomických kontrol či nikoliv.

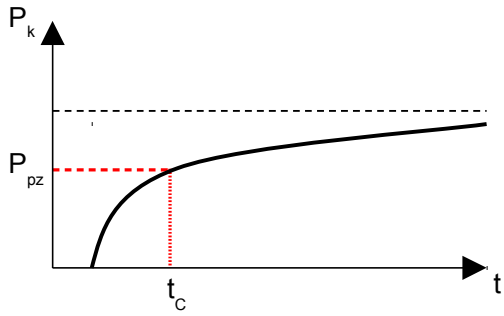
V případě, že není prováděna analýza diagnostického grafu v rámci CS, je naopak nutno předem vypočítat pravděpodobnost, že **zkontrolovány budou všechny mo-**

duly alespoň jednou atomickou kontrolou. To je dostatečné pro fázi samokontroly a příliš to nezatěžuje systém (samokontrola je prováděna souběžně s během systému). V praxi se používá opačný postup, kdy je na základě požadované pravděpodobnosti události, že jsou zkontrolovány všechny moduly, vypočítána doba trvání cyklu samokontroly, resp. počet atomických kontrol, které by měly být provedeny za dobu trvání cyklu.

V druhém případě je nutno provést analýzu DG v rámci každého cyklu, abychom se přesvědčili, že každý modul byl alespoň jednou zkontrolován, resp. obecněji, že diagnostický graf splňuje předem definované požadavky (tj. patří do určité třídy diagnostických grafů). Se znalostí DG může být spočítána důvěryhodnost výsledku samokontroly. V případě, že vypočtená důvěryhodnost nevyhoví stanoveným požadavkům, je možno prodloužit dobu trvání cyklu samokontroly o předem stanovený časový interval. Po vypršení prodlouženého termínu je opětně proveden výpočet důvěryhodnosti výsledku se započítáním původních a dodatečných atomických kontrol. Zjištění optimálního počtu možných prodloužení a doby jejich trvání je složitý problém, který je stručně diskutován v [15] a [17].

2.4. Organizace provedení samokontroly s pevnou dobou trvání cyklu

Pro organizace samokontroly s pevnou dobou trvání cyklu je velmi důležitá funkcionální závislost pravděpodobnosti události, že všechny moduly systému alespoň jednou byly zkontrolovány, v závislosti na čase t, tj. $P_k = f(t)$. Průběh této funkce je znázorněn na obrázku 2.9.



Obrázek 2.9.: Závislost pravděpodobnosti P_k na čase t

Znalost této funkce umožňuje na základě požadavku na důvěryhodnost výsledku samokontroly (vyjádřeným v podobě pravděpodobnosti P_{pz}) vypočíst dobu trvání

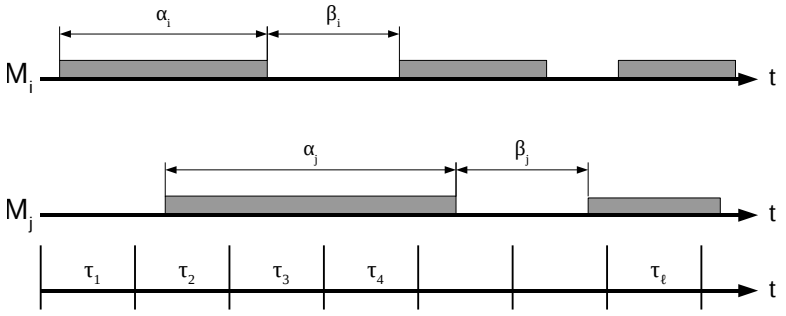
cyklu samokontroly t_C.

Důvěryhodnost výsledku samokontroly je pravděpodobností události, že **všechny nesprávné moduly budou odhaleny**. Pro *zjednodušení* budeme prozatím předpokládat, že tato pravděpodobnost je rovna *pravděpodobnosti provedení kontroly všech modulů*. Nezohledňujeme tak pravděpodobnost nesprávného výsledku atomické kontroly, tj. událost, kdy je chybný modul označen za správný. Důvěryhodnost samokontroly navíc také závisí na pravděpodobnosti události, že selhání modulu nastane až po provedení všech atomických kontrol nad modulem, nicméně ještě v intervalu cyklu samokontroly. Tato pravděpodobnost závisí na intenzitě atomických kontrol, ale její výpočet je již náročnější. Budeme se mu věnovat v kapitole 2.6.

Je také nutno poznamenat, že ne všechny atomické kontroly jsou stejně důležité resp. informativní. Obecně je proto možné zavést omezení, která eliminují méně důležité atomické kontroly. Jako příklad méně důležitých atomických kontrol je možno uvést duplicitní kontroly, které nepřidávají nové informace o stavu systému. Omezení počtu prováděných kontrol přirozeně ovlivňuje jak pravděpodobnost P_K, tak celkovou dobu trvání cyklu samokontroly.

Existuje několik způsobů jak je možno získat funkci $P_K = f(t)$. Lze ji získat jak teoreticky resp. na základě modelování tak pomocí reálného testování. Teoretický výpočet je založen na použití teorie pravděpodobnosti a kombinatoriky.

Nejdříve uvažujme případ, kdy nejsou kladena žádná omezení na provedení atomických kontrol. Dále je v souladu s výše uvedenou organizací putujícího jádra nutno předpokládat, že provedení atomických kontrol je možné pouze tehdy, pokud moduly zapojené do samokontroly neprovádí systémové úkoly (viz obrázek 2.10).



Obrázek 2.10.: Časový průběh zaneprázdnění modulů

Na obrázku 2.10 jsou symboly α_i, α_j resp. β_i, β_j označeny periody, kdy jsou moduly

M_i a M_j zaneprázdněny, resp. jsou volné (v tomto pořadí). Celkový čas běhu systému je rovnoměrně rozdělen na shodné intervaly τ_ℓ, kde $l = 1,2,...,s$. Délka tohoto intervalu musí splňovat podmínku:

$$t_{AT} < \tau_\ell < 2t_{AT},$$

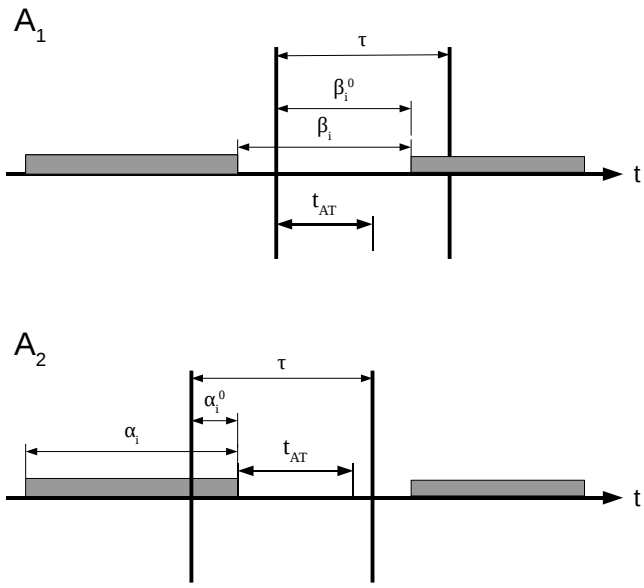
kde t_{AT} je doba provedení atomické kontroly.

Pravděpodobnost P_i, že i-tý modul je v libovolně vybraný okamžik volný, a tudíž je schopen se zúčastnit atomické kontroly, tj. může být kontrolován nebo naopak kontrolovat ostatní moduly, lze vyjádřit jako:

$$P_i = \frac{\bar{\beta}_i}{\bar{\alpha}_i + \bar{\beta}_i},$$

kde $\bar{\alpha}_i$ resp. $\bar{\beta}_i$ jsou střední hodnoty trvání zaneprázdněné resp. volné periody.

Pravděpodobnost P_i^{AT}, že i-tý modul bude schopný atomické kontroly během trvání intervalu τ, je určena dvěma událostmi – A_1 a A_2 (viz obr. 2.11).



Obrázek 2.11.: Události determinující pravděpodobnost P_i^{AT}

Událost A_1nastává tehdy, když začátek intervalu τ spadá do volného intervalu i-tého modulu (β_i) a zároveň je zbytek volné periody β_i (označený jako β_i^0) delší než

t_{AT}. Událost A_2 pak nastane, pokud začátek intervalu τ leží v periodě, kdy je systém zaneprázdněn (α_i) . Pro zbytek zaneprázdněné periody (α_i^0) platí, že je menší než $\tau - t_{AT}$.

Nyní můžeme vyjádřit pravděpodobnost P_i^{AT} následovně:

$$P_i^{AT}(\tau) = P(A_1 \vee A_2) = \frac{\bar{\beta}_i + \tau - 2t_{AT}}{\bar{\alpha}_i + \bar{\beta}_i}$$

Protože do provedení každé atomické kontroly jsou zapojeny *dva* moduly, je maximální počet atomických kontrol proveditelných v průběhu intervalu τ roven $\lfloor \frac{N}{2} \rfloor$. Dále platí, že pravděpodobnost provedení právě jedné kontroly během intervalu τ je rovna pravděpodobnosti situace, kdy zároveň nastanou volné periody u obou modulů a navíc budou mít oba moduly dostatek času na provedení AT.

Tato pravděpodobnost P^{1AT} může být vyčíslena podle vztahu:

$$P^{1AT}(\tau) = P_{2,N}$$

kde výraz $P_{2,N}$ označuje pravděpodobnost situace, že právě dva moduly z N jsou schopny zúčastnit se atomické kontroly. Pravděpodobnost $P_{2,N}$ lze odvodit z následujícího vztahu pro generující (vytvořující) funkci, pro opakované pokusy, kde $m = 2$, $n = N$, $Q_i^{AT} = 1 - P_i^{AT}$ a Z je libovolné:

$$\prod_{i=1}^{n} (Q_i^{AT} + P_i^{AT} \cdot Z) = \sum_{m=0}^{n} P_{m,n} \cdot Z^m \tag{2.1}$$

Podobně lze vyjádřit pravděpodobnost, že bude v průběhu intervalu provedeno právě k atomických kontrol:

$$P^{kAT} = P_{2k,N}$$

kde hodnotu $P_{2k,N}$ lze opět odvodit z rovnosti (2.1).

Střední hodnotu počtu kontrol provedených v intervalu τ lze poté vyjádřit následovně:

$$r = \sum_{k=0}^{\lfloor N/2 \rfloor} k \cdot P_{2k,N}$$

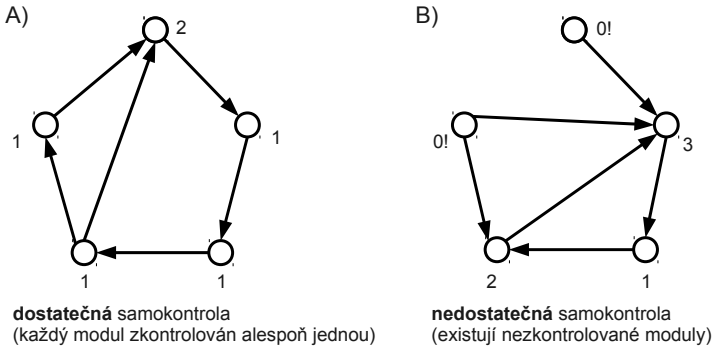
Je zřejmé, že pokud uvážíme větší počet bezprostředně následujících intervalů (řekněme s), tak může být v průběhu s intervalů provedeno v průměru $\omega = s \cdot r$ atomických kontrol. Hodnota ω pak hraje klíčovou roli pro výpočet pravděpodobnosti P_K.

Výpočet P_K v reálném systému je ovlivněn řadou kritérií a omezení, např. nerovnoměrností zatížení jednotlivých modulů, omezenou schopností modulů paralelního provádění kontrol apod. Hrubý, ale relevantní odhad však lze získat i za použití zjednodušujících předpokladů:

- modul může provést více atomických kontrol v rámci intervalu τ (paralelně)
- všechny moduly systému jsou přibližně stejně zatíženy (tj. mají obdobné hodnoty $\bar{\alpha}_i$ a $\bar{\beta}_i$)

Za těchto podmínek může být pravděpodobnost P_K zjištěna analogicky k jednoduchému modelu opakovaných pokusů, v němž každý pokus může mít N navzájem vylučujících se výsledků se shodnou pravděpodobností $P = 1/N$. V našem případě je analogií pokusu rozdělení atomických kontrol mezi moduly, tj. předpokládáme, že s pravděpodobností $P = 1/N$ bude atomická kontrola přidělena i-tému modulu.

Výpočet vychází z počtu možných rozmístění ω atomických kontrol na N modulů (analogie rozdělení ω objektů do N přihrádek), přičemž uvažujeme všechny kombinace jak neúspěšné s 0 kontrolami u některého z modulů (počet = Z_0), tak i úspěšné kombinace, u nichž je zaručena alespoň jedna kontrola u každého modulu (počet = Z_1).



Obrázek 2.12.: Příklad diagnostických grafů pro výpočet $P_k(\omega)$

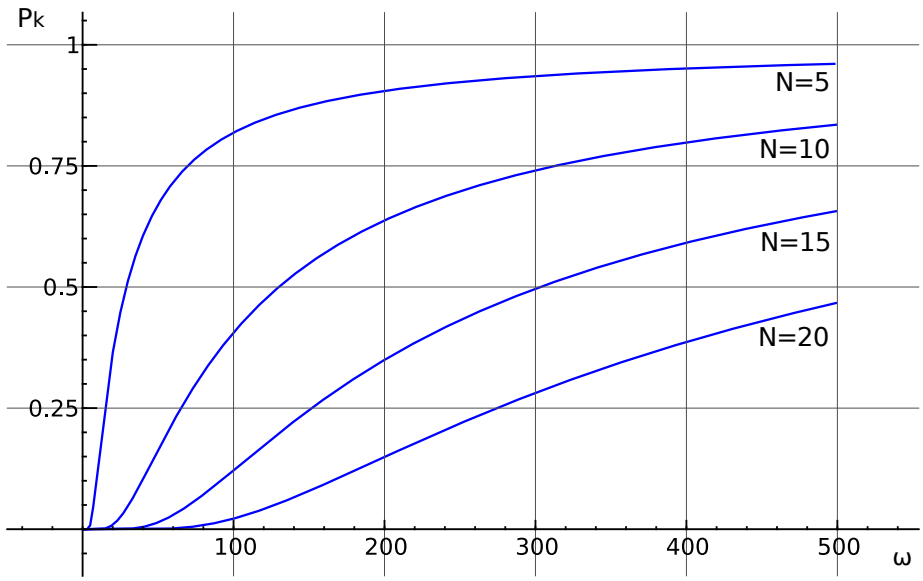
Obrázek 2.12-A ukazuje příklad diagnostického grafu s dostatečnou kontrolou. Rozmístění atomických kontrol je určeno o n-tici počtu kontrol u jednotlivých modulů (zde např. 2,1,1,1,1). Další úspěšné kombinace jsou (1,2,1,1,1), (1,1,2,1,1), (1,1,1,2,1) a (1,1,1,1,2). Jejich počet (Z_1) lze vyjádřit jako: $\binom{\omega-1}{N-1} = \binom{5}{4} = 5$. Na obrázku 2.12-B je graf nedostatečné kontroly s rozmístěním (0,3,1,2,0). Celkový počet kombinací je roven $\binom{N+\omega-1}{N-1} = \binom{10}{4} = 210$.

Pravděpodobnost dostatečné kontroly pro $N = 5$ a $\omega = 6$ je rovna $\frac{5}{210} \doteq 0.024 = 2,4\%$.

Pro případ $\omega > n$ lze tedy pravděpodobnost $P_k(\omega)$ vyjádřit vztahem:

$$P_k = \frac{Z_1}{Z_0} = \frac{\binom{\omega-1}{N-1}}{\binom{N+\omega-1}{N-1}}$$

Pro $\omega < N$ je pravděpodobnost P_k samozřejmě nulová. V případě, že ω je rovno N nebo je mírně vyšší, je pravděpodobnost nenulová avšak relativně nízká, jedná se však pouze o **hrubý odhad**. Pro vyšší hodnoty ω (a také nižší N) se však již odhad může blížit realitě. Závislost pravděpodobnosti P_k na ω pro různá malá N je zobrazena na grafu (2.13).



Obrázek 2.13.: Závislost P_k na počtu provedených atomických kontrol (ω)

Získaná hodnota pravděpodobnosti P_K umožňuje v prvé řadě stanovit dobu trvání cyklu samokontroly, jež by odpovídala předběžnému požadavku na důvěryhodnost samokontroly. Protože se však pravděpodobnost P_k v reálných systémech k hodnotě 1 pouze blíží, tj. existuje možnost, že v systému zůstanou nezkontrolované

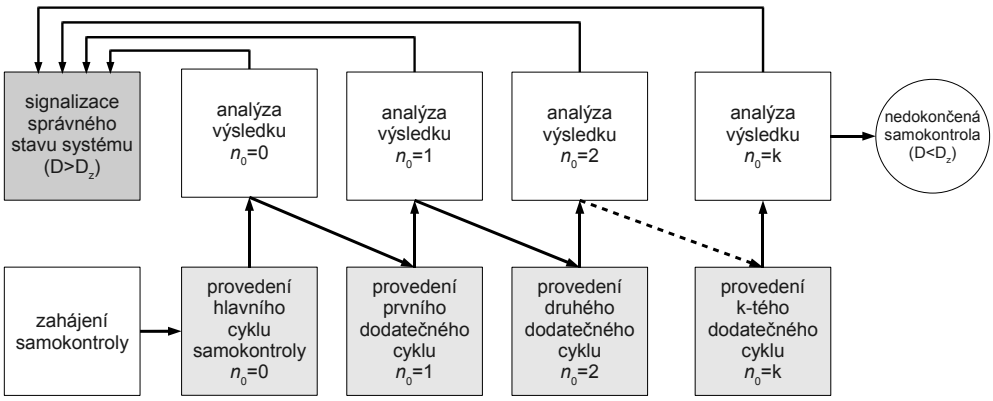
moduly, je možno uvažovat i o provedení dodatečné analýzy po skončení cyklu samokontroly. Tato analýza může striktně ověřit, zda byly všechny moduly zkontrolovány, může však zbytečně zatěžovat systém.

2.5. Organizace samokontroly s provedením analýzy

Dodatečná analýza se používá v případech, kdy se chceme vyhnout situaci, kdy nezkontrolovaný modul šíří nesprávné výsledky. Analýza kontroly modulů může být jak jednoduchá tak velmi složitá — podle požadované jistoty, že každý modul je alespoň jednou zkontrolován. To má samozřejmě přímý vliv na důvěryhodnost výsledků samokontroly.

Po ukončení procesu analýzy kontroly modelů lze v případě nevyhovujícího výsledku, který neodpovídá stanoveným požadavkům D_z, pokračovat v samokontrole, tj. spustit další (dodatečný) cyklus samokontroly. V rámci tohoto dodatečného cyklu samokontroly lze přirozeně využít (započítat) i kontroly provedené v hlavním cyklu.Dodatečný cyklus je následován novou analýzou kontroly. Obecně může být provedeno několik za sebou následujících dodatečných cyklů. Každý z nich je následován analýzou kontroly modulů.

Proces je znázorněn na obrázku 2.14 ve formě přechodového grafu.



Obrázek 2.14.: Proces samokontroly s analýzou a dodatečnými cykly

Proces samokontroly má v tomto případě několik dílčích podstavů, konkrétně:

- začátek samokontroly
- provedení prvního (hlavního) cyklu samokontroly ($n_0 = 0$)

- analýza výsledku provedení hlavního cyklu
  - je-li ověřena dostatečná důvěryhodnost, signalizace správného stavu systému
- provedení prvního dodatečného cyklu samokontroly ($n_0 = 1$)
- analýza výsledku provedení prvního dodatečného cyklu
  - je-li ověřena dostatečná důvěryhodnost, signalizace správného stavu systému

$$\Downarrow$$

- provedení k-tého dodatečného cyklu samokontroly ($n_0 = k$)
- analýza výsledku provedení k-tého dodatečného cyklu
  - je-li ověřena dostatečná důvěryhodnost signalizace správného stavu
  - jinak: výjimečná situace, samokontrola nebyla dokončena. Do prostředí systému lze signalizovat buď výjimečný stav nebo správný stav systému s dodatečnou informací o snížené důvěryhodnosti (tuto úroveň důvěryhodnosti je možno v rámci analýzy spočítat).

Hlavní problém výše navrženého procesu spočívá jednak ve stanovení optimální doby provádění hlavního cyklu a cyklů dodatečných, jednak ve stanovení optimálního počtu dodatečných cyklů. Na jedné straně je zřejmé, že čím je větší počet prováděných atomických kontrol, tím je větší důvěryhodnost výsledku samokontroly. Z tohoto pohledu by bylo vhodné maximálně zvýšit dobu provedení hlavního cyklu samokontroly. Na druhou stranu prodloužení cyklu samokontroly znamená prodloužení intervalu mezi signalizacemi správného stavu systému. To může být nevyhovující či dokonce nežádoucí pro mnohé reálné systémy. Nezbývá než hledat kompromis.

S podobnými situacemi se můžeme setkat i v běžném životě.

Například, pokud se řidič potřebuje dostat z města do vesnice při minimalizaci ceny za benzín, musí hledat kompromis mezi zakoupením benzínu ve městě (bývá levnější) a na venkově, kde je dražší. Základní strategií je zakoupení veškerého benzínu ve městě. Bohužel díky uzavírkám a objížďkám, resp. obecně dopravní situaci, nemůže řidič odhadnout přesnou spotřebu benzínu (koupení nadbytečného benzínu zvýší cenu). Je zřejmé, že pro rozhodnutí řidiče je důležitý poměr cen ve městě a na venkově.

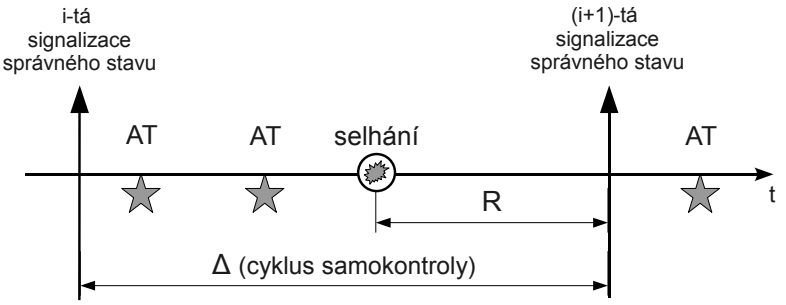
Podobně se musíme rozhodovat, zda zakoupíme u mobilního operátora výhodný balíček volných minut, nebo zda budeme platit běžné (vyšší) minutové sazby. Rozhodnutí je dynamické, neboť předem nevíme, kolik provoláme minut, a i zde rozhodnutí závisí na poměru cen.

Je zřejmé, že cena benzínu ve městě resp. balíčku volných minut, je analogií doby provádění cyklů vlastní samokontroly a výrazně dražší benzín na venkově (resp. běžné minuty) odpovídá časově i výpočetně náročnější analýze.

Pro řešení daného problému může být použit aparát *Markovských procesů*. V daném případě jsou procedury provedení hlavního a všech dodatečných cyklů a analýzy kontroly modulů representovány jako Markovský řetěz s diskrétními stavy a nepřetržitým časem (přehledný popis problematiky viz [18] nebo [19]).

2.6. Důvěryhodnost výsledku samokontroly

Důvěryhodnost výsledku samokontroly je pravděpodobnost události, že v rámci samokontroly jsou odhaleny všechny nesprávné moduly. V předchozích kapitolách jsme pro zjednodušení předpokládali, že se tato pravděpodobnost rovná pravděpodobnosti provedení atomické kontroly všech modulů systému = P_k. Jak však již bylo naznačeno, je výpočet skutečné pravděpodobnosti náročnější. Nejdříve uvažujme událost (**B**) zohledňující potenciální selhání v nenulovém intervalu mezi provedením poslední atomické kontroly libovolného modulu M_j a signalizací správného stavu po provedení cyklu atomických kontrol (viz diagram 2.15). Pokud dojde k selhání v tomto intervalu, není v daném cyklu zachyceno (efekt je stejný, jako by modul nebyl vůbec kontrolován). Tato událost samozřejmě výrazně ovlivňuje (snižuje) důvěryhodnost výsledku.



Obrázek 2.15.: Selhání po provedení atomických kontrol v rámci CS

Jako Δ je na obrázku 2.15 označen interval mezi dvěma bezprostředně následujícími signalizacemi správného stavu systému. Hvězdičky označují jednotlivé atomické kontroly konkrétního modulu M_j. Lze snadno vidět, že mezi vznikem selhání a signalizací výsledku samokontroly (interval R) není modul M_j kontrolován.

Pravděpodobnost události B může být spočítána jako $P(B) = 1 - P(E)$, kde E je událost neprovedení atomické kontroly v intervalu R. Pro výpočet $P(E)$ stačí znát

pouze intenzitu atomických kontrol μ. Počet atomických kontrol provedených v rámci intervalu R odpovídá počtu nahodilých zotavení v rámci nahodilého intervalu τ, jenž se používá v teorii zotavení (*renewal theory*, [20]).

Podle této teorie je pravděpodobnost, že v intervalu τ (pro náš účel R) vznikne S obnovení (pro náš účel S atomických kontrol), určena generující funkcí:

$$G(z) = \int_0^\infty G(\tau, z) f_R \, \mathrm{d}\tau$$

kde $f_R(\tau)$ je funkce hustoty pravděpodobnosti náhodné veličiny R.

Například v případě konstantní intenzity selhání je funkce hustoty pravděpodobnosti náhodné veličiny R v intervalu Δ rovna:

$$f_R(\tau) = \begin{cases} \frac{1}{\Delta}, & \text{při } \tau \leq \Delta \\ 0, & \text{při } \tau > \Delta \end{cases}$$

Pro systém s velkým počtem modulů má náhodná veličina délky intervalu mezi dvěma sousedními atomickými kontrolami exponenciální rozdělení s parametrem μ. Pokud vezmeme v úvahu tyto předpoklady, pak má Laplaceova transformace generující funkce $G(\tau,z)$ následující tvar:

$$G(p, z) = \frac{1}{p + \mu(1 - z)}$$

a následná inverzní transformace umožňuje vyjádřit hledanou funkci $G(z)$:

$$G(z) = \frac{1 - e^{-\Delta\mu(1-z)}}{\Delta\mu(1 - z)} \tag{2.2}$$

Pravděpodobnost $P\{S = 0\}$, tj. pravděpodobnost události, že v intervalu R nebude provedena žádná atomická kontrola, může být zjištěna jako koeficient k_0 (při z_0) po rozepsání funkce $G(z)$ do polynomiálního tvaru $\sum_{s=0} k_s z^s$.

Hledanou pravděpodobnost tak lze vyjádřit následovně:

$$P(E) = P\{S = 0\} = \frac{1 - e^{-\Delta\mu}}{\Delta\mu}$$

Důvěryhodnost výsledku samokontroly však závisí i na míře, do jaké můžeme spoléhat, že atomická kontrola nesprávného modulu odhalí jeho selhání. Tato míra se

nejčastěji vyjadřuje jednoduchou pravděpodobností P_{AT}, která již byla popsána a uvažována v předchozí kapitole.

Zohlednění omezené důvěryhodnosti atomických kontrol připouští možnost, že v rámci intervalu R budou provedeny atomické kontroly, které však neodhalí selhání modulu - událost C.

Použití generující funkce (2.2) umožňuje vyjádřit pravděpodobnost$P\{S/S \neq 0\}$, tj. pravděpodobnost, že se v intervalu R vyskytne přesně s atomických kontrol, kde $s \neq 0$.

$$P\{S/S \neq 0\} = \frac{1}{s!} \left. \frac{\mathrm{d}^s G(z)}{\mathrm{d}z^s} \right|_{z=0}$$

Když započítáme, že atomická kontrola nemusí odhalit selhání modulu, lze pravděpodobnost neodhalení chybného modulu P_0 (= události B nebo C) vyjádřit jako:

$$P_0 = \frac{1 - e^{-\mu\Delta}}{\mu\Delta} + \sum_{s=1} \frac{1}{s!} \left. \frac{\mathrm{d}^s G(z)}{\mathrm{d}z^s} \right|_{z=0} (1 - P_{AT})^s$$

Vzhledem k tomu, že hodnota $(1 - P_{AT})^s$ pro $P_{AT} \approx 1$ s rostoucím s velmi rychle klesá, lze se při výpočtu omezit na malé hodnoty $s \in \{0, 1, 2\}$, tj. vztah lze upravit na:

$$P_0 = \frac{1 - e^{-\mu\Delta}}{\mu\Delta} + \frac{1 - (1 + \mu\Delta)e^{-\mu\Delta}}{\mu\Delta}(1 - P_{AT}) + \frac{2 - (2 + 2\mu\Delta + \mu^2\Delta^2)e^{-\mu\Delta}}{\mu\Delta}(1 - P_{AT})^2 \tag{2.3}$$

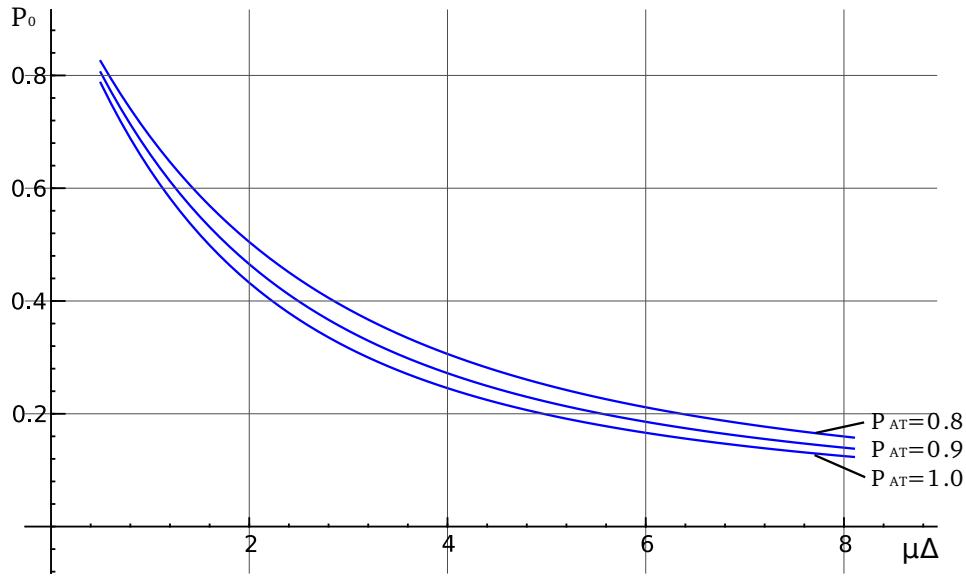
Funkcionální závislost P_0 na $\mu\Delta$ (tj. počtu atomických kontrol) pro různé hodnoty P_{AT} je znázorněna v grafu 2.16.

Celkovou důvěryhodnost v rámci daného cyklu samokontroly lze vyjádřit vztahem

$$P_D = P_K(1 - P_0)^N \tag{2.4}$$

kde jak P_K tak P_0 je závislé na počtu provedených atomických kontrol. Pro úspěšnou kontrolu je nutno, aby byly všechny moduly zkontrolovány (pravděpodobnost P_k) a zároveň nedošlo u žádného modulu k události B nebo C (pravděpodobnost P_0).

Na základě závislosti $P_0 = f(\mu\Delta)$ je možné doporučit takový interval Δ pro signalizaci výsledku samokontroly, který zajistí požadovanou pravděpodobnost P_0 za podmínky, že intenzita atomických kontrol je rovna hodnotě μ.



Obrázek 2.16.: Pravděpodobnosti neodhalení chybného modulu v závislosti na $\mu\Delta$ a P_{AT}

Popřípadě je možno na základě předem stanoveného intervalu Δ určit dostatečnou intenzitu atomických kontrol, která by zajistila požadovanou pravděpodobnost P_0. Intenzita provádění atomických kontrol může být zvýšena tím, že rozšíříme intervaly, v nichž modul neprovádí systémové činnosti a může tak provádět atomické kontroly (tj. intervaly β_i).

Výpočet pravděpodobnosti P_0 lze použít pouze v případě pevně daného intervalu Δ. Pokud je tato veličina náhodná (tak je tomu u schémat, u nichž není čas provedení atomických kontrol předem definován), je výpočet výrazně komplikovanější.

2.7. Schémata pro organizace samokontroly systému

Kromě stanovení trvání cyklů samokontroly jsou pro celkovou organizaci samokontroly klíčové i struktura informací, na jejichž základě se provádí analýza kontroly a na nichž je také založen mechanismus stanovení modulu (diagnostického jádra) odpovědného za tuto analýzu. Vstupní informace pro analýzu může být buď jednoduchá, například údaj o počtu provedených kontrol, nebo komplexní, například detailní informace o struktuře atomických kontrol.

Z tohoto hlediska mohou být uvažována následující obecná schémata:

Schéma 1

V průběhu předem stanoveného času moduly provádějí navzájem atomické kontroly. Moduly si **nevyměňují žádné informace**. Po uplynutí předem stanoveného času jeden z modulů (může to být kterýkoliv modul) signalizuje do prostředí informaci o správném stavu systému.

Schéma 2

V průběhu předem definovaného času moduly provádějí navzájem atomické kontroly. Každý modul si spočítá celkový počet atomických kontrol, jichž se zúčastnil. Po uplynutí předem stanoveného času si moduly tato data vyměňují. Každý modul provádí vlastní analýzu, která spočívá v porovnání vypočtené hodnoty celkového počtu provedených atomických kontrol s určitou předem stanovenou hodnotou. Tato hodnota je stanovena na základě požadované důvěryhodnosti výsledku samokontroly. V případě, kdy podmínka je splněna, modul signalizuje správný stav systému. Když ani v jednom z modulů není splněna podmínka, budou atomické kontroly modulů pokračovat, a to po předem určenou dobu. Po uplynutí této doby si moduly opět vyměňují data o počtu provedených atomických kontrol a provádí se výše uvedená analýza. Toto opakování může proběhnout vícekrát, dokud jeden z modulů nesignalizuje správný stav systému nebo nastane situace, že bude signalizována informace o kontrole s důvěryhodností menší, než je požadovaná.

Schéma 3

Organizace samokontroly systému v tomto případě je velmi podobná té ve schématu 2. Jediný rozdíl spočívá v tom, že si moduly vyměňují informace ohledně DG systému, což umožňuje přesněji spočítat důvěryhodnost výsledku samokontroly.

Podobně je možno uvažovat i organizace samokontroly, ve kterých není čas provádění atomických kontrol předem definován. To znamená, že okamžik, kdy je výsledek samokontroly signalizován do prostředí systému, je nahodilý. Za této situace je nutno stanovit horní meze pro čas, po jejíž dosažení bude signalizován stav systému do prostředí (nezáleží na důvěryhodnosti výsledku samokontroly).

V průběhu provádění atomických kontrol si moduly vyměňují informace o DG, kterou mají k dispozici, o diagnostickém systému. Každý modul neprovádějící systémové úkoly analyzuje DG systému a zjišťuje, zda jsou všechny moduly zkontrolovány. Pokud ano, modul okamžitě signalizuje stav systému do prostředí. Realizace

takové organizace samokontroly vyžaduje vyřešení několika dalších problémů, například jak provést synchronizaci modulů po signalizaci stavu systému, aby bylo provedeno obnovení lokálních dat modulů. Informace o DG systému, kterou si moduly vyměňují, může být jak velmi jednoduchá (např. mohou být použity pouze kódové invarianty DG) tak i složitá (kompletní DG systému), což ovlivní čas potřebný pro analýzu.

Ve všech uvedených schématech a organizacích samokontroly s putujícím jádrem však nastává principiálně zcela odlišná situace v případě, že libovolná atomická kontrola vrátí hodnotu, jež svědčí o existenci **nesprávného modulu** (tj. v Preparatově representaci hodnotu „1“). V tomto případě je **samokontrola bezprostředně ukončena** a nastává fáze samodiagnostiky.

2.8. Organizace samodiagnostiky systému

Jak již bylo uvedeno výše, organizace s putujícím jádrem striktně odděluje proceduru samokontroly a samodiagnostiky. Toto oddělení je důsledkem přístupu, při němž se kontrola stavu systému a případná diagnostika provádí v průběhu běhu systému, konkrétně v periodách, kdy moduly neprovádí systémové úkoly. Jinak řečeno pro provedení samokontroly a samodiagnostiky nejsou přiděleny zvláštní časové intervaly.

Pokud bychom však pro samodiagnostiku použili stejnou strategii, jakou jsme navrhli pro samokontrolu, nezabránili bychom nesprávnému modulu, tj. modulu, který byl jako chybný detekován poslední atomickou kontrolou samokontroly, dále provádět systémové úlohy. Z tohoto důvodu používá samodiagnostika modifikovanou strategii, jejímž hlavním cílem je bezprostřední vyloučení nesprávného modulu z provádění systémových úloh.

Vzhledem k tomu, že na základě jediné atomické kontroly není možno stanovit, který ze dvou modulů je nesprávný (resp. mohou být nesprávné oba), je nutno pozastavit systémovou činnost obou modulů během trvání procedury samodiagnostiky a umožnit pouze jejich účast v provádění atomických kontrol. Oba moduly se označují jako *podezřelá skupina*, neboť existuje podezření na jejich selhání.

V závislosti na úrovni důvěryhodnosti výsledku samodiagnostiky a na čase, během něhož mohou být pozastaveny podezřelé moduly, může být uvažováno několik schémat organizace samodiagnostiky.

Schéma 1

Atomické kontroly provádí pouze moduly mimo podezřelou skupinu a kontrolovány jsou pouze podezřelé moduly. Počet atomických kontrol prováděných v rámci samodiagnostiky závisí na požadované důvěryhodnosti diagnostiky.

Schéma 2

Vzhledem k tomu, že existuje pravděpodobnost selhání modulu, který nepatří k podezřelé skupině (ať již dříve nebo během fáze diagnostiky), provádí se samodiagnostika, dokud nejsou zkontrolovány všechny moduly. To znamená, že se moduly, které nepatří k podezřelé skupině, kontrolují i navzájem. Jednou z variant tohoto schématu je provedení samodiagnostiky v situaci, kdy několik modulů selhalo najednou (rep. prakticky ve stejnou dobu). V tom případě mohou být použity diagnostické algoritmy popsané v předchozí kapitole. Této problematice se budeme věnovat detailněji níže.

Schéma 3

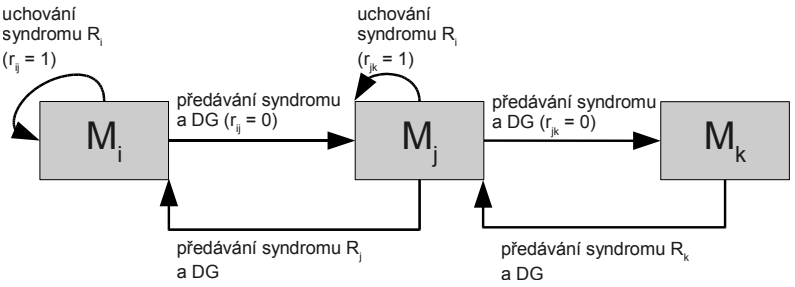
V případě, kde je čas provedení samodiagnostiky kritický (tj. je převažujícím kritériem), může být stanovena horní mez doby provedení samodiagnostiky. Pokud budou v průběhu provedení samodiagnostiky splněny podmínky uvažované ve schématech 1 a 2, bude procedura samodiagnostiky ukončena před stanovenou mezí. V opačném případě bude samodiagnostika ukončena po uplynutí předem stanoveného času a výsledek samodiagnostiky lokalizuje pouze podezřelou skupinu (tj. dva moduly z nichž alespoň jeden je nesprávný).

V případě, že v systému **selhalo několik modulů téměř současně** a je nutné provést úplnou diagnostiku, musí být ustanoveno diagnostické jádro.

Pro stanovení diagnostického jádra byla navržena metoda operativního předávání informací [21] viz obrázek 2.17 na následující straně.

Specifické rysy této metody lze shrnout následovně:

1. **podmínečný charakter předávání informace**. To znamená, že modul M_i předává informace modulu M_j pouze tehdy, pokud jej považuje za správný
2. **integrované předávání informace**, tj. předávány jsou jak informace o výsledcích atomických kontrol tak i o diagnostickém grafu systému
3. **vzájemné předávání**, tj. okamžitě po ukončení atomické kontroly si moduly, které byly angažovány v jejím provedení, navzájem vyměňují informace (i ve



Obrázek 2.17.: Operativní předávání informací

směru od kontrolovaného ke kontrolujícímu). Předání se však v souladu s bodem 1 neprovede v případě atomické kontroly s negativním výsledkem. To jest, je-li $r_{ij} = 1$, nepředává kontrolující modul M_i žádné informace kontrolovanému modulu M_j. Tím je zaručeno, že roli diagnostického jádra bude zajišťovat správný modul.

Každý modul neustále provádí rozbor obdržené informace a ověřuje, jestli je tato informace dostačující pro signalizaci výsledku samodiagnostiky do prostředí systému.

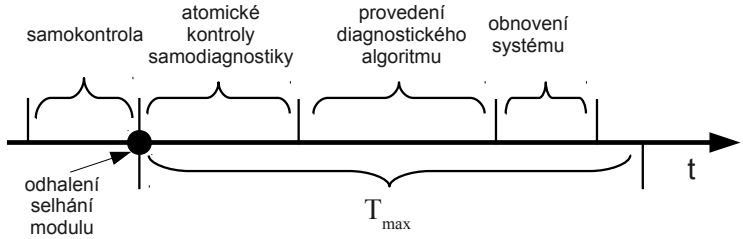
2.9. Selhání systému

Po ukončení procedury samodiagnostiky musí být provedeno obnovení systému. Obnovení systému lze provést buď na základě využití redundance (nadbytečnosti) nebo podle strategie postupné degradace systému.

První přístup spočívá v nahrazení chybného modulu (tj. modulu lokalizovaného ve fázi samodiagnostiky) správným (náhradním) modulem, jenž musí být k dispozici a musí bezprostředně převzít úlohu selhavšího modulu. Druhý přístup izoluje chybný modul v rámci systému, tj. modul již nevykonává žádné systémové úkoly. Tento přístup vyžaduje přerozdělení úkolů na zbývající (správné) moduly. Systémové úkoly jsou tak prováděny menším počtem modulů, služba poskytovaná systémem se tak stává degradovanou (ať už v kontextu hodnotové nebo časové domény).

Důvěryhodný výsledek samokontroly však ještě neznamená, že systém bude skutečně úspěšně diagnostikován a obnoven, a bude tedy pokračovat v poskytování služby. Důvodem je skutečnost, že v systémech reálného času je *čas* vyhrazený na procedury samodiagnostiky a následující obnovení systému *omezený* (na obrázku 2.18 na následující straně je označen jako T_{max}). Překročení limitního času

může negativně ovlivnit provádění systémových úkolů, a tím pádem neakceptovatelně změnit i službu poskytovanou systémem, což je považováno za selhání systému.



Obrázek 2.18.: Časová omezení v systému

Při úvahách o selhání systému je nutné vzít v potaz i případ, kdy samokontrola chybný modul neodhalí, a ten bude pokračovat v systémové činnosti. Pravděpodobnost této události byla diskutována výše.

Existuje také možnost, že k selhání jednoho nebo více modulů může dojít ve fázi samodiagnostiky a obnovení systému. V případě samodiagnostiky bude selhání s vysokou pravděpodobností odhaleno během atomických kontrol samodiagnostiky. Výrazně komplikovanější je situace v případě fáze obnovení systému, ale pravděpodobnost takovéto události je velmi malá, neboť čas potřebný k obnově systému je relativně krátký.

Záleží na konkrétním systému, jak dlouho může zůstat modul neodhalený, a především, za jak dlouho se jeho chybná funkce projeví na úrovni služby poskytované celým systémem. V mnoha případech nevede chybný modul k okamžitému selhání systému. Pro každý konkrétní systém může být stanoven čas, který určuje maximální dobu, v níž musí být nesprávný modul odhalen (jinak řečeno maximální přípustnou dobu pro neodhalení selhání modulu). Systém může obvykle v rámci této doby provést několik cyklů samokontroly.

Vzhledem k uvažovaným organizacím samokontroly, samodiagnostiky a obnovení tak může být selhání systému přesněji definováno jako neodhalení nesprávného modulu během stanovené doby nebo nemožnost provedení obnovení po jeho odhalení z níže uvedených důvodů:

- vyčerpání redundance systému (v případě využití náhradních modulů)
- neakceptovatelná degradace systému (v případě použití redundance)
- na obnovení systému nezbývá čas (viz τ_{max} na obrázku 2.18).

V opačném případě můžeme říci, že systém odolal selhání jednoho nebo dokonce více modulů.

3. Model spolehlivosti systému

3.1. Zjednodušený model spolehlivosti systému

Odolnost systému proti selhání jednotlivých modulů lze modelovat prostřednictvím různých matematických modelů. Nejčastěji se při modelování používá graf přechodů mezi jednotlivými stavy systému (vizualizovatelný jako přechodový resp. stavový digram).

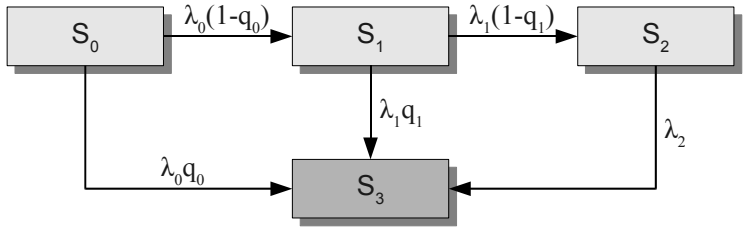
Grafový model umožňuje zjistit pravděpodobnost události, že se systém nachází v určitém stavu. Na základě těchto pravděpodobností lze následně pomocí konkrétních parametrů ohodnotit a kvantifikovat spolehlivost systému a jeho odolnost proti selhání modulů.

Pro systém s postupnou degradací je možné vyjít z modelu, jenž obsahuje následující čtyři stavy:

S_0	–	**aktivní jsou všechny moduly systému** (tj. N modulů). Aktivní moduly provádí systémové úkoly a zároveň jsou zapojeny do samodiagnostiky. Systém poskytuje správnou službu.
S_1	–	**jeden modul je izolován ze systému** (systém má $N-1$ aktivních modulů). Systém poskytuje správnou službu, jenž je však degradována.
S_2	–	**dva moduly jsou izolovány ze systému** (systém má $N-2$ aktivních modulů). Systém poskytuje správnou službu, jenž je však degradována.
S_3	–	**selhání systému**

Je zřejmé, že pro zjednodušení se uvažuje pouze **odolnost proti selhání nanejvýš dvou modulů**. Přechodový graf mezi jednotlivými stavy je znázorněn na obrázku 3.1 na následující straně.

Na grafu jsou pomocí symbolů $\lambda_0, \lambda_1, \lambda_2$ označeny intenzity přechodů systému z jednoho stavu do druhého a symboly q_0, q_1 označující pravděpodobnosti odpovídajících přechodů. Tyto parametry závisí jak na spolehlivosti jednotlivých modulů, tak na procedurách samokontroly, samodiagnostiky a obnovení.



Obrázek 3.1.: Základní model spolehlivosti systému

Parametry spolehlivosti jednotlivých modulů

Závislost spolehlivosti jednotlivých modulů na čase může být matematicky formalizována a tím kvantifikována pomocí

1. funkce
2. algoritmu
3. rovnic (diferenciálních)
4. systému rovnic

Nejrozšířenějším modelem spolehlivosti je statistický model využívající exponenciální rozdělení. Podle tohoto modelu je pravděpodobnost správného fungování modulu v čase t rovna:

$$P(t) = \mathrm{e}^{-\lambda t} \tag{3.1}$$

kde λ je parametr modelu.

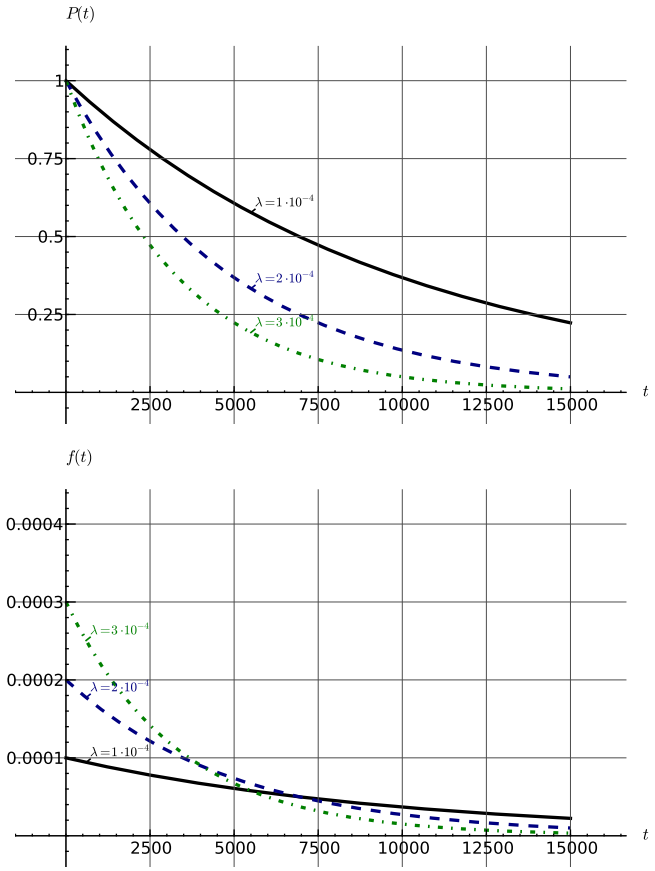
Funkce hustoty pravděpodobnosti doby do selhání má pro exponenciální rozdělení tvar:

$$f(t) = \frac{\mathrm{d}(P)}{\mathrm{d}t} = \lambda \mathrm{e}^{-\lambda t}$$

Funkce intenzity selhání je pak rovna:

$$\lambda(t) = \frac{f(t)}{P(t)} = \lambda = \mathrm{const}$$

tj. model exponenciálního rozdělení může být použit, pokud je intenzita selhání konstantní.



Obrázek 3.2.: Exponenciální rozdělení

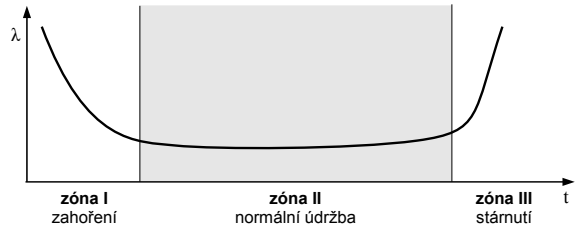
Funkce pravděpodobnosti, hustoty a intenzity spojené s exponenciálním rozdělením jsou pro tři ukázkové hodnoty parametru λ uvedeny na ilustračním obrázku 3.2.

Střední dobu do selhání lze v tomto modelu vyjádřit jako:

$$T_0 = \int_0^\infty t \cdot f(t)\,\mathrm{d}t = \frac{1}{\lambda}$$

U běžných modulů není intenzita selhání konstantní, ale má v grafické podobě tvar písmene U, viz obrázek 3.3 na následující straně.

Křivka intenzity má tři základní zóny. V zóně I se systém zahořuje (intenzita rychle klesá). Pak následuje dlouhé období normálního provozu a údržby (s téměř kon-



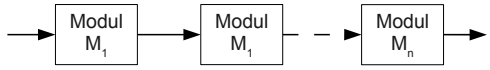
Obrázek 3.3.: Intenzita selhání modulu

stantní intenzitou). Poslední období reflektuje růst intenzity v závěrečném období stárnutí modulu. V našem modelu budeme předpokládat používání modulů nacházejících se ve druhé oblasti tj. předpokládat konstantní intenzitu a exponenciální rozdělení.

Kvantitativní popis spolehlivosti jednotlivých modulů nám umožní přejít ke kvantifikaci spolehlivosti celého systému.

Schémata pro výpočet spolehlivosti systému

Z hlediska spolehlivosti systému existují dvě základní schémata zapojení (resp. propojení) komponent systému, a to sekvenční (sériové) a paralelní. Tato schémata lze dále kombinovat.



Obrázek 3.4.: Sekvenční zapojení modulů

V případě sekvenčního zapojení modulů (viz obrázek 3.4) vede selhání jednoho modulu k selhání celého systému. Za předpokladu nezávislosti selhání komponent (selhání jednoho modulu neovlivňuje stav jiného) je pravděpodobnost správného fungování systému rovna:

$$P_S = \prod_{i=1}^{N} P_i \tag{3.2}$$

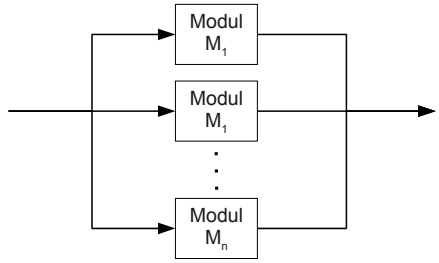
kde P_i – pravděpodobnost správného fungování i-tého modulu
N – počet komponent v systému.

V případě exponenciálního modelu selhání jednotlivých komponent lze pravděpodobnost vyjádřit s pomocí vztahů (3.1) a (3.4) takto:

$$P_s(t) = e^{-\sum_{i=1}^{N} \lambda_i t}$$

a v případě, že je intenzita selhání jednotlivých modulů shodná, pak ve zjednodušeném tvaru ($\forall i : \lambda = \lambda_i$):

$$P_S(t) = e^{-N\lambda t} \tag{3.3}$$



Obrázek 3.5.: Paralelní zapojení modulů

V případě paralelního zapojení systém selže pouze v případě, že selžou všechny jeho moduly (obr. 3.5). V tomto případě je pravděpodobnost správného fungování systému rovna:

$$P_S = 1 - \prod_{i=1}^{N} (1 - P_i)$$

Pro moduly s exponenciálním modelem spolehlivosti a při shodné intenzitě selhání modulů získáme vztah:

$$P_S(t) = 1 - \prod_{i=1}^{N} (1 - e^{-\lambda t})$$

a pro střední hodnotu doby do selhání systému vztah

$$T_0 = \int_0^{\infty} P_s(t)\, dt = \frac{1}{\lambda} \sum_{i=1}^{N} \frac{1}{i}$$

Uvedené parametry spolehlivosti mohou být využity pro zjištění intenzit přechodů λ_0, λ_1 a λ_2 v našem grafovém modelu.

Vzhledem k tomu, že procedury odhalení selhání a obnovení systému jsou zanedbatelně krátké vůči střední době selhání modulu (milisekundy vers. stovky hodin), nemusíme tento čas zohledňovat a tak předpokládat, že systém opustí stav S_0 bezprostředně po selhání modulu. Navíc můžeme pro stav S_0 předpokládat sekvenční zapojení modulů, protože selhání kteréhokoliv modulu vede k opuštění stavu S_0. Intenzitu opuštění stavu S_0 tak lze zjistit ze vztahu (3.3):

$$\lambda_0 = N\lambda$$

kde λ je intenzita selhání jednoho (libovolného) modulu.

Podobné předpoklady platí i pro stav S_1, v němž však systém obsahuje jen $N-1$ aktivních modulů:

$$\lambda_1 = (N-1)\lambda$$

Obdobný vztah lze využít i pro stav S_2 pro systém s $N-2$ aktivními moduly:

$$\lambda_2 = (N-2)\lambda$$

Pravděpodobnosti přechodů q_0 a q_1 závisí na organizaci samodiagnostiky a především její efektivitě. Tyto pravděpodobnosti budou diskutovány v následující podkapitole. Prozatím budeme uvažovat dva typické příklady.

Pokud je $q_0 = q_1 = 0$, pak nedochází k předčasnému selhání systému, tj. systém je vždy úspěšně obnoven, pokud je počet chybných modulů≤ 2. V reálných systémech však jsou pravděpodobnosti nenulové a q_1 je typicky vyšší než q_0, neboť diagnostika a obnova v degradovaném systému je náročnější.

Poissonovský proces přechodů mezi stavy

Přechody mezi jednotlivými stavy v našem zjednodušeném modelu mohou být uvažovány jako **poissonovské procesy**.

Poissonovské procesy umožňují modelovat širokou třídu přírodních a technických procesů, a to především v případě, kdy je pozorován společný efekt většího počtu nezávislých individuálních procesů.

Například ve článku [22] ukázal Fray, že poissonovský proces popisuje počet vojáků v armádě, kteří byli zabiti kopyty svých koní. Existují však i praktičtější příklady, jako například sekvence intervalů mezi voláními v telefonní síti nebo ještě aktuálnější příklad časové rozdělení požadavků na internetový server.

Pro náš model je důležité, že Palm [23] a Chinčin [24] ukázali, že se v mnoha případech suma velkého počtu stacionárních obnovení (z nichž každý může mít libovolné rozdělení času obnovení) blíží k poissonovskému procesu.

Zde je také nutno připomenout souvislost mezi poissonovskými procesy a exponenciálním rozdělením. U poissonovských procesů mají časové intervaly mezi přechody systému z jednoho stavu do druhého exponenciální rozdělení.

Jak již bylo řečeno v námi uvažovaném grafu přechodů (obr. 3.1) je čas setrvání systému ve stavech S_0, S_1 a S_2 ovlivněn především intenzitou vzniku selhání modulu. Čas setrvání systému v těchto stavech má proto exponenciální rozdělení, a tudíž lze přechody systému z jednoho stavu do druhého skutečně popisovat jako poissonovský proces.

Pravděpodobnost události, že se systém v kterýkoliv náhodně vybraný okamžik nachází v i-tém stavu, označme jako $P_i(t)$, $i = 0 \ldots 3$. Je zřejmé, že musí platit

$$\sum_{i=0}^{3} P_i(t) = 1 \tag{3.4}$$

Za předpokladu, že proces přechodu systému ze stavu do stavu je poissonovský, mohou být hledané hodnoty vypočítány ze soustavy Kolmogorovových diferenciálních rovnic (přehledně popsáno v [18]).

$$\begin{aligned}
\frac{\mathrm{d}P_0(t)}{\mathrm{d}t} &= -\lambda_0 P_0(t) \\
\frac{\mathrm{d}P_1(t)}{\mathrm{d}t} &= -\lambda_1 P_1(t) + \lambda_0(1-q_0)P_0(t) \\
\frac{\mathrm{d}P_2(t)}{\mathrm{d}t} &= -\lambda_2 P_2(t) + \lambda_1(1-q_1)P_1(t) \\
\frac{\mathrm{d}P_3(t)}{\mathrm{d}t} &= \lambda_0 q_0 P_0(t) + \lambda_1 q_1 P_1(t) + \lambda_2 P_2(t)
\end{aligned} \tag{3.5}$$

Kolmogorovova soustava diferenciálních rovnic popisuje dynamiku vstupu systému do určitého stavu resp. výstupu z určitého stavu. Například pro stav S_1 je dynamika vyjádřena pomocí diferenciální rovnice $\frac{\mathrm{d}P_1(t)}{\mathrm{d}t} = -\lambda_1 P_1(t) + \lambda_0(1-q_0)P_0$. To znamená, že systém vystupuje (proto znaménko minus) ze stavu S_1 s intenzitou λ_1 a vstupuje (plus) s intenzitou $\lambda_0(1-q_0)$, což je zároveň intenzita výstupu z S_0.

Stav S_0 je počáteční stav systému, tj. musí platit, že $P_0(t=0)=1$ (a samozřejmě i $P_i(t=0)=0$, pro každé $i=1\ldots3$).

Soustavu diferenciálních rovnic (3.5) převedeme Laplaceovou transformací na soustavu rovnic s komplexní neznámou s:

$$\begin{aligned} sP_0(s)-1 &= -\lambda_0 P_0(s) \\ sP_1(s) &= -\lambda_1 P_1(s)+\lambda_0(1-q_0)P_0(s) \\ sP_2(s) &= -\lambda_2 P_2(s)+\lambda_1(1-q_1)P_1(s) \\ sP_3(s) &= \lambda_0 q_0 P_0(s)+\lambda_1 q_1 P_1(s)+\lambda_2 P_2(s) \end{aligned} \tag{3.6}$$

Řešení systému rovnic (3.6) pro $P_i(s)$, $i=0,1,2$ je následující:

$$\begin{aligned} P_0(s) &= \frac{1}{s+\lambda_0} \\ P_1(s) &= \frac{\lambda_0(1-q_0)}{(s+\lambda_0)(s+\lambda_1)} \\ P_2(s) &= \frac{\lambda_0\lambda_1(1-q_0)(1-q_1)}{(s+\lambda_0)(s+\lambda_1)(s+\lambda_2)} \end{aligned}$$

Není těžké si všimnout, že jednotlivé rovnice lze vyjádřit ve tvaru:

$$P_i(s)=\frac{\prod_{j=0}^{i-1}(1-q_j)\lambda_j}{\prod_{j=0}^{i}(s+\lambda_j)}$$

Nyní lze využít následující vzorec pro inverzní Laplaceovu transformaci

$$Z(s)=\frac{1}{\prod_{i=1}^{m}(s+k_i)}\ \text{(obraz řešení)} \quad \longrightarrow$$

$$z(t)=\sum_{\substack{i=1\\ i\neq n}}^{m}\frac{e^{-k_i t}}{\prod_{\substack{n=0\\ n\neq i}}^{m}(k_n-k_i)},\ m\geq 1\ \text{(předmět zobrazení)}$$

Po jeho aplikování získáme následující výsledek ($P_3(t)$ je dopočítáno ze vztahu 3.4):

$$P_0(t) = e^{-\lambda_0 t}$$

$$P_1(t) = \frac{\lambda_0(1-q_0)}{\lambda_0-\lambda_1}(e^{-\lambda_1 t} - e^{-\lambda_0 t})$$

$$P_2(t) = \frac{\lambda_0\lambda_1(1-q_0)(1-q_1)}{(\lambda_0-\lambda_1)(\lambda_1-\lambda_2)(\lambda_0-\lambda_2)}(e^{-\lambda_2 t}\lambda_0 - \mathrm{e}^{-\lambda_2 t}\lambda_1 - \mathrm{e}^{-\lambda_1 t}\lambda_0 + \tag{3.7}$$

$$+ \mathrm{e}^{-\lambda_1 t}\lambda_2 + \mathrm{e}^{-\lambda_0 t}\lambda_1 - \mathrm{e}^{-\lambda_0 t}\lambda_2) \tag{3.8}$$

$$P_3(t) = 1 - \sum_{i=0}^{2} P_i(t)$$

Pravděpodobnosti výskytu systému v jednotlivých stavech S_0,S_1,S_2 a S_3 tj. $P_0(t)$, $P_1(t), P_2(t)$ a $P_3(t)$ jsou funkcemi času, jsou však závislé i na dalších parametrech. Přímo jsou závislé na pravděpodobnostech q_0 a q_1 a prostřednictvím intenzit λ_0, λ_1 a λ_2 na modelu selhání jednotlivých modulů a jejich schématu (tj. v našem případě nepřímo i na intenzitě selhání jednotlivého modulu λ a počtu modulů N).

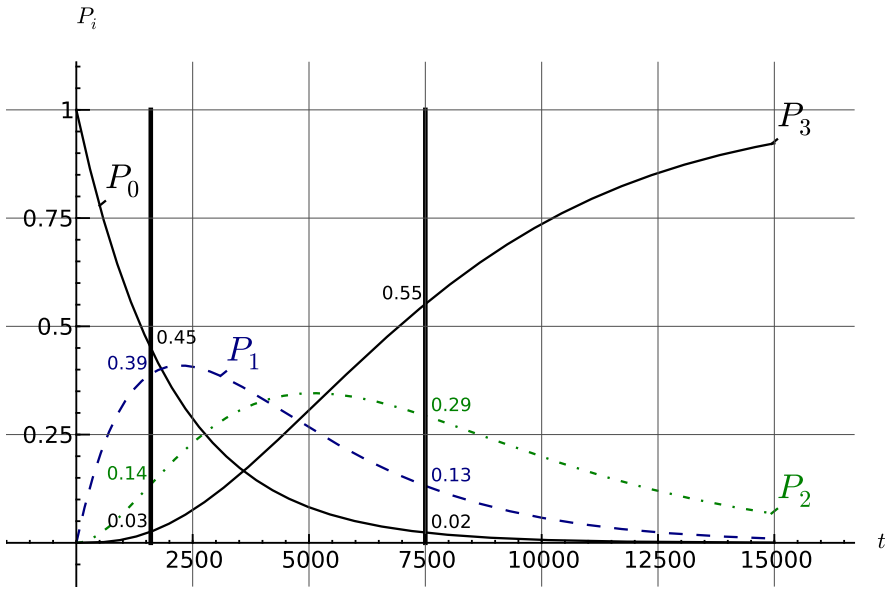
Funkční závislosti $P_i(t)$ na obrázcích 3.6 a 3.7 jsou spočítány pro ukázkový systém s pěti moduly $N = 5$ a intenzitou selhání modulu $\lambda = 10^{-4}$ (střední doba selhání modulu je tedy 10 000 hodin).

Obrázek 3.6 na následující straně ukazuje průběh pravděpodobností pro „ideální“ systém s $q_0 = q_1 = 0$, tj. systém s dokonalou diagnostikou a obnovením. Obrázek 3.7 pak obdobný průběh pro reálný systém s $q_0 = 0,2$ a $q_1 = 0,3$.

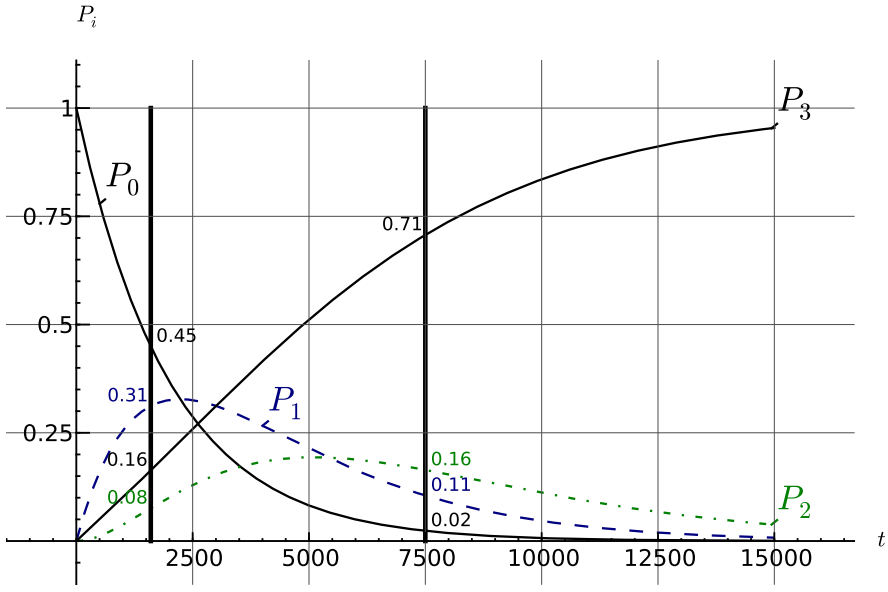
Pro dva časové okamžiky $t = 1600$ a $t = 7500$ jsou uvedeny i hodnoty pravděpodobnosti. Lze zřetelně vidět, jak zdokonalená samodiagnostika a obnovení vede k nižší pravděpodobnosti selhání systému. Platí to především pro počáteční fáze běhu systému, později se rozdíl stírá. V prvním znázorněném okamžiku je pravděpodobnost selhání u „ideálního“ systému pětkrát nižší, u druhého jen 1,3krát a v čase 15 000 hodin je rozdíl jen zcela minimální.

Tento charakteristický rys je ještě lépe vidět na obr. 3.8 na straně 103, který znázorňuje závislost pravděpodobnosti selhání $P_3(t)$ pro několik vybraných hodnot q (tj. pro zjednodušení jsou zohledněny jen shodné hodnoty q_0 a q_1 tj. $q = q_0 = q_1$). Pro hodnotu $q = 1.0$ je modelován systém bez možnosti obnovení (systém přechází ze stavu S_0 vždy bezprostředně do stavu S_3).

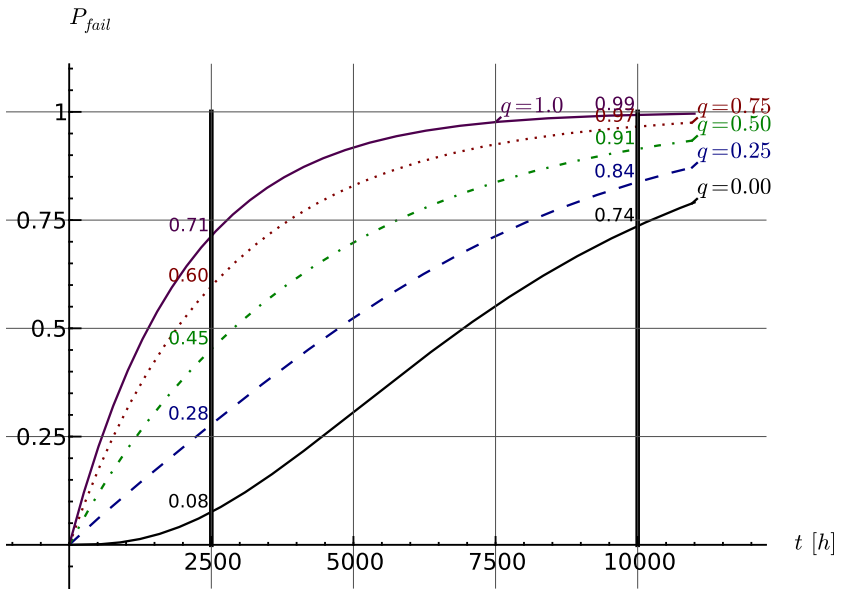
Při praktickém nasazení běží systémy jen relativně krátkou dobu bez možnosti zastavení a servisu (relativně vůči střední době selhání modulu například desítky nebo stovky hodin). Efekt samodiagnostiky a obnovení na spolehlivost systému je proto výrazný (na grafu oblast vlevo od výpisu pravděpodobností pro $t = 2500$).



Obrázek 3.6.: Pravděpodobnosti P_i(t) pro $q_0 = q_1 = 0$



Obrázek 3.7.: Pravděpodobnosti P_i(t) pro $q_0 = 0,2$ a $q_1 = 0,3$



Obrázek 3.8.: Pravděpodobnosti selhání v závislosti na t a param. $q = q_0 = q_1$

U složitějších systémů a modelů, je však konstrukce podobných grafů problematická, a proto se pro charakteristiku systému využívají jednodušší parametry (statistiky), které lze vypočíst na základě pravděpodobností (3.7).

Parametry spolehlivosti systému

Jedním z nejdůležitějších charakteristických parametrů je **střední doba do selhání systému** (*mean time to failure*), jež je označována jako T_0. Pro výpočet použijeme Tauberovu větu, podle níž lze T_0 vypočíst z výrazu:

$$T_0 = \lim_{s \to 0} P_\Sigma(s), \text{ kde } P_\Sigma(s) = \sum_{i=0}^{2} P_i(s)$$

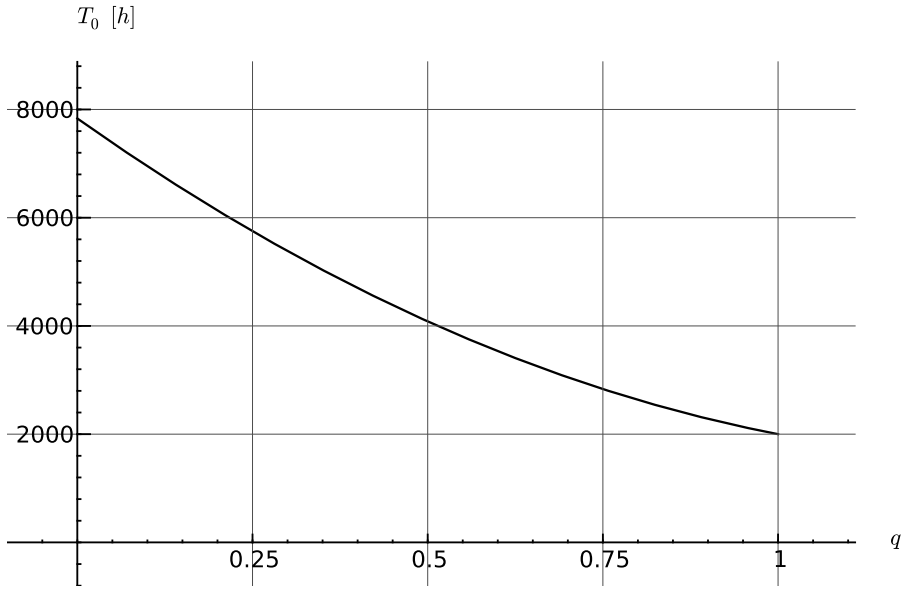
V našem případě:

$$T_0 = \sum_{i=0}^{2} P_i(s)\Big|_{s=0} = \frac{1}{\lambda_0} + \sum_{i=1}^{2} \frac{\prod_{k=0}^{i-1}(1-q_k)}{\lambda_i}$$

Tento vztah lze pro uvažovaný model dále zjednodušit:

$$T_0 = \frac{1}{\lambda_0} + \frac{(1-q_0)}{\lambda_1} + \frac{(1-q_0)(1-q_1)}{\lambda_2}$$

Na obrázku 3.9 jsou uvedeny hodnoty střední doby selhání systému v závislosti na $q = q_0 = q_1$ pro náš ukázkový systém ($N = 5$, $\lambda = 10^{-4}$). Z obrázku lze vidět, jak zdokonalení samokontroly a obnovení může ovlivnit střední dobu do selhání systému.



Obrázek 3.9.: Závislost T_0 na $q = q_0 = q_1$

Dalším parametrem spolehlivosti systému je **pravděpodobnost bezporuchového fungování systému v čase** t označená jako $P_B(t)$. Tuto pravděpodobnost lze vyjádřit jako součet pravděpodobností všech stavů, kromě stavu selhání, tj. přímo ze vztahů (3.7):

$$P_B(t) = \sum_{i=1}^{2} P_i(t)$$

Pro systém, který není schopen odolávat selhání jednotlivých modulů, vede opuštění stavu S_0 bezprostředně k selhání systému (přechod do stavu S_3). Z toho lze odvodit, že doba setrvávání systému ve stavech S_1 a S_2 odráží schopnost systému odolávat selhání jednotlivých modulů. Střední hodnotu tohoto času (T_ω) lze spočítat jako:

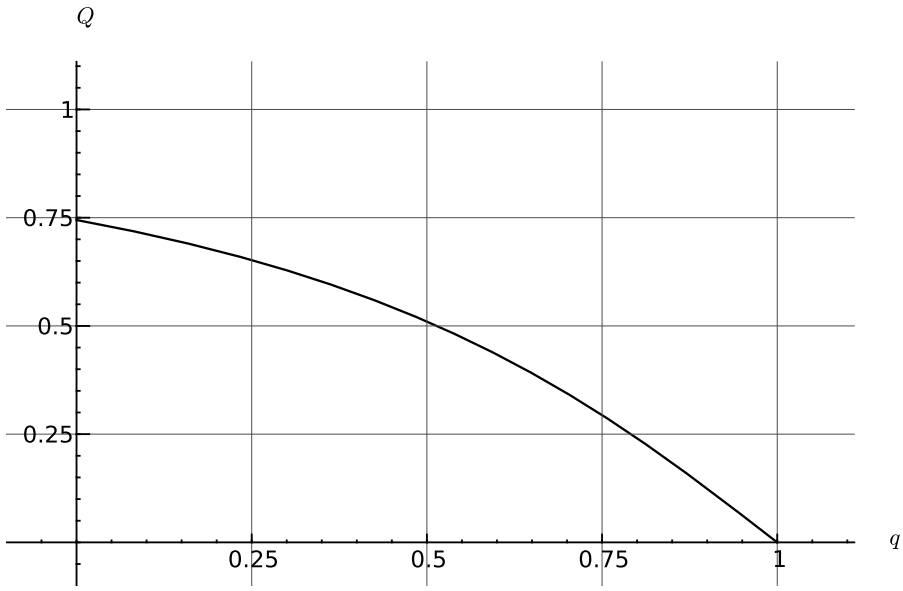
$$T_\omega = \lim_{s\to 0} P_\omega(s), \text{ kde } P_\omega(s) = \sum_{i=1}^{2} P_i(s)$$

Jako ukazatel odolnosti systému proti selhání se obvykle uvažuje i sumární počet nesprávných modulů, při nichž může být systém ještě schopen poskytovat akceptovatelnou službu. Jako další ukazatel odolnosti systému tak můžeme uvažovat i poměr:

$$Q = \frac{T\omega}{T_0}$$

Pro uvažovaný model se ukazatel Q rovná:

$$Q = \frac{R}{\frac{1}{\lambda_0} + R}, \text{ kde } R = \sum_{i=1}^{2} \frac{\prod_{k=0}^{i-1}(1-q_k)}{\lambda_i}$$



Obrázek 3.10.: Závislost ukazatele Q na $q = q_0 = q_1$

Závislost ukazatele Q na $q = q_0 = q_1$ je zobrazena na obrázku 3.10.

Pro ilustraci toho, jak ukazatel Q charakterizuje odolnost systému proti selhání modulů, uvažujme dva systémy. Předpokládejme, že oba systémy mají stejnou střední

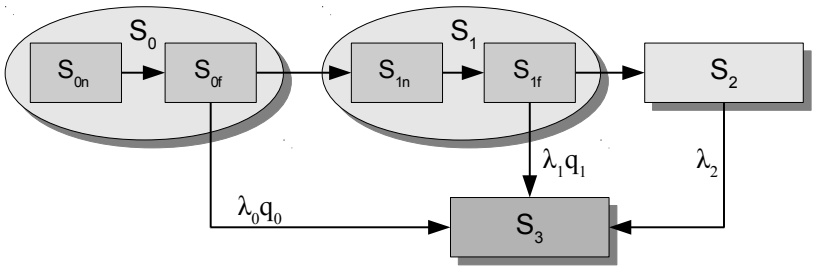
dobu do selhání, tj. $T_0^1 = T_0^2$. Pokud má například první systém poměr Q roven 0,2 a druhý 0,8, pak lze usuzovat, že první systém má spolehlivé moduly, ale méně spolehlivé prostředky samokontroly, samodiagnostiky a obnovení. Druhý systém má naopak méně spolehlivé moduly, ale o to více výkonnější prostředky samokontroly, samodiagnostiky a obnovení.

V případě, že $T_0^1 \neq T_0^2$, můžeme odolnost systému proti selhání modulů ohodnotit pomocí hodnoty T_ω, tento ukazatel však poskytuje jen hrubý odhad.

Zde je třeba zdůraznit, že výše uvažovaný zjednodušený model dává pouze *obecnou představu,* jak parametry spolehlivosti a odolnosti systému proti selhání závisí na procedurách odhalení selhání modulů a obnovení systému. Model obsahuje nespecifikované parametry (například pravděpodobnosti q_i) a nelze jej využít pro modelování konkrétních systémů, resp. za účelem predikce reálných hodnot parametrů spolehlivosti. Model také neumožňuje stanovit, do jaké míry jsou parametry systému závislé na organizaci procedur odhalení selhání modulů a jejich efektivitě. Lze jej však použít jako východiska zpřesněných modulů, které detailněji rozlišují ty stavy, v nichž probíhají procedury pro odhalení selhání a obnovení systému .

3.2. Zpřesněný model

Zpřesněný model spolehlivosti systému vychází z detailnější interpretace stavů S_0 a S_1, které uvažuje jako složené, a rozděluje je na dva (pod)stavy. Celkový pohled na tento zpřesněný model znázorňuje obrázek 3.11. Nově přidané stavy jsou označeny S_{0n}, S_{0f} a S_{1n}, S_{1f}.



Obrázek 3.11.: Zpřesněný model spolehlivosti systému

Jednotlivé stavy zpřesněného modelu mají následující význam:

S_{0n} – všechny moduly systému jsou správné. Počet aktivních modelů je roven N

S_{0f} – jeden modul systému je **nesprávný**. Počet aktivních modulů je roven N

S_{1n} – systém po obnovení. Poskytuje správnou ale degradovanou službu. Počet aktivních modulů je roven $N-1$

S_{1f} – jeden z $N-1$modulů je **nesprávný**

S_2 – systém po obnovení. Poskytuje správnou ale degradovanou službu. Počet aktivních modulů je roven $N-2$.

S_3 – selhání systému

Tento model spolehlivosti systému nicméně ještě detailně nezohledňuje, jak a kdy se provádí procedury odhalení selhání modulů a obnovení systému. Pouze předpokládá, že tyto procedury mají být provedeny v době, kdy se systém nachází buď ve stavu S_{0f} nebo S_{1f}. Je navíc možné (tj. existuje pravděpodobnost), že tyto procedury vůbec nebudou provedeny a systém po vzniku selhání modulu bezprostředně přejde do stavu selhání, resp. budou provedeny jen částečně (úspěšná diagnostika, ale neúspěšné obnovení).

Dále je možné obecně předpokládat, že čas pro odhalení selhání modulu a obnovení systému je omezený, tj. systém může setrvávat ve stavech S_{0f} nebo S_{1f} jen po předem určený maximální časový interval. Tento interval je dán schopností systému běžet s chybným aktivním modulem bez ovlivnění služby systému. Interval se označuje jako T_{max} a je ve většině případů velmi krátký. Platí tedy $T(S_{0f}) < T_{max} \wedge T(S_{1f}) < T_{max}$.

Model spolehlivosti systému, který by umožňoval počítačové modelování (simulaci), a tudíž i výpočet parametrů modelu (q_0 a q_1) v závislosti na čase provedení procedur odhalení a obnovení, však závisí především na konkrétní vnitřní organizaci těchto procedur.

Účel těchto procedur je zřejmý:

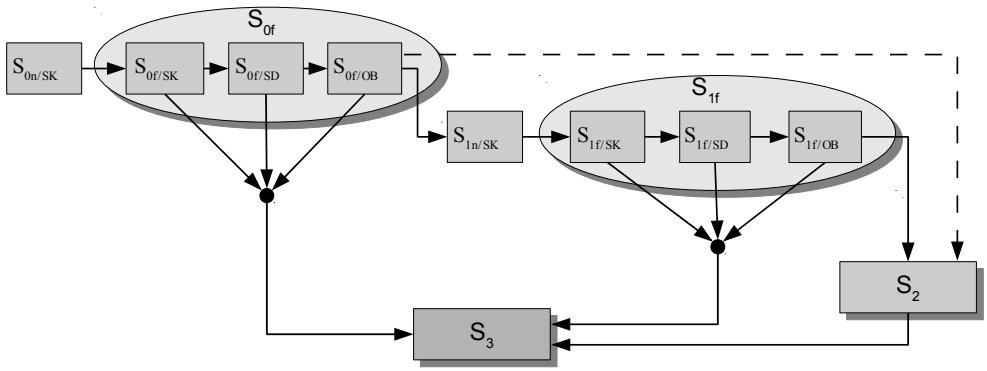
- odhalení události, že v systému vzniklo selhání jednoho z modulů (ale ještě není známo kterého)
- lokalizace nesprávného modulu, jinými slovy detekce (= nalezení) nesprávného modulu
- izolace nesprávného modulu a obnovení systému (převážně pomocí rekonfigurace systému a přerozdělení úkolů mezi správné moduly).

Dále budeme uvažovat pouze organizaci putujícího jádra, u níž jsou procedury odhalení a obnovy rozděleny do tří následných a v čase oddělených částí:

- samokontrola systému (probíhá v rámci běžné funkce systému) – zajišťuje odhalení události vzniku selhání modulu
- samodiagnostika – zajišťuje detekci chybného modulu

- obnovení systému

Na základě tohoto rozdělení lze navrhnout další zpřesnění našeho modelu (viz obrázek 3.12). Význam jednotlivých stavů je následující (model již obsahuje celkem 10 stavů):



Obrázek 3.12.: Model selhání pro systémy s putujícím jádrem

$S_{0n/SK}$ – všechny moduly systému jsou správné. Počet aktivních modulů je roven N. Probíhá samokontrola systému.

$S_{0f/SK}$ – jeden modul systému je **nesprávný**. Tato událost nebyla prozatím odhalena, tudíž probíhá stále samokontrola. Počet aktivních modulů je roven N.

$S_{0f/SD}$ – jeden modul systému je **nesprávný**. Tato událost již byla odhalena, ale chybný modul není lokalizován. Pro jeho odhalení běží procedura samodiagnostiky. Počet aktivních modulů je roven N.

$S_{0f/OB}$ – jeden modul systému je **nesprávný**. Modul byl již lokalizován a probíhá obnova systému. Počet aktivních modulů je stále roven N, ale po skončení bude chybný modul izolován.

$S_{1n/SK}$ – všechny aktivní moduly systému jsou správné. Počet aktivních modulů je roven $N-1$. Probíhá samokontrola systému.

$S_{1f/SK}$ – jeden aktivní modul systému je **nesprávný**. Tato událost nebyla prozatím odhalena, tudíž probíhá stále samokontrola. Počet aktivních modulů je roven $N-1$.

$S_{1f/SD}$ – jeden modul systému je **nesprávný**. Tato událost již byla odhalena, ale chybný modul není lokalizován. Pro jeho odhalení běží procedura samodiagnostiky. Počet aktivních modulů je roven $N-1$.

$S_{1f/OB}$ – jeden modul systému je **nesprávný**. Modul byl již lokalizován a probíhá obnova systému. Počet aktivních modulů je stále roven $N-1$, ale po skončení bude chybný modul izolován.

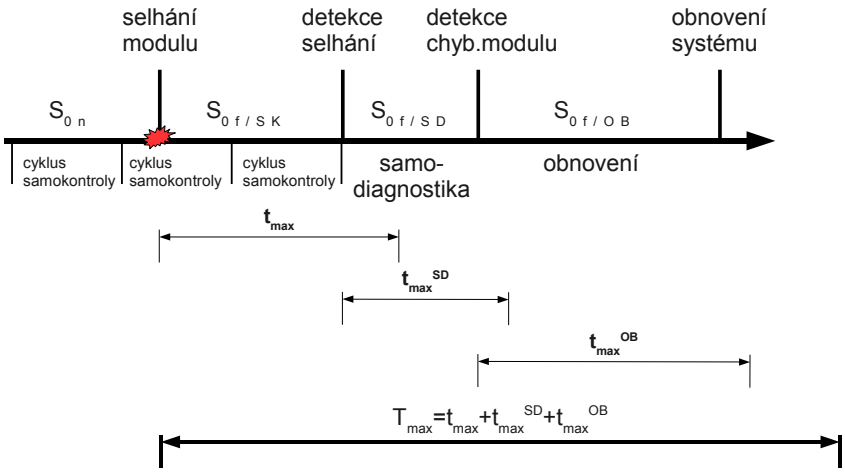
S_2 – systém po obnovení. Poskytuje správnou, ale degradovanou službu. Počet aktivních modulů je roven $N-2$

S_3 – selhání systému

Tento zpřesněný a upravený grafový model již detailněji odráží specifiku organizace samodiagnostiky využívající putující jádro. Zohledňuje například situaci, kdy čas vymezený na diagnostiku nestačí na přesnou lokalizaci modulu a výsledkem diagnostiky je pouze podezřelý pár. V tomto případě jsou izolovány zároveň oba podezřelé moduly a systém tak přejde ze stavu $S_{0f/OB}$ do stavu S_2(na obrázku 3.12 čárkovaná čára v pravém horním rohu). Podobně jsou zohledněna i výše diskutovaná časová omezení procedur odhalení a obnovy. Ze všech stavů s chybným modulem existuje přímý přechod do stavu selhání (S_3). Ve výjimečném případě nemusí být dokončena ani samokontrola (resp. celý cyklus samokontroly). Pokud je doba provedení cyklu samokontroly větší než T_{max}, pak systém může přejít přímo ze stavu $S_{0f/SK}$ nebo $S_{1f/SK}$ do stavu selhání. Nedokončení samodiagnostiky je representováno přechody ze stavů $S_{0f/SD}$ nebo $S_{1f/SD}$ do stavu S_3 (čas dokončení samokontroly + čas diagnostiky je větší než T_{max}). Předčasné selhání systému může nastat i ve stavech obnovy ($S_{0f/OB}$nebo $S_{1f/0B}$), pokud by celkový čas samokontroly, samodiagnostiky a obnovy přesáhl $T_{max.}$

Pro tento model můžeme navíc časový interval T_{max} rozdělit na dílčí intervaly t^{SK}_{max}, t^{SD}_{max} a t^{OB}_{max}. Čas t^{SK}_{max} je maximální možná doba od okamžiku selhání modulu po okamžik detekce selhání. Během tohoto intervalu probíhá samokontrola tj. i běžná činnost systému. Další intervaly omezují maximální trvání samodiagnostiky (t^{SD}_{max}) a obnovy (t^{OB}_{max}). Pokud je některý z dílčích intervalů překročen, přechází systém do stavu selhání. Obrázek 3.13 na následující straně ukazuje význam jednotlivých limitních časových intervalů. Průběh je znázorněn pro systém, který stihne provést všechny procedury samodiagnostiky a obnovení včas.

I takto zpřesněný model však nezohledňuje některé specifičtější charakteristiky organizace samokontroly a samodiagnostiky, které byly diskutovány v předchozí kapitole. Jedná se především o konkrétní schémata samokontroly a samodiagnostiky. V případě samokontroly bylo diskutováno jednoduché schéma s fixní dobou trvání cyklu samokontroly a rozšířené schéma s hlavním cyklem samokontroly doplněným analýzou DG, jež může být následována několika dodatečnými cykly (o zařazení dalšího cyklu rozhoduje analýza po ukončení každého cyklu). Výběr schématu samokontroly závisí na požadavcích na důvěryhodnost samokontroly, na parametrech systému (včetně hodnoty T_{max}, zatížení modulů, časový průběh atomických

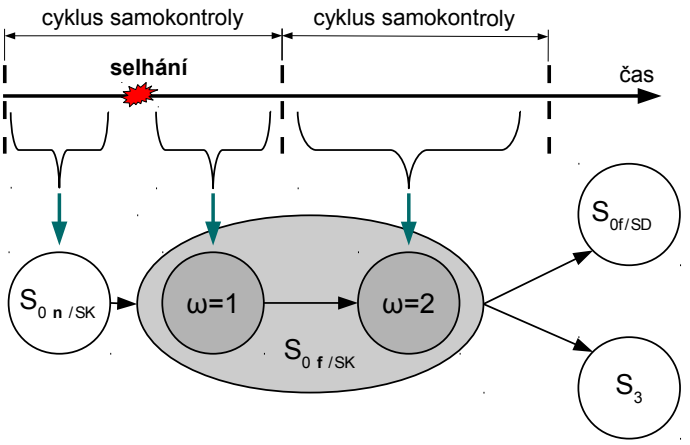


Obrázek 3.13.: Limitní časové intervaly samodiagnostiky a obnovy

kontrol). Zde však budeme pro zjednodušení uvažovat model s pevnou dobou trvání cyklu samokontroly (t_c) bez dodatečné analýzy. Toto schéma je popsáno v kapitole 2.4 na straně 75 (viz především obrázek 2.9).

Na základě hodnot T_{max} a t_c lze odhadnout, kolik cyklů samokontroly může být provedeno. Tento parametr označovaný jako ω není vlastností systému (na rozdíl např. od T_{max}), ale je parametrem modelu spolehlivosti a je využíván výhradně pro simulaci systému. Obrázek 3.14 graficky znázorňuje význam parametru ω a jeho spojitost s grafovým modelem spolehlivosti systému.

Obrázek se zaměřuje na stav $S_{0f/SK}$ původního modelu (obr. 3.12). V rámci tohoto stavu (tj. bezprostředně po vzniku selhání) může proběhnout několik cyklů samokontroly. V našem případě předpokládáme provedení dvou cyklů (obecně jich může být i jiný počet). Při dalším zpřesnění modelu je tak nutno zajistit identifikaci každého jednotlivého cyklu samokontroly, a to nejen ve stavu $S_{0f/SK}$, ale i ve stavech, které bezprostředně následují. Na obrázku je znázorněno pouze rozdělení stavu $S_{0f/SK}$ na dva stavy identifikované pomocí parametru ω (podle počtu proběhlých cyklů samokontroly). Toto rozdělení však ovlivňuje i následnou samodiagnostiku (stav $S_{0f/SD}$), která musí být také rozdělena na dva odlišné stavy, neboť časový průběh diagnostiky je ovlivněn počtem cyklů samokontroly. Rozdělení se projeví i ve stavu, v němž probíhá obnovení systému (zde $S_{0f/OB}$). Ten také musí být rozdělen na dva různé stavy, z nichž každý je opět jednoznačně identifikován parametrem ω.



Obrázek 3.14.: Model spolehlivosti systému a parametr ω

Toto rozdělení však výrazně komplikuje model, neboť hodnota parametru ω může obecně nabývat i hodnoty vyšší než dvě a může se kromě stavu S_{0f} projevovat i ve stavu S_{1f}. Z tohoto důvodu zavedeme *stavový prostor*, který stav procesu popisuje pomocí následující uspořádané n-tice:

$$\langle T_p, N_a, N_f, \omega \rangle$$

kde T_p = typ procedury popsaný libovolným symbolem z množiny sk, sd, ob)

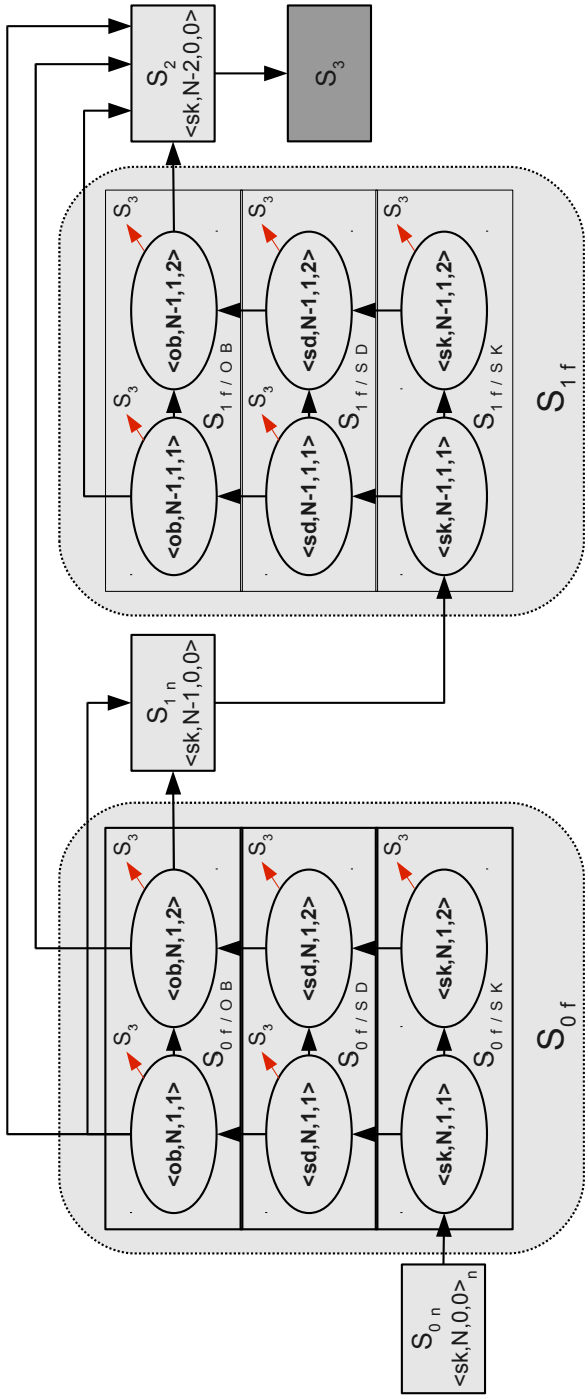
N_a = celkový počet aktivních modulů v systému

N_f = celkový počet chybných aktivních (tj. neodhalených) modulů v systému

ω = parametr identifikující cyklus samokontroly (v okamžik vzniku selhání modulu).

Pro zjednodušení (a v souhlase s předchozími modely) můžeme jednotlivé souřadnice stavového prostoru omezit takto: $N_a \in \{N, N-1, N-2\}$, $N_f \in \{0,1\}$ a $\omega \in \{1,2\}$. Mimo stavový prostor je stav selhání (nadále bude označován jako S_3). V rámci tohoto stavového prostoru můžeme nyní definovat detailní grafový model spolehlivosti systému (viz obrázek 3.15 na následující straně), který již lze využít pro simulaci systémů.

V uvažovaném modelu spolehlivosti se předpokládá, že samodiagnostika systému je prováděna podle schématu s pevně stanovenou maximální dobou provedení samodiagnostiky t_{max}^{SD}. Samodiagnostika tak může být buď úspěšně dokončena ještě



Obrázek 3.15.: Detailní model selhání

před vypršením intervalu t_{max}^{SD}, nebo je ukončena předčasně v čase t_{max}^{SD} , přičemž lokalizována je pouze dvouprvková množina podezřelých modulů. V další fázi jsou oba podezřelé moduly izolovány, což je na grafu vyjádřeno přímým přechodem ze stavu <ob,N,1,1>, resp <ob, N, 1,2>, do stavu S_2.

Po úspěšném provedení samodiagnostiky následuje obnovení systému. Kromě izolace nesprávného modulu (resp. modulů) musí procedura obnovení uvést systém do konzistentního stavu a určit (alokovat) prostředky zajišťující správnou službu systému. Vlastní mechanismus obnovy je závislý nejen na obecných parametrech systému, ale i na specifikaci požadované služby. Specifikace musí určovat nejen požadavky příslušné služby, ale také její maximálně akceptovatelnou degradaci.

V praxi mohou být při obnově použity následující postupy (použití závisí na charakteru služby):

- snížení počtu úkolů prováděných systémem (kvalitativní posun)
- použití zjednodušených resp. zkrácených algoritmů (poskytuje méně přesné, resp. méně důvěryhodné výsledky, kvalitativní posun)
- omezení výkonnosti systému (systém provádí úlohy pomaleji, kvantitativní posun)

Zvolený postup obnovy systému má i přímý vliv na další následné cykly samokontroly, popřípadě na další samodiagnostiku (po selhání modulu v degenerovaném systému). Je například možné, že po obnovení poklesne intenzita samokontrol, neboť v obnoveném systému je méně modulů, které jsou navíc ve většině případů více zatíženy. To vše se musí zohlednit i v rámci modelu spolehlivosti systému (např. v rozdílném popisu stavů $S_{0f/SK}$ a $S_{1f/SK}$).

Dalším modelovacím krokem již povětšinou bývá přechod k modelu konkrétního systému. Přechod od obecného (generického) modulu k modelu pro konkrétní systém vyžaduje kromě specifikace základních parametrů systému i detailní specifikace služeb poskytovaných systémem. Toto upřesnění je nezbytné především pro modelování fáze obnovy.

V případě stavů samokontroly však lze pokračovat v dalším zpřesňování obecného modelu. Zpřesněný model nám navíc poskytne další zajímavé výstupy použitelné v praxi.

3.3. Detailnější pohled na samokontrolu

Každý jednotlivý modul systému není a nemůže být kontrolován nepřetržitě. Proto vždy existuje nenulová pravděpodobnost, že modul může po svém selhání pokra-

čovat v provádění systémových úkolů, a tím změnit službu poskytovanou systémem. Výše bylo uvedeno, že pro každý systém lze stanovit dobu t_{max}^{SK}, během níž nedetekované selhání modulu ještě nevede ke změně služby systému, jinak řečeno změny jsou stále akceptovatelné. Pokud je tento čas překročen (např. selhání je odhaleno pozdě), jsou tyto změny neakceptovatelné a v některých případech i nevratné. Pravděpodobnost pozdního odhalení selhání lze odhadnout na základě metodiky uvedené v kapitole 2.4 na straně 75. Je to sice jen hrubý odhad, ale je zřejmé, že i tato pravděpodobnost je nenulová a pro reálné hodnoty t_{max}^{SK} blízké t_c i nezanedbatelná. Jaký vliv však má toto pozdní odhalení na stavový model spolehlivosti systému popsaný v předchozí podkapitole?

V prvé řadě je nutné zdůraznit rozdíl mezi pohledem, jež vychází z (viditelného) chování systému, a modelem, který modeluje přechody mezi stavy systému. Systém a ani vnější pozorovatel nepozná, že došlo k překročení času t_{max}^{SK} (a tudíž systém přešel do stavu S_3, neboť de facto selhal). Namísto toho systém pokračuje v samokontrole, v rámci níž může signalizovat správný stav, ačkoliv to není pravda. V dalším cyklu (cyklech) však může chybu zachytit a přejít na diagnostiku a obnovu. Pokud je však změna služby nevratná, nevede obnovení (z hlediska systému úspěšné) ke skutečně akceptovatelnému stavu. Vzniká tak rozpor mezi stavem modelovaným (stav S_3, tj. selhání systému, které je reálné a může být detekováno konzumentem služby) a chováním systému (systém se chová, jako by stále poskytoval správnou i když degradovanou službu). Rozpor je to však jen zdánlivý, neboť stav S_3 je v modelu definován pomocí akceptovatelnosti/neakceptovatelnosti poskytované služby, nikoliv jako funkce výstupu procedur samokontroly a samodiagnostiky.

Nejdříve se zaměřme na „falešné“ signalizace, které mohou nastat v průběhu času t_{max}^{SK}. Samokontrola v tomto případě signalizuje bezchybný stav modulů, přestože v systému již došlo k prozatím neodhalenému selhání modulu. Tyto falešné signalizace jsou prozatím tolerovány, a to jak samotným systémem, tak okolními systémy, které využívají jeho služby. Maximální počet tolerovatelných „falešných“ signalizací je možno určit na základě vztahu mezi intervaly t_{max}^{SK} a t_c. Tento počet určuje počet podstavů v rámci stavů samokontroly (tj. rozsah parametru ω ve stavovém prostoru stavů).

Například ve výše uvedeném detailním modelu, jenž předpokládá, že $t_{max}^{SK} \lesssim t_c$ (mírně menší nebo rovno), se nacházejí dva podstavy stavu $S_{0f/SK}$: $< sk, N, 1, 1 >$ a $< sk, N, 1, 2 >$ (viz obrázek 3.15 na straně 112). Systém tudíž toleruje pouze jedinou falešnou signalizaci. Při přechodu ze stavu $< sk, N, 1, 1 >$ do stavu $< sk, N, 1, 2 >$ je signalizován správný stav systému. Falešná signalizace může pokračovat i po uplynutí intervalu t_{max}^{SK}. Je však téměř jisté, že chybný modul bude nakonec detekován ($\lim_{t\to\infty} P_D(t) = 1$). Nelze stanovit, kdy k tomu dojde, ale při běžné úrovni důvě-

ryhodnosti samokontroly blížící se rychle k jedné, dojde k odhalení velmi brzy (tj. v několika málo dalších cyklech samokontroly). Systém je však už ve stavu selhání a následná diagnostika a obnovení nemůže tento stav změnit. Rozpor mezi skutečností (popsanou modelem) a stavem, jehož si je vědom systém, může být pouze dočasný, neboť diagnostika, resp. obnovení, se nemusí zdařit (například z důvodu nedostatku času), resp. trvalejší (pokud obnovení proběhne zdánlivě úspěšně).

Protože přechod ze stavu $S_{0f/SK}$ (resp. ze stavu $S_{1f/SK}$) do stavu S_3 vede k interně nedetekovanému selhání systému, je vhodné provést odhad pravděpodobnosti této situace v rámci našeho modelu.

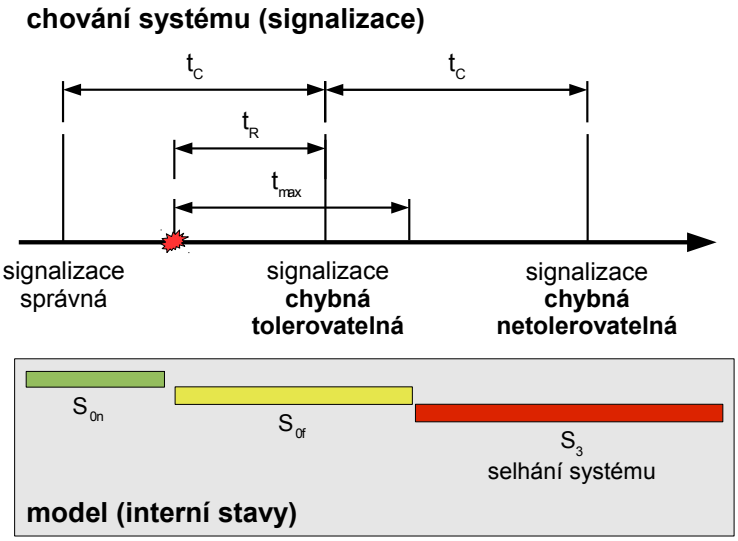
Ke vzniku selhání může dojít kdykoliv v průběhu cyklu samokontroly (předpokládá se rovnoměrné rozdělením pravděpodobnosti). K odhalení může dojít už v rámci prvního cyklu samokontroly, přesněji řečeno v jeho části počínající okamžikem selhání. Okamžik odhalení je podle zvoleného modelu opět náhodnou veličinou, jelikož náhodné jsou okamžiky atomických kontrol. Čas od vzniku selhání do ukončení cyklu označme jako t_R (viz interval R na obrázku 2.15 na straně 83).

Pokud je $t_R < t^{SK}_{max}$, může k detekci dojít i v rámci následujícího cyklu samokontroly (ω= 2). Tento cyklus je v tomto případě rozdělen mezi dva stavy modelu. Pokud k detekci dojde na počátku cyklu (v čase $t^{SK}_{max} - t_R$ od počátku druhého cyklu), pak systém podle modelu přechází do stavu $S_{0f/SD}$. Pokud dojde k detekci později, nebo vůbec (tj. cyklus končí falešnou signalizací), pak systém podle modelu selhal (viz obrázek 3.16 na následující straně)

Pro popis vnějšího chování systému je důležité zjistit pravděpodobnost první falešné signalizace po uplynutí intervalu t^{SK}_{max}, neboť touto signalizací se neshoda mezi skutečným stavem a pohledem systému poprvé projeví navenek (viz obrázek 3.16, kde je tato signalizace již v oblasti selhání systému).

Pravděpodobnost první falešné (netolerovatelné) signalizace lze odhadnout na základě vztahu pro pravděpodobnost $P_k(t)$, jenž byl odvozen v podkapitole 2.4 na straně 75. $P_k(t)$ zde vyjadřuje pravděpodobnost, že všechny moduly v systému byly alespoň jednou zkontrolovány během intervalu t. Pokud budeme pro zjednodušení předpokládat, že všechny atomické kontroly jsou plně důvěryhodné, pak je událost nezachycení chybného modulu v jistém intervalu t doplňkem události plné kontroly všech modulů v témže intervalu. Pravděpodobnost této události se proto rovná $1 - P_k$.

V případě pravděpodobnosti první falešné signalizace je nutno pravděpodobnost $1 - P_k$ vztáhnout k intervalu $t_A = t_R + t_c$, tj. k intervalu od vzniku události do první falešné signalizace. Délka intervalu t_A je však náhodnou veličinou, neboť závisí na náhodném čase t_R. Čas t_R může v našem modelu nabývat hodnot $\langle 0, t_c \rangle$ s rovnoměrným rozdělením pravděpodobnosti. Matematické vyjádření rozdělení pravdě-



Obrázek 3.16.: Časový průběh selhání systému (signalizace vers. model)

podobností hodnot $1-P_k(t_A)$ není jednoduché, neboť funkce $P_k(t)$ je relativně složitá. V praxi nám však postačí základní charakteristiky tohoto rozdělení, především střední hodnota, resp. rozptyl. Tyto charakteristiky lze s dostatečnou přesností odhadnout vytvořením modelu systému, a výpočtem hodnoty $1-P_k(t_A)$ pro náhodně zvolené časy z intervalu $\langle 0, t_c \rangle$ (s použitím generátoru náhodných čísel s rovnoměrným rozdělením), tj. využitím metody Monte-Carlo. Pro odhad střední hodnoty pak lze vypočíst aritmetický průměr náhodně získaných hodnot $1-P_k(t_A)$.

Pro vytvoření modelu jsme zvolili jazyk *Python*, neboť poskytuje potřebné vysokoúrovňové knihovny (elementární statistika a kombinatorika, operace s polynomy apod). Navíc výsledky bylo možno snadno převést do podoby grafů za využití softwaru *Sage*, jenž je na Pythonu založen a v němž byla vytvořena i většina dříve uvedených grafů. Výhodnou se ukázala i přímá podpora OOP, která umožnila výrazně zpřehlednit representaci modelu.

Zvolená representace modelu používá dvou základních tříd. Třída *Modul* popisuje modul systému, jenž je reprezentován dvojicí čísel $\overline{\alpha}$ resp. $\overline{\beta}$ tj. středními hodnotami času zaneprázdnění resp. zahálení modulu v rámci běžné funkce systému (viz obr. 2.10 na straně 76). Objekty třídy *System* pak representují celý systém, jenž je kromě seznamu modulů určen i fixním časem provedení jednotlivé atomické kontroly. Pro generování náhodných časů slouží pomocná třída *RandomTime*, jejíž objekty jsou parametrizovány časem t_c.

```
from scipy import poly1d, comb
import math
import random

class RandomTime:
    def __init__(self, tc):
        self.tc = tc

    def tc_tr(self):
        return self.tc + self.tc*random.random()

class Module:
    def __init__(self, alpha, beta):
        self.alpha = alpha
        self.beta = beta

class System:
    def __init__(self, modules, t_at):
        self.t_at = t_at
        self.m = modules
        self.n = len(modules)

    def Pat(self,i,t):
        return (self.m[i].beta + t - 2*self.t_at)/(self.m[i].alpha
                                                   + self.m[i].beta)

    def Qat(self,i,t):
        return 1-self.Pat(i,t)

    def r(self, t):
        poly = poly1d([1])
        for i in range(self.n):
            poly *= poly1d([self.Pat(i,t), self.Qat(i,t)])
        return sum(k*poly[2*k] for k in range(self.n / 2))

    def omega(self,t):
        smin = math.floor(t/(2*self.t_at)) + 1
        smax = math.ceil(t/self.t_at) - 1
        if smax < smin:
            raise ValueError("Not usable s for omega (%d, %d)" % (smin,smax))
        tquantum = t/smax
        return int(self.r(tquantum)*smax)

    def Pk(self, t):
        o = self.omega(t)
        n = self.n
        return comb(o-1, n-1)/comb(n+o-1, n-1)
```

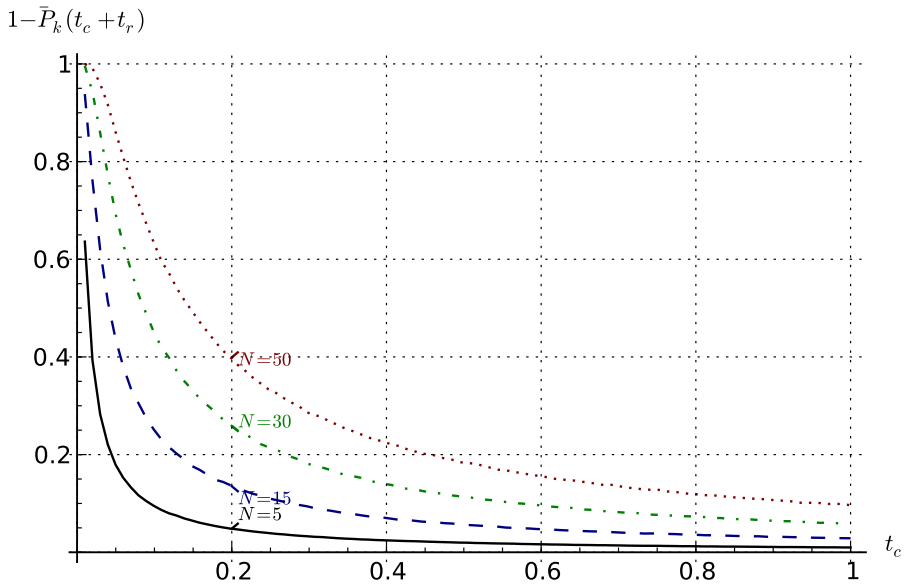
S využitím těchto tříd spolu s grafickými knihovnami rozšíření *Sage* je již snadné vytvořit model systému, vygenerovat náhodná data a zobrazit je ve formě grafu.

```
from model import System, Module, RandomTime
from graph import Graph, Function
from scipy import mean, std

class TcTrFuncProvider:
    def __init__(self, system): self.system = system
```

```
7     def MeanPk(self, tc):
8         generator = RandomTime(tc)
9         pks = [self.system.Pk(generator.tc_tr()) for i in range(1000)]
10        return 1-mean(pks)
11
12  graf = Graph(1e-2, 1.0, 0.0, 1.0, "t_c", r"1-\bar{P}_k(t_c+t_r)",
13               notesize = 0.8, points = 100)
14
15  for m in [5,15,30,50]:
16      modules = [Module(2e-3, 1e-3) for i in range(m)]
17      system = System(modules, 2e-4)
18      funcProvider = TcTrFuncProvider(system)
19      graf.add_function(Function(funcProvider.MeanPk, "N=%d" % (m)), xnote=0.2)
20
21  graf.show("tc_tr-plus.pdf")
```

Jádrem programu je cyklus na konci výpisu, v jehož rámci se do grafu přidávají funkce zobrazující závislost střední hodnoty veličiny $1-P_k(t_A = t_R + t_c)$ v závislosti na času t_c, a to pro ukázkové systémy s 5, 15, 30 a 50 identickými moduly, jejichž $\alpha = 2 \cdot 10^{-3} sec$ a $\beta = 10^{-3} sec$. Trvání jedné atomické kontroly je $t_{AT} = 2 \cdot 10^{-4} sec$.

Vlastní výpočet střední hodnoty $1-P_k(t_A)$ je prováděn v metodě *meanPk* třídy *TcTrFuncProvider*. Střední hodnota je odhadnuta jako průměr jednoho tisíce hodnot pravděpodobnosti $P_k(t_A)$ vypočítaných pro jeden tisíc náhodně zvolených intervalů t_R. Výsledný graf je na obrázku 3.17.



Obrázek 3.17.: Střední hodnoty pravděpodobnosti P_k v závislosti na t_c a N

4. Samokontrola a samodiagnostika v kontextu spolehlivosti a dependability

4.1. Spolehlivost

Spolehlivost může být definována jako *souhrnný termín používaný pro popis pohotovosti a činitelů, které ji ovlivňují: bezporuchovost, udržovatelnost a zajištěnost údržby* (ČSN IEC 50(191):1993).

V případě spolehlivosti hardwarového systému (může to být mechanický stroj, elektromechanický přístroj, apod.) je klíčové stanovení termínu údržby zařízení či intervalů jejich provozu. Například lze stanovit dobu provozu bez provádění kontrol nebo stanovit intervaly mezi jednotlivými kontrolami, resp. opravami tak, abychom vyhověli požadavkům na spolehlivost daného zařízení. V základní teorii spolehlivosti **se uvažují pouze vnější (externí) kontroly** nikoliv samokontrola.

Hodnocení spolehlivosti zařízení a následně i časový plán jeho provozu resp. údržby je možno stanovit na základě znalosti několika ukazatelů. Volba používaných ukazatelů spolehlivosti závisí na charakteru zařízení, především na tom, zda je zařízení opravitelné či nikoliv.

Neopravitelná zařízení

Pro neopravitelná zařízení se používají následující ukazatele spolehlivosti:

1. **Pravděpodobnost bezporuchového provozu** $P(t)$, jinak také označována jako **funkce spolehlivosti**.

   $$P(t) = \int_t^\infty \mathrm{f}(x)\,\mathrm{d}x$$

   kde $f(x)$ je hustota poruch.

2. **Střední doba bezporuchového provozu** T_s

$$T_s = \int_0^\infty t \cdot \mathrm{f}(x)\,\mathrm{d}x$$

3. **Intenzita poruch** λ

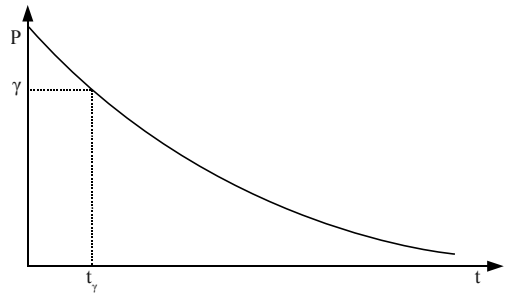
$$\lambda(t) = \frac{\mathrm{f}(t)}{P(t)}$$

V praxi lze intenzitu poruch odhadnout na základě statistiky ve tvaru:

$$\hat{\lambda}(t) = \frac{N(t, t+\triangle_t)}{N(t)\triangle_t}$$

kde $N(t, t+\triangle_t)$ je počet zařízení, u nichž se vyskytla porucha během intervalu $\langle t, t+\triangle_t\rangle$ (kde $\triangle_t$ je dostatečně malé), a $N(t)$ je počet jednotek zařízení, které zůstaly v provozu v okamžiku t.
Intenzita poruch je tedy rovna střednímu počtu poruch v časovém intervalu $\triangle_t$(začínajícím v čase t) vztaženému k počtu jednotek zařízení bez poruchy v čase t.

4. **Gama-procentní život** t_γ
Tento ukazatel se používá, pokud je známa funkce spolehlivosti $P(t)$.
Gama-procentní život je hodnotou inverzní funkce P^{-1} pro zvolenou hodnotu pravděpodobnosti bezporuchového provozu γ, tj. $t_\gamma = P^{-1}(\gamma)$. Jinak řečeno pro stanovenou hodnotu závisle proměnné P na ose Y stanovíme hodnotu nezávisle proměnné t na ose X (tj. abscisu). Viz obrázek 4.1.
Interpretace je zřejmá: *gama-procentní život t_γ je doba, v jejímž průběhu lze s pravděpodobností rovnou γ garantovat bezporuchový chod zařízení.* V praxi se hodnota γ volí z intervalu 0,8 až 0,99.



Obrázek 4.1.: Gama-procentní život

Opravitelná zařízení

Pro opravitelná zařízení se používají následující ukazatele spolehlivosti:

1. **Parametr proudu poruch** $\omega(t)$
   Parametr proudu poruch je počet poruch za určitý časový interval $\triangle_t$ vztažený k velikosti intervalu a počtu jednotek zařízení v systému (je roven počátečnímu počtu jednotek a v čase se nemění).
   $$\omega(t) = \frac{N(t, t+\triangle_t)}{N_0 \triangle_t}$$
   kdeN_0 je počet jednotek zařízení na počátku provozu systému.

2. **Střední doba provozu do poruchy** t_p
   Střední dobu lze vyjádřit jako poměr doby provozu opravitelného zařízení a očekávaného počtu poruch za tuto dobu.

3. **Součinitel pohotovosti** $k_g(t)$
   Tento součinitel se používá, pokud je nutno zohlednit nejen vlastní vznik (opakovaných) poruch, ale i čas nezbytný na opravu porouchaného zařízení. Součinitel pohotovosti je pravděpodobnost události, že v daném čase t bude zařízení ve správném stavu (s výjimkou period, v nichž je prováděna plánovaná údržba zařízení). Součinitel lze vyjádřit pomocí statistiky:
   $$\hat{k}_g(t) = \frac{N_s(t)}{N_0} \tag{4.1}$$
   kde $N_s(t)$ je počet zařízení, jež se v daném časovém okamžiku nacházejí ve správném stavu. Rozdíl $N_0 - N_s$ je roven počtu jednotek zařízení, která se v okamžiku t právě opravují.

4. **Součinitel ustálené pohotovosti** k_{ust}
   Tento součinitel se vypočítá jako limita součinitele pohotovosti (4.1) pro $t \to \infty$ nebo pomocí vztahu:
   $$k_{ust} = \frac{\sum t_p}{\sum t_p + \sum t_o}$$
   kde $\sum t_p$ je roven celkovému času provozu zařízení a $\sum t_o$ je celkový čas oprav.

5. **Koeficient operativní pohotovosti** $R(t)$
   Tento koeficient je roven pravděpodobnosti události, pro niž platí zároveň obě následující podmínky :

   a) zařízení bude v *libovolný okamžik* ve správném stavu (opět s výjimkou period plánované údržby zařízení)

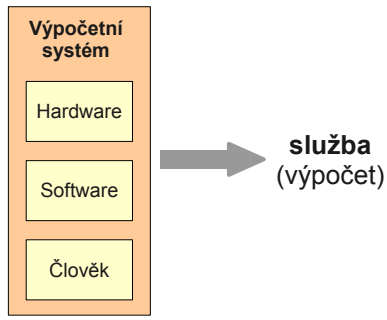
   b) počínaje od tohoto okamžiku bude zařízení fungovat po dobu minimálně t.

4.2. Dependabilita

Pojem spolehlivosti je však nedostatečný, pokud se budeme věnovat komplexnějším výpočetním zařízením, resp. se zaměříme na *služby,* které toto zařízení. respektive systém, poskytuje. Na úrovni služeb existuje obecnější pojem **dependabilní výpočet** (angl. *dependable computing*). „Dependabilnímu“ výpočtu či službě můžeme důvěřovat (v nejširším významu slova).

Zobecněná (potenciálně dependabilní) služba je prováděna a zajištěna prostřednictvím tří základních prvků (viz také obrázek 4.2):

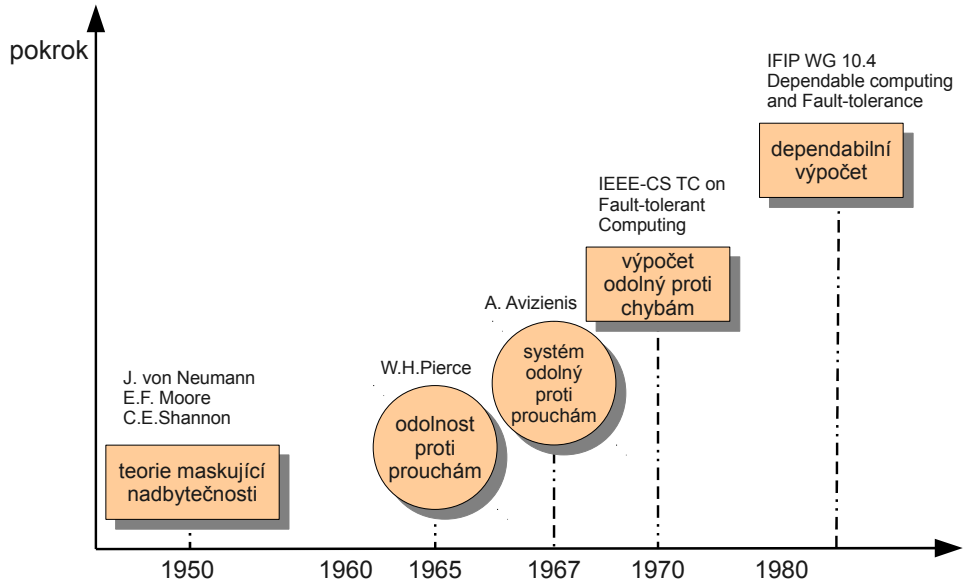
- hardware
- software
- člověk



Obrázek 4.2.: Výpočetní systém

V průběhu vývoje informačních technologií se role těchto tří základních prvků měnily, stejně jako se měnil jejich vliv na dependabilitu systému. Na počátku vývoje (tj. ve čtyřicátých a padesátých letech) převažoval vliv hardwarových zařízení, neboť poskytovaná služba závisela především na spolehlivosti hardwaru (spolehlivost hardwaru byla ve srovnání se současností výrazně nižší). V období let šedesátých až osmdesátých se proto pozornost zaměřovala především na snížení vlivu (ne)spolehlivosti jednotlivých hardwarových komponent na poskytovanou výpočetní službu. To se projevilo i v oblasti nových teorií a návrhů (viz obrázek 4.3 na následující straně, zdroje [16] (Moore, Shannon), [29] (Neumann) a [28](Avižienis)).

Zdokonalení technologií produkce hardwaru však již na konci tohoto období vedlo k výraznému růstu jeho spolehlivosti a také k rozšíření možnosti využití hardwarové redundance pro tvorbu hardwaru s větší odolností proti poruchám. Zároveň rostla složitost softwaru, čímž se zvýšil jeho vliv na spolehlivost výpočetní služby.



Obrázek 4.3.: Pokrok v oblasti dependability

Vznik počítačových sítí a následně především Internetu (v průběhu devadesátých let a na začátku nového století) pak akcentoval vliv lidského faktoru a především rozšířil požadavky na důvěryhodnost poskytované služby daleko za hranici klasické spolehlivosti.

Tato změna vedla ke vzniku nové koncepce pro hodnocení poskytovaných výpočetních služeb a ke vzniku nového pojmu: ***dependabilita*** (angl. *dependability*).

Jedna z prvních definic dependability má následující tvar [25]:

Dependabilita je schopnost výpočetního systému poskytovat službu, na niž se lze spolehnout.

Postupem času byla koncepce dependability hlouběji prostudována (především z pohledu praktického využití a kvantitativního hodnocení) a objevily se i další alternativní definice jako například [26] :

Dependabilita systému je schopnost systému vyhnout se takovým selháním, jejichž četnost výskytu, resp. závažnost, by byla větší než úroveň přípustná pro uživatele.

Formálně je dependabilita interpretována jako jeden z vlastností systému a má své samostatné specifikace a standardy. Na rozdíl od specifikací věnovaných funkčnosti a výkonnosti systému, stanovuje specifikace dependability požadavky ke každému jednotlivému atributu dependability. V současné době existuje *„Technický*

výbor 56 : Dependability“ v IEC (to jest *International Electrotechnical Commission*, www.iec.ch), který navrhuje a udržuje mezinárodní standardy v oboru dependability. Standardy navržené IEC TC 56 poskytují metody a nástroje pro hodnocení dependability a pro údržbu zařízení, služeb a systémů v průběhu jejich života.

Atributy dependability odrážejí jak kvantitativní tak kvalitativní hodnocení dependability systémů. V současnosti se uvádějí tyto primární atributy dependability:

1. **dostupnost** (*availability*): pohotovost k provedení korektní služby
2. **spolehlivost** (*reliability*): kontinuita poskytování korektní služby
3. **zabezpečení** (*safety*): absence katastrofických následků pro uživatele a prostředí
4. **důvěrnost** (*confidentiality*): absence neautorizovaného prozrazení (odhalení) informací
5. **integrita** (*integrity*): absence nevhodných změn stavu systému
6. **udržovatelnost** (*maintainability*): schopnost podstoupit opravy a modifikace

Je nutno poznamenat, že pouze „*dostupnost*“ a „*spolehlivost*“ mohou být kvantitativně vyhodnoceny. Ostatní atributy jsou pouze kvalitativní a do jisté míry závisí na subjektivním posuzování.

Kromě primárních atributů dependability existuje i celá řada atributů sekundárních, které jsou buď odvozeny z atributů primárních (a jsou tudíž vyjádřitelné pomocí primárních atributů) nebo jsou zaměřeny na určité hrozby pro dependabilitu (tj. hodnotí dependabilitu systémů vzhledem k těmto specifickým hrozbám).

Mezi sekundární atributy prvého druhu patří například:

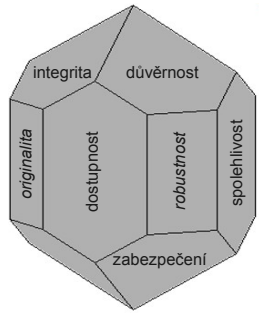
- *zodpovědnost* (dostupnost a integrita totožnosti osoby, která provádí určitou operaci v rámci služby)
- *originalita* (integrita obsahu zprávy včetně metadat, například času odeslání zprávy)
- *nepopíratelnost* (dostupnost a integrita totožnosti odesílatele nebo příjemce zprávy)

Typickým sekundárním atributem druhého druhu je *robustnost*. V širokém smyslu robustnost hodnotí dependabilitu vzhledem k externím závadám, tj. charakterizuje reakci systému na specifickou třídu závad. V užším pojetí je robustnost definována jako odolnost systému proti chybným vstupním datům.

Zvláštní postavení v kontextu dependability zaujímá termín „*bezpečnost*“ (*security*). *Bezpečnost* není jednoduchým atributem dependability, ale jedná se o kombinovaný pojem, který lze definovat například takto [30]:

Bezpečnost = *absence neoprávněného přístupu ke stavu systému, resp. neoprávněné ošetření jeho stavu.*

Jednotlivé atributy dependability si lze představit jako *fasety* omezující hodnocení dependability konkrétního systému (vychází se z představy rovin = faset, které vymezují výbrus drahého kamene, viz obrázek 4.4) nebo duálně jako fasety, které umožňují vyjádřit požadavek na dependabilitu libovolného systému.



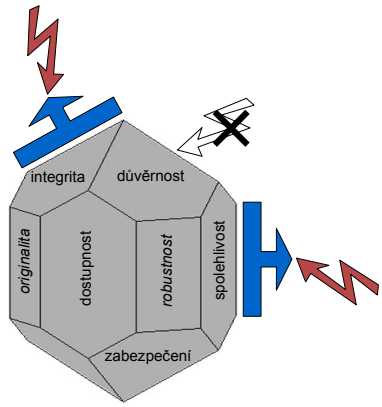
Obrázek 4.4.: Fasety dependabilního systému

V jednotlivých reálných situacích je kladen důraz na různé atributy, tj. fasety dependability. Preference jednotlivých faset přímo ovlivňuje výběr technik, které by měly být použity pro zajištění dependability, resp. k odvrácení hrozeb v dané oblasti. Atributy, u nichž se nepředpokládají hrozby, nebo na něž není kladen dostatečný důraz, mohou být opomenuty (tj. nejsou uvažovány v návrhu systému). Tento případ je znázorněn na obrázku 4.5 na následující straně. Hrozby jsou znázorněny symbolem blesku, prostředky odvrácení modrými šipkami. Zohledněny jsou pouze hrozby v oblasti spolehlivosti a integrity, tj specifikace dependability musí zahrnout požadavky specifikované v termínech akceptovatelné úrovně četnosti a závažnosti selhání pro určité třídy závad (a v kontextu prostředí, ve kterém bude systém použit).

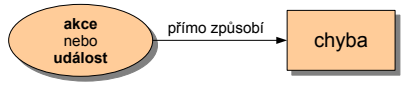
Závada, chyba, selhání

Pro lepší pochopení hrozeb se zaměříme na klíčový řetězec: závada → chyba → selhání a přesněji definujeme každou jeho část.

Závada (*fault*) je zjištěná nebo hypotetická příčina chyby. Je to odpověď na otázku, co přímo zapříčinilo konkrétní chybu. Přímou příčinou (tj. odpovědí na tuto otázku) může být akce nebo událost např. porucha hardwaru, nesprávně napsaný software nebo cizí vniknutí.



Obrázek 4.5.: Fasety dependabilního systému a hrozby



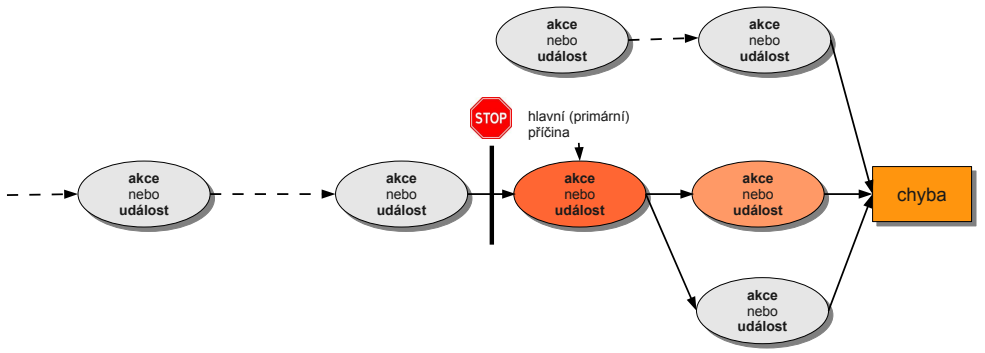
Obrázek 4.6.: Závada

V pokládání otázek však můžeme přirozeně pokračovat: co způsobilo, vyvolalo nebo zapříčinilo akci, resp. událost, která vedla k chybě. Takto se lze dotazovat rekurzivně znovu a znovu (co bylo příčinou akce, která způsobila akci, ... která vedla k chybě), potenciálně donekonečna (viz obrázek 4.7). V praxi musíme potenciálně nekonečné kladení otázek o příčině v některém bodě ukončit a konstatovat, že poslední nalezená příčina je hlavní či primární příčina, tj. akce, resp. událost, spouštějící řetězec dalších událostí vedoucích k chybě.

Rozhodování o tom, v jakém místě zastavit, může být učiněno buď na základě důsledného prohlížení řetězce závad směrem k primární příčině (viz obrázek), nebo můžeme přeskočit určitou část řetězce nebo jeho určitou větev (řetězec se může v obecném případě i větvit), tj. neprovádět hledání všech potenciálních akcí nebo událostí. V prvním případě se jedná o zjištěné *příčiny chyb*, v druhém o *příčiny hypotetické*.

Každou závadu můžeme charakterizovat pomocí 8 různých kriterií [26]:

1. *fáze vzniku nebo výskytu* (phase of creation or occurrence)
2. umístění vzhledem k *ohraničení systému* (system boundary)
3. *fenomenologická příčina* (phenomenological cause)



Obrázek 4.7.: Řetězec závad (příčin)

4. *dimenze* (dimension)
5. *cíl* (objective)
6. *úmysl* (intent)
7. *kapacita* (capacity)
8. *setrvání, perzistence* (persistence).

Kriteria v podstatě odpovídají následujícím upřesňujícím otázkám:

1. kdy vznikla závada (v době návrhu systému nebo v průběhu jej údržby a použití)
2. kde vznikla závada (uvnitř systému nebo vně systému)
3. jaký původ má závada (přirozený nebo lidsky zapříčiněný)
4. v jakém komponentu systému vznikla závada (HW nebo SW)
5. jaký záměr má závada (zlomyslná nebo nezlomyslná)
6. jak byla závada naplánována (promyšlená, resp. úkladná či nikoliv)
7. jak vznikla závada (nahodile nebo v důsledku lidské nekompetence)
8. jaké je setrvání závady (trvalé nebo přechodné)

Na každou otázku existují dvě možné odpovědi. Závada, která je charakterizována jen podle jednoho kriteria, se nazývá *elementární*. Vzhledem k počtu kriterií resp. otázek existuje 16 elementárních závad. Elementární kriteria lze přirozeně kombinovat a vytvářet tak kombinované charakteristiky závad (jako výsledek odpovědi na více než jednu otázku). Například konkrétní závada může být charakterizována jako externí zlomyslná nebo jako nahodilá interní hardwarová. Teoreticky je možno

vytvořit $3^8 - 17 = 6544$ různých kombinovaných závad, ale jen 31 kombinací je v praxi použitelných (tj. jsou možné a navíc mají vyšší pravděpodobnost vzniku). Těchto 31 kombinovaných závad lze rozčlenit do 3 částečně se překrývajících hlavních skupin:

1. závady návrhu (zahrnují všechny závady, které vznikly v době návrhu systému)
2. fyzické závady (zahrnují všechny závady, které se týkají hardwaru)
3. závady interakce (zahrnují všechny externí závady).

Závada se nachází v jednom ze dvou stavů:

- latentní (spící) závada
- aktivní závada.

Přechod z latentního do aktivního stavu se označuje jako aktivace závady. Závada, která se aktivovala (= je v aktivním stavu), *může* způsobit chybu.

Chyba *(angl. error) je část celkového stavu systému, která může vést k následujícímu selhání služby* [27].

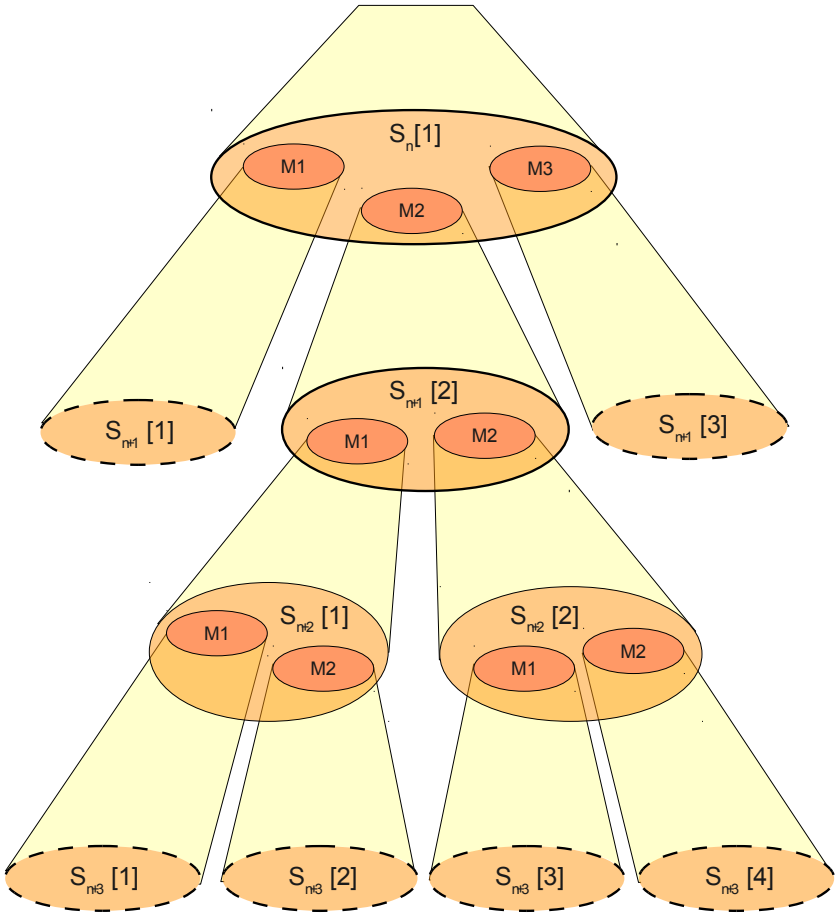
Chybu lze proto též stručně charakterizovat jako neplatný (invalidní) stav systému.

Pro chyby je typický proces tzv. **šíření chyb**, kdy jedna chyba může vyvolat chyby další (a ty mohou vyvolat zase další chyby, atd). Pro lepší pochopení šíření chyb je nutné mít jasnou představu o tom, co je systém, hranice systému a systémové prostředí.

Jedním ze základních rysů systému je strukturovanost. Systém se skládá z komponent, které jsou určitým způsobem spojené a vzájemně se ovlivňují. Komponenta je v podstatě další systém, jenž je opět strukturovaný, a tudíž se skládá opět z dalších komponent. Tato rekurze pokračuje, dokud není dosaženo komponenty, která je považována za atomickou. Tato rekurze v podstatě odráží hierarchickou strukturu systému (viz obr. 4.8 na následující straně).

Hranice systému vymezuje samotný systém a odděluje ho od ostatních systémů. Systémy vně hranice pak tvoří systémové prostředí daného systému.

Na dané úrovni hierarchie si lze představit, že chyba vznikne v jedné z komponent a další komponenty systému jsou prozatím bezchybné. Pokud se chyba šíří jen v rámci dané komponenty, jedná se o interní šíření chyby. Chyba se však může rozšířit i na další komponenty (externí šíření) a nakonec může ovlivnit službu poskytovanou systémem, tj. ovlivnit tu část systému, která navenek službu poskytuje (tzv. *rozhraní služby*).



Obrázek 4.8.: Hierarchická struktura systému

Selhání systému vzniká v okamžiku, kdy šíření chyby dosáhne rozhraní služby a tuto službu neakceptovatelně (a tudíž i detekovatelně) změní.

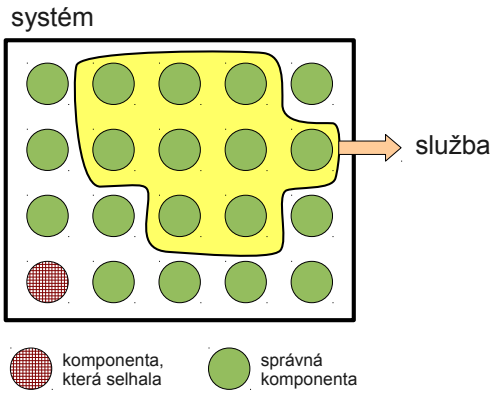
Protože však systém může být zároveň i komponentou v hierarchicky nadřazeném systému, je *selhání* této komponenty-podsystému zároveň i *chybou* v nadřízeném systému. Chyba vzniká v daném podsystému-komponentě a může se dál šířit na úrovni nadsystému (zde může dosáhnout jeho rozhraní, způsobit tak jeho selhání, atd.). Oba termíny splývají u atomických komponent (chyba v komponentě je zároveň i selháním komponenty-podsystému).

Pro zajištění dependability je klíčovým východiskem fakt, že selhání komponenty (tj. chyba v systému) ***nemusí*** vést k selhání celého systému. Obrázek 4.9 na následující straně ukazuje situaci, kdy se vadná komponenta neúčastní procesu posky-

tování služby, a tudíž neovlivňuje její rozhraní. Komponenta sice může být později do tohoto procesu zapojena, ale do té doby může být obnovena její správná funkčnost.

Selhání komponenty (tj. chyba v systému) může proto mít následující následky:

- může okamžitě ovlivnit službu poskytovanou systémem (tj. vést k bezprostřednímu selhání systému)
- může ovlivnit službu poskytovanou systémem až po časové prodlevě, tj. až v okamžiku zapojení komponenty do procesu poskytování služby. Délka prodlevy je potenciálně neomezená.
- nemusí vůbec ovlivnit službu poskytovanou systémem, systém tak zůstává dependabilní.



Obrázek 4.9.: Chyba a selhání systému

To, že konkrétní selhání komponenty nevede k selhání systému, nemusí být jen nahodilá shoda okolností. Naopak, do systému lze přidat prostředky, které mohou řízeně eliminovat vliv vadného komponentu na službu poskytovanou systémem. Tato koncepce se označuje jako *odolnost systému proti závadám* (angl. *fault-tolerance*).

Vzhledem k tomu, že selhání komponenty je zároveň i chybou systému, je třídění chyb a selhání velmi podobné. Protože je však selhání systému-komponenty chápáno jako událost, kdy se poskytovaná služba liší od služby správné, je nutno stanovit kriteria, na jejichž základě bude možno tento fakt zaregistrovat. Poskytovanou službu je možné hodnotit v různých dimenzích, a to jak z hlediska jejího obsahu, tak načasování dodání služby. V kontextu dependability je takovéto hodnocení služby definováno jako *doména selhání*. Selhání mohou být dále tříděna podle

detekovatelnosti, konzistentnosti a následků. Vyčerpávající rozbor selhání a jejich třídění je uveden v [26].

4.3. Bezpečnostní selhání

Zvláštní místo při zajištění dependability výpočetního systému a rozsáhlých webových aplikací patří *bezpečnostním selháním*. Bezpečnostní selhání vzniká, pokud systém nedodržuje stanovené standardy a politiky v oblasti bezpečnosti. Bezpečnostní politika definuje požadavky na systém pomocí cílů a pravidel. Cíle se snaží zahrnout bezpečnostní požadavky vysoké úrovně. Typické bezpečnostní cíle mohou zahrnovat:

- musí být udržována důvěrnost citlivých dat
- musí být udržována integrita a dostupnost systémových dat pro oprávněné uživatele.

Porušení (nedodržení) cílů vede okamžitě k bezpečnostnímu selhání. Pravidla omezují chování systému na nižší úrovni abstrakce (tj. omezení stavů systému a přechodů z jednoho stavu do druhého). Pravidla jsou navržena tak, aby zajistila robustnost systému (tj. jeho odolnost proti svévolným činnostem), a vycházejí z bezpečnostních cílů. Bohužel ne všechny cíle lze v praxi mapovat na pravidla. Nedodržení pravidel vede k chybě systému. Naopak jejich dodržování by mělo v ideálním případě vést k eliminaci bezpečnostních selhání.

Některá bezpečnostní pravidla jsou aplikována na uživatele a primárně omezují jeho činnosti. Mají tudíž podobu zákazů (*prohibition*), povinností a závazků. Další pravidla jsou aplikována na technický systém, a vystupují tudíž jako pravidla řízení přístupu do systému nebo řízení přístupových práv uživatelů.

Závady (tj. příčiny vedoucí k bezpečnostním chybám) lze rozdělit do pěti hlavních skupin [31]:

1. závady ve specifikaci bezpečnostních cílů
2. závady ve specifikaci bezpečnostní politiky (pravidel)
3. závady v implementaci technických pravidel
4. závady v nízkoúrovňových technických mechanismech
5. závady v sociálně-technickém mechanismu.

Závady ve specifikaci cílů vznikají kvůli *nesprávnému vyjádření bezpečnostních požadavků*.

Závady ve specifikaci pravidel vznikají, když pravidla ne zcela odpovídají cílům. To je možné z několika důvodů:

- pravidla mohou být nekompletní
- pravidla mohou být nekonzistentní nebo nejednoznačná
- nesprávná analýza logických následků dané množiny pravidel .

Závady v implementaci technických pravidel vznikají z následujících důvodů:

- jednoduchá chyba v kódování
- pravidla neodpovídají abstraktně specifikovaným pravidlům
- integrita pravidel není garantována
- chybná údržba pravidel
- nesprávné mapování pravidel na architekturu systému, atd.

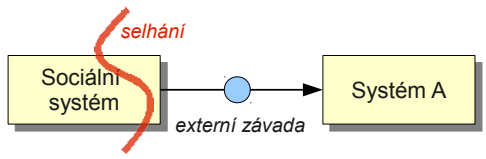
Závady v nízkoúrovňových technických mechanismech mohou například vznikat jako následek nesprávného kódování (šifrování) v mechanismech autentifikace a autorizace.

Závady v sociálně-technickém mechanismu mohou vznikat, protože:

- povinnosti a závazky uživatelů jsou nepřesně nebo nedostatečně formulovány
- uživatelé nedodržují své povinnosti a závazky

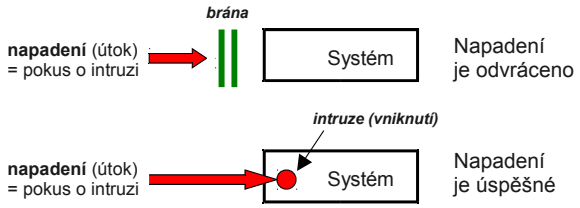
Sociálně-technický mechanismus má omezit chování uživatelů a odvrátit vznik takových závad. Obvykle se provádí důkladný výběr pracovníků a jejich následné cvičení v otázkách bezpečnosti. Dále se provádějí pravidelné kontroly a audity.

Zvláštní role hrají externí závady systému, které vznikají v důsledku selhání sociálního systému. Obrázek 4.10 znázorňuje podstatu externí závady. Selhání sociálního systému je zároveň i externí závadou systému *A*.



Obrázek 4.10.: Externí závada jako selhání sociálního systému

Často se selhání sociálního systému projevuje jako napadení systému, s nímž je sociální systém v kontaktu. Napadení lze definovat jako pokus o intruzi (*intrusion*,

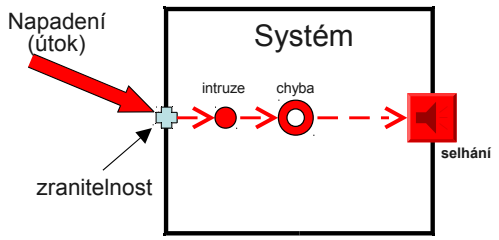


Obrázek 4.11.: Napadení a intruze

nežádoucí vniknutí do systému). Intruzi pak lze definovat jako záměrnou operační závadu, která přichází zvenku systému (jako výsledek úspěšného napadení).

Pro vznik intruze jsou proto nutné dva předpoklady (viz obr. 4.12):

1. zlomyslný čin nebo napadení, které se pokusí využít slabiny systému
2. nejméně jedna slabina, nedostatek, vada neboli zranitelnost.



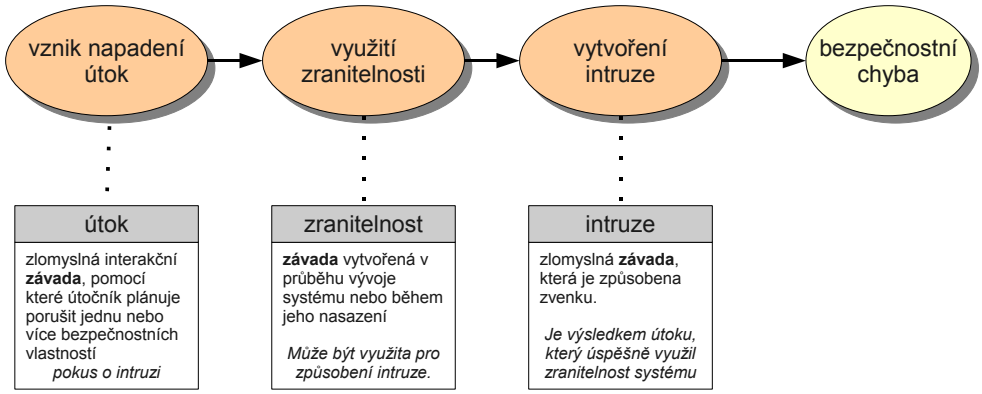
Obrázek 4.12.: Intruze a zranitelnost

Zranitelnost (tj. slabina systému) může vzniknout v průběhu vývoje systému nebo během jeho nasazení (provozu). Zranitelnosti mohou vznikat neplánovaně (tj. jsou způsobeny například návrhářem, vývojářem, operátorem nebo provozovatelem) nebo záměrně (způsobené např. hackerem). Lze je také klasifikovat jako zlomyslné (hacker) nebo vytvořené bez zlého úmyslu (návrhář, operátor).

Podobně jako v případě ostatních závad existují metody, které zajišťují odolnost proti intruzím, přesněji umožňují tolerovat události, kdy zranitelnost je úspěšně zneužita útočníkem.

Následující příklady ukazují typické intruze z pohledu napadení a zranitelnosti:

- *cizí osoba proniká do systému pomocí odhalení hesla uživatele*



Obrázek 4.13.: Řetězec závad při bezpečnostní chybě

Zranitelnost spočívá ve špatné konfiguraci systému a/nebo ve špatně zvoleném hesle (příliš krátké nebo snadno podléhající slovníkovému útoku).

- *interní pracovník zneužívá svá privilegia v rámci systému*
  Zranitelnost zde spočívá v nesprávné specifikaci nebo v nesprávném návrhu sociálně-technického mechanismu. Např. nedodržení principu „nejmenších privilegií“, nedostatečné prověření zaměstnanců atd.
- *cizí osoba využívá „sociální inženýrství“ (např. podplácení) aby přinutil interního pracovníka zneužít svoje privilegium ve svůj prospěch*
  Zranitelností je zde přítomnost podplaceného interního pracovníka, což je výsledkem špatného návrhu sociálně-technického mechanismu (např. nedostatečné prověření zaměstnanců).

4.4. Odvracení hrozeb

Pro odvrácení hrozeb a zajištění dependability systému je nutné kombinované použití následujících čtyř metod:

- prevence závad
- odolnost proti závadám
- odstranění závad
- předpověď závad

Prevence závad

Prevence závad je dosažitelná prostřednictvím technik řízení kontroly kvality v průběhu návrhu a výroby hardwarových i softwarových komponent, ale také pomocí cvičení personálu a striktním dodržováním stanovených pravidel a termínů údržby.

Příklady prevence závad v oblasti softwaru jsou:

- *objektově orientované programování* (s využitím prověřených hotových knihoven)
- *komponentové programování*. Komponenty je možné bezpečně využít v různých aplikacích. Na počátku to byly především prvky grafického návrhu (tlačítko, textové políčko apod.), dnes jsou z komponent skládány celé systémy.

V kontextu hardwaru se může jednat o instalaci krytů omezujících vliv okolního prostředí (např. radiace) nebo striktní dodržování norem při výrobě. Příkladem prevence interaktivních (lidmi způsobených) závad je kromě tréninku personálu i zatajování informací o systému (v lokálním měřítku, v globálním není příliš účinné viz např. utajování kryptografických metod) a dále moduly omezující nesprávné použití systému (*foolproof*, neoficiálně označované jako blbuvzdorné).

Odstranění závad

Je zřejmé, že odstranění závady může do značné míry zvýšit spolehlivost systému. Nicméně, je nutné poznamenat, že především v případě softwaru je velmi těžké kvantitativně vyhodnotit zvýšení spolehlivosti SW po odstranění konkrétní závady.

Odstranění závad probíhá jak během vývoje (návrhu) tak i za běhu systému. Odstranění závad během vývoje systému se skládá ze tří kroků:

- verifikace a validace
- diagnostika
- korekce.

Celý proces odstranění závad může být proveden jak v průběhu vývoje systému (obsahující fázi specifikace, návrhu a implementace) tak i v průběhu jeho praktického využití, tj. během nasazení. Ve fázi návrhu se provádí verifikace i validace systému.

Verifikace a validace

Verifikace odpovídá na otázku, zda je systém vytvářen správně. Je to interní proces kontroly, který má ověřit, zda systém v dané fázi vývoje odpovídá konceptuálnímu

modelu. Proces verifikace není jednolitý proces, ale jedná se o souhrn procesů, které se provádějí vícenásobně a jsou zaměřeny na jednotlivé komponenty systému. Teprve po ukončení testování jednotlivých komponent a po integraci komponent do výsledného systému je prováděno **integrační testování**, jako jedna z fází verifikace.

Verifikace se realizuje prostřednictvím dílčích testů. Testovací techniky lze rozdělit na statické a dynamické. Dynamické testování vyžaduje provoz komponent, tj. např. zprovoznění hardwaru nebo provedení testovaného kódu. Dynamické testování lze dále rozdělit na:

- funkcionální testování (*black box* testování)
- strukturální testování (*white box* testování)
- nahodilé testování (*random* testování).

Statické testování na rozdíl od dynamického nevyžaduje provoz nebo vykonání testovaných komponent. Mezi statické testování patří:

- techniky založené na analýze konzistence kódu (resp. korektnosti programu)
- techniky kvalitativně hodnotící některé obecné vlastnosti komponenty (např. náchylnost k jistým druhům chyb).

Validace na rozdíl od verifikace odpovídá na otázku, zda hotový systém odpovídá všem požadavkům externího zákazníka nebo uživatele systému. Při validaci navíc vždy operujeme se systémem jako celkem a nikoliv s jednotlivými komponentami, i když i zde uvažujeme systém jako sadu propojených a spolupracujících komponent. Validační testování proto začíná tam, kde končí testování integrační (tj. verifikace).

K validačnímu testování patří akceptační testování, resp. alfa a beta testy.

Akceptační testování se provádí pro systémy, které byly vyvinuty pro konkrétního zákazníka. Tento zákazník provádí akceptační testování na reálných datech, přičemž hodnotí zda systém vyhovuje jeho požadavkům. Pro systémy, které jsou vytvořeny pro širší veřejnost, se provádí alfa a beta testy. Alfa testy se provádí v prostředí, v němž byl systém vytvořen. Beta testy pak provádí uživatelé ve svém prostředí.

K obecným technikám, které se používají při validaci systému, patří:

- formální metody
- podsouvání závad (vstřikování závad)
- analýza závislostí

- analýza hazardu
- analýza risku.

Je nutné poznamenat, že procesy verifikace a validace jsou časově velmi náročné a vyžadují velké množství předem připravených testů. Kromě toho mohou analýzy výsledků provádět pouze předem připravení specialisté spolu s vývojáři.

Návrh hardwaru a softwaru je tradičně oddělen a podobně je oddělena i metodika verifikace hardwaru a softwaru. Testování hardwaru se navíc děje odděleně od testování softwaru v rámci dvou různých procesů.

Novým trendem v oblasti vestavěných systémů je tzv. *co-design*, který spojuje návrh hardwaru i softwaru do jediného formalizovaného procesu. Spojení je dáno relativní blízkostí hardwaru a softwaru ve vestavěných systémech. Software je navrhován pro konkrétní hardware a dokonce i hardware je přizpůsobován používanému softwaru.

Co-design nabízí formalizovaný návrh s formalizovaným testováním jak hardwaru tak softwaru. Rozdělení vývoje na softwarovou a hardwarovou část se neděje na začátku vývojového procesu, ale průběžně během celého vývoje tak, aby se dosáhlo optimálního výsledku (efektivity, složitosti, ceny). To se přirozeně projevuje i v oblasti verifikace, kde se testuje systém jako celek (tj. hardware i software).

Cena verifikace a validace reálně využívaných výpočetních systémů dosahuje téměř třetiny ceny jejich vývoje a může dosáhnout až 50 % u kritických systémů (u systémů, kde selhání může mít velmi závažné následky) [41]. V případě softwarových kódů umožňuje verifikace redukovat počet závad s četností 100 – 300 závad na 1000 řádků kódu (výsledek vývoje SW) na 0.01 až 10 závad na 1000 řádků kódu. Stále však zůstává v průměru jedna závada na 1000 řádků kódu (tj. komplexní systémy mohou obsahovat stovky až tisíce závad).

Jestliže verifikace resp. validace ukáže, že systém nesplňuje stanovené vlastnosti, pak musí být provedeny další kroky, tj. diagnostika závad, jež způsobily odchylku systému od stanovených vlastností, a následná korekce těchto závad.

Diagnostika závad

V této sekci se zaměříme především na softwarové závady. Diagnostika softwarových závad, která se provádí v rámci vývoje výpočetního systému, se liší od diagnostiky prováděné v době jeho nasazení. Důvodem je skutečnost, že současná vývojová prostředí podporují použití složitějších automatických diagnostických technik, které výrazně usnadňují diagnostický proces. Automatická diagnostika nevyžaduje na rozdíl od ručního ladění vynaložení značného mentálního úsilí pro lokalizaci závad a je prováděna ve výrazně kratším čase.

Metody používané pro diagnostiku, tj. lokalizaci softwarových závad, lze třídit podle informací o softwarovém systému, které jsou k dispozici, např. informace o interní struktuře nebo o chování systému.

Hlavními třídami jsou:

- metody černé skříňky (*black box*)
- metody bílé skříňky (*white box*).

Metody černé skříňky nevyžadují znalosti interní struktury programu, interní logiky struktury dat či interního stavu, resp. chování programu. Dostačuje pouze dostupnost zdrojového kódu v podobě posloupnosti řádků.

Mezi nejčastější metody černé skříňky patří:

1. lokalizace závad na základě spektra
2. metoda nejbližšího souseda (*nearest neighbor*)
3. dynamické odkrajování (plátkování, segmentace) programu (*dynamic program slicing*)
4. delta-ladění (*delta debugging*).

Podstatu metod černé skříňky je možné ozřejmit na příkladu *lokalizace závad založené na výpočtu tzv. spektra.*

Předpokládejme následující schéma procedurálního kódu:

```
1 příkaz3;
2 if (podmínka) {
3     příkaz1;
4     if (podmínka)
5         příkaz2;
6 }
```

Příkazem zde mohou být nejen elementární operace jako přiřazení, ale i bloky příkazů či volání funkcí, resp. procedur. Volbou komplexnosti zohledněných příkazů lze řídit granularitu diagnostiky (od úrovně komponent až na úroveň jednotlivých řádků kódu).

Diagnostika vychází z opakovaného spouštění programu nad různými vstupními daty, přičemž se posuzuje, zda je výsledek programu správný či nikoliv. Navíc je nutno pro každý zohledněný příkaz (resp. blok nebo volání funkce/metody) poznamenat, zda byl při daném spuštění vykonán. Jinými slovy, po každém provedení je každý zohledněný příkaz označen příznakem, který nese informaci o pořadí spuštění, v jehož rámci byl vykonán, a tím následně i o výsledku spuštění.

Rozhodující jsou příznaky vytvořené v rámci těch spuštění, která vedla k nesprávnému výsledku. Intuitivně můžeme očekávat, že příkazy které mají těchto příznaků

nejvíce, budou s vyšší pravděpodobností vadnými (tj. obsahují závadu, která způsobila chybný výsledek).

V reálných diagnostických nástrojích (např. v aplikaci *Spectrum* [32]) se rozhodování o vadném příkaze děje na základě koeficientu podobnosti (angl. *similarity coefficient*), který v kompaktní podobě (jedná se o jediné číslo) odráží pravděpodobnost, že daný příkaz obsahuje chybu.

Podstatu koeficientu podobnosti si vysvětlíme na zjednodušeném příkladě programu s pouhými třemi příkazy/bloky (viz předchozí schéma) a třemi testovacími spuštěními programu.

Následující tabulka ukazuje výsledek programu pro jednotlivé běhy včetně příznaků provedení jednotlivých příkazů. Příznaky lze v takto jednoduchém příkladě určit prostým pohledem do zdrojového kódu, ale u reálných aplikací je nutné využít specializované diagnostické programy, resp. knihovny, které aktivitu příkazů automaticky monitorují. V našem ukázkovém výstupu je chybný výsledek detekován jen u druhého spuštění (hodnota „1“ v posledním sloupci). *Příkaz3* je vykonán při všech spuštěních (sloupec obsahuje samé hodnoty „1“). *Příkaz2* je vykonán jen při druhém spuštění a *příkaz1* navíc ještě při spuštění prvém.

	příkaz1	příkaz2	příkaz3		výsledek
1. spuštění	1	0	1		0
2. spuštění	1	1	1		1
3.spuštění	0	0	1		0

Koeficient podobnosti odráží podobnost dvou sloupců: sloupce příznaku provedení konkrétního příkazu a sloupce výsledků. Výpočet koeficientů vychází z hodnot s_{ij}, které získáme jako počet řádků, u nichž první sloupec nabývá hodnoty i a druhý hodnoty j ($i, j \in \{0,1\}$). Například pro příkaz3 se vychází z porovnání následujících dvou sloupců:

příkaz3		výsledek
1	$\Longleftrightarrow$	0
1	$\Longleftrightarrow$	1
1	$\Longleftrightarrow$	0

V tabulce se dvakrát vyskytuje kombinace $(1,0)$, a tudíž $s_{10} = 2$, jednou kombinace $(1,1)$, tj. $s_{11} = 1$. Ostatní kombinace se nevyskytují tj. $s_{00} = s_{01} = 0$.

V průběhu posledních desetiletí bylo navrženo několik koeficientů podobnosti vycházejících z hodnot s_{ij}. Nejčastěji se používá koeficient Jaccarda [33] (K_j), koeficient Tarantula [34] (K_T), koeficient Ochiai [35] (K_O) a koeficient AMPLE [36] (K_a). Tyto koeficienty lze vyčíslit následovně:

$$
\begin{aligned}
K_j &= \frac{s_{11}}{s_{11}+s_{01}+s_{10}} \\
K_T &= \frac{\frac{s_{11}}{s_{11}+s_{01}}}{\frac{s_{11}}{s_{11}+s_{01}}+\frac{s_{10}}{s_{10}+s_{00}}} \\
K_O &= \frac{s_{11}}{\sqrt{(s_{11}+s_{01})(s_{11}+s_{10})}} \\
K_A &= \left| \frac{s_{11}}{s_{01}+s_{11}} - \frac{s_{10}}{s_{00}+s_{10}} \right|
\end{aligned}
$$

Volba koeficientu závisí na mnoha faktorech, mimo jiné i na charakteru softwarového systému.

V našem ukázkovém příkladě zvolíme nejjednodušší Jaccardův koeficient. Pro *příkaz3* je tento koeficient roven $K_j = \frac{1}{1+2} = \frac{1}{3}$. Podobně lze Jaccardův koeficient vypočíst i pro *příkaz1* (= 0.5) a *příkaz2* (= 1.0). Protože platí, že čím je Jaccardův koeficient vyšší, tím je vyšší i pravděpodobnost závady v daném příkaze, resp. bloku, lze jako nejpodezřelejší označit provedení *příkazu2* (tj. tento příkaz s nejvyšší pravděpodobností příčinou chybného výstupu). To lze v tomto jednoduchém příkladě konstatovat i pouhým pohledem, ale u rozsáhlejších systémů to výrazně usnadňuje diagnostiku.

Použití pokročilých diagnostických metod černé skříňky vyžaduje podporu na úrovni aplikací, softwaru a často i hardwaru. Na aplikační úrovni je to pomocný diagnostický program, který shromažďuje informace z hardwaru a softwaru a umožňuje jejich konfiguraci (včetně automatizace testovacích běhů a volby vstupů) a vizualizaci (včetně například výpočtu koeficientů podobnosti). Systém také může poskytovat statistiky popisující důvěryhodnost diagnostiky (angl. *accuracy of diagnosis*).

Na softwarové úrovni je nutno zajistit aktivitu jednotlivých bloků, resp. obecněji pokrytí kódu (tj. část kódu, která je při daném provedení spuštěna). Pro to lze použít buď klasické profilovací nástroje nebo podporu v kompilátoru (v některých programovacích jazycích je programovatelný i kompilátor). Nejúčinnější je využití profilingu na úrovni hardwaru, např. podpora tzv. *Performance Monitoring Unit* (PMU) u moderních procesorů (detaily viz [37]).

Koeficienty podobnosti ostatní metody černé skříňky lze využít i pro diagnostiku softwarových závad v distribuovaných aplikacích. Zde je základní entitou diagnostiky distribuovaná komponenta (subsystém). Sběr a zpracování diagnostických informací musí být implementováno také distribuovaně.

Kromě metody lokalizace závad na základě spektra se používají i další metody černé skříňky.

Metoda nejbližšího souseda [38] vychází z libovolného spuštění, které vyprodukovalo chybný výsledek. V dalším kroku je určeno (vypočítáno) spuštění, které má nejpodobnější pokrytí kódu a zároveň nevedlo k chybnému výsledku. Toto spuštění tedy vykoná téměř všechny příkazy chybného spuštění (přesněji: počet příkazů, které byly provedeny jak v chybném tak úspěšném spuštění, je maximální). Je zřejmé, že za nejpravděpodobnější příčinu chyby lze označit ty příkazy, které byly provedeny jen v neúspěšném provedení (jejichž počet je navíc minimalizován).

Metoda dynamického odkrajování [39] postupně zužuje oblast podezřelých příkazů do stavu, kdy zůstávají jen příkazy, které přímo ovlivňují výstup např. tím, že nastavují výstupní proměnné.

Delta-ladění [40] vychází z porovnání stavů při úspěšném a neúspěšném spuštění. Z tohoto srovnání se odvozuje příčina, která způsobila rozdíl ve stavech softwarového systému.

Obecně platí, že metody černé skříňky na jedné straně nepotřebují detailní rozbor kódu, což může být enormně náročně, na straně druhé však často neposkytují přesnou lokalizaci závady. Přesnost závisí nejen na zvolené metodě, ale i na počtu spuštění a také poměru úspěšných a neúspěšných spuštění.

Z tohoto důvodu se využívají i tzv. **metody bílé skříňky**, které vyžadují velké množství informací o interní struktuře programu a jeho chování. Zohledněny musí být například tyto informace:

- větvení a další komplexnější cesty toku programu
- interní struktura a logika dat
- interní stavy systému.

Rozlišují se následující druhy metod bílé skříňky :

- tradiční metody, které používají model softwarového systému buď ve formě diagramu toku řízení, resp. *Bipartite Network Flow*, nebo grafů syntaxe nebo grafů stavových přechodů,
- objektově orientované metody,
- komponentně orientované metody.

Nízkoúrovňové prostředky pro odhalení (signalizaci) chyb se liší pro různé druhy softwarových systémů:

U vestavěných softwarových systémů se signalizace chyb děje především na základě testování předběžných a následných podmínek (angl. *precondition*, *postcondition*) za použití testovacích tvrzení tzv. asercí.

U běžných desktopových aplikací a centralizovaných služeb se chyby nejčastěji signalizují prostřednictvím mechanismu **výjimek**, které navíc umožňují zachycení i následnou běhovou korekci.

V případě webových a distribuovaných aplikací se používají chybové zprávy podporované použitými přenosovými protokoly (např. HTTP nebo SOAP).

Korekce závad

Odstranění (korekce) závad v průběhu běhu systému (tj. ve fázi nasazení) má charakter korekční nebo preventivní údržby. Korekční údržba má za cíl odstranit závady, které způsobily jednu nebo více chyb a byly odhaleny. Preventivní údržba má oproti tomu za cíl odhalit a odstranit závady ještě před tím, než způsobí chyby v průběhu normálního fungování systému. Příkladem je odstranění závad návrhu, které byly odhaleny v jiných podobných systémech. Je nutno poznamenat, že pro odstranění závad se používá výhradně vnější diagnostika.

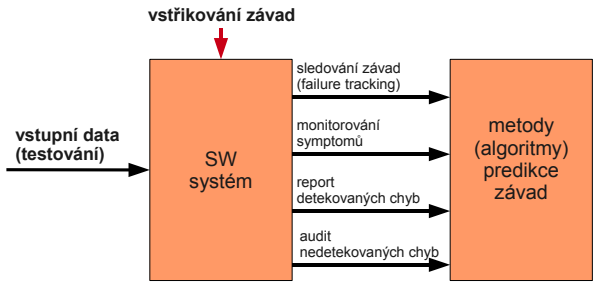
Jak již bylo řečeno výše nelze bohužel odstranit všechny závady ze systému (čas i zdroje jsou omezené). Vždy je proto zapotřebí provést i předpověď závad.

Předpověď závad

Předpověď závad zahrnuje techniky pro vyhodnocení přítomnosti závad, četnosti výskytů selhání systému včetně možných následků těchto selhání. Tyto techniky jsou používány ve fázi validace systémů a využívají podobných základních metod (analýza hazardu, analýza zisku a vstřikování závad). Hodnocení systému z pohledu předpovědi závad může být kvalitativní nebo kvantitativní.

Kvalitativní hodnocení je zaměřené na režimy selhání a jeho hlavním cílem je identifikace, seřazení a roztřídění závad podle jejich projevu a závažnosti, respektive stanovení situací, které by mohly k takovým selháním vést. Metody využívané pro kvalitativní hodnocení lze rozčlenit do čtyř skupin na základě používaných resp. zohledněných výstupů systému. Základní přehled poskytuje obrázek 4.14 na následující straně.

Obrázek ukazuje i důležitou roli metody vstřikování závad pro ohodnocení vlivu závad na chování systému (tj. poskytovanou službu) resp. pro validaci mechanismu



Obrázek 4.14.: Vstupní data pro metody předpovědi selhání systému

ošetření závad. Lze tak vyhodnotit i efektivitu mechanismu zajištění odolnosti systému proti závadám, například ohodnotit pokrytí závad (angl. *fault coverage*) nebo latenci chyb (angl. *error latency*).

Kvantitativní ohodnocení spočívá ve výpočtu spolehlivosti systému jako je střední doba do poruchy (dále MTTF), resp. pravděpodobnost selhání požadavku (dále *pdf*). Spolehlivost systému může být vyhodnocena jak pro stávající stav systému tak predikována pro jeho chování v budoucnu. Pro výpočet stávající spolehlivosti se používají data o selhání systému v průběhu jeho testování a především údaje o selháních během jeho nasazení.

V případě, kdy nejsou data o selháních, pak lze spolehlivost systému vypočítat buď na základě jeho modelování nebo odhadnout na základě počtu testování, které proběhlo bez selhání, resp. na základě doby testování, během níž nedošlo k selhání. Použití „hrubé síly“ a striktních statistických metod však vede k velmi vysokým požadavkům na dobu trvání testovací fáze. Uveďme několik příkladů.

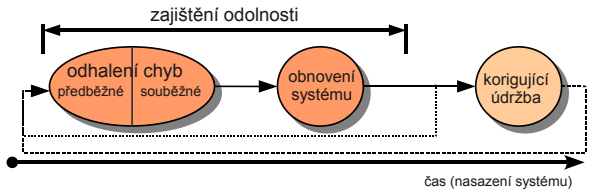
Pro systémy poskytující služby na požádání musí být pro dosažení 99% jistoty, že *pdf* je lepší (tj. menší) než 10^{-3}, provedeno 4600 testovacích požadavků, z nichž žádný nesmí vést k selhání. Pokud je vyžadováno *pdf* lepší než 10^{-4} (opět při 99% jistotě) vzroste toto číslo na 46 000. U systémů s nepřetržitou obsluhou (například u řídících systémů) je pro dosažení MTTF nad 10^4 hodin (~ 1,14 roku) nutno testovat 46 000 hodin (~ 5,25 roku), během nichž nesmí dojít k selhání. Zvýšení MTTF na hodnotu 10^5 hodin zdesateronásobí čas testování (na 460 000 hodin) [42].

Vzhledem k této enormní časové náročnosti se odborníci snaží využívat efektivnějších pravděpodobnostních metod pro předvídání spolehlivosti systému. Tyto metody jsou založeny na principu „*krátkodobé pozorování/testování ⟶ předpověď na dlouhé období*“. Hodnocení spolehlivosti systémů založené na těchto metodách je však stále kontroverzním tématem. Tak například v CLR (výzkumná laboratoř v Londýně) došli k závěru, že krátkodobé pozorování přispívá jen velmi málo k jistotě, že systém bude v budoucnu dlouhodobě fungovat.

Předpověď závad je důležitá i v kontextu poslední metody odvracení hrozeb – tj. v rámci zajištění dostatečné odolnosti systému proti závadám, neboť může ukázat, do jaké míry, resp. v jakém rozsahu, je nutno použít mechanismy, které tuto odolnost zajišťují.

4.5. Odolnost systému proti závadám

Mechanismy zajišťující odolnost proti závadám jsou zaměřeny na zabezpečení poskytování správné služby systémem za přítomnosti závad, které jsou v aktivním stavu. Základní časový průběh zajištění odolnosti systému proti závadám je ilustrován na obrázku 4.15.



Obrázek 4.15.: Odolnost systému proti závadám – časový průběh

Jak lze vidět, odolnost proti závadám je zajištěna dvěma dílčími procesy – nejdříve odhalením chyb a následným obnovením systému. O odhalení chyby v systému obvykle informuje buď signál nebo zpráva. Existují dvě třídy metod odhalení chyb: souběžné a předběžné. Souběžné odhalení chyb probíhá v průběhu poskytování služby systémem. Předběžné odhalení chyb probíhá v době, kdy je poskytování služby pozastaveno. V tomto případě je systém kontrolován na přítomnost latentních chyb a spících (neaktivních) závad.

Po úspěšném odhalení chyb musí následovat procedura obnovení systému. Obnovení transformuje systém ze stavu, který obsahuje jednu nebo více chyb a závad, do stavu bez odhalených chyb a závad, resp. bez závad, které by mohly být znovu aktivovány. Procesy odhalení chyb a obnovení systému se mohou v průběhu nasazení systému mnohonásobně opakovat.

Výše uvedené schéma presentuje zcela obecný pohled na proces zajištění odolnosti systému proti závadám. V konkrétním případě se mohou dílčí prvky procesu v čase překrývat, resp. nemusí být provedeny v celém svém rozsahu.

Konkrétní mechanismus *ošetření chyb,* tj. reakce na událost odhalení chyby, závisí v prvé řadě na principu implementace odolnosti systému proti závadám.V zásadě existují tři různé principy implementace tohoto procesu:

odrolování — transformace stavu probíhá jako vracení stavu systému do předem uloženého správného stavu (tj. např. stavu, který byl ještě před odhalením chyby).

přerolování — transformace stavu systému je přechodem do nového stavu, který neobsahuje chyby, které již byly odhaleny. Přerolování se nejčastěji spoléhá na nízkoúrovňový mechanismus výjimek a mechanismy z něho odvozené (např. ošetření vzniku souběžných výjimek).

maskování — transformace spočívá v odstranění chybného stavu systému tím, že chybné moduly jsou izolovány (tj. dále se neúčastní poskytování služby). To je možné pouze v systémech s redundancí prostředků (tj. s nadbytečnými prostředky, které by byly v případě bezchybných komponent zbytečné).

Součástí mechanismu ošetření chyb může být tzv. *ošetření závad,* tj. ošetření příčin chyb. Ošetření závad může obecně obsahovat následující kroky:

Krok 1: *Diagnostika závad.* Identifikace a zaznamenání příčiny chyb. V záznamech se uvádí lokalita a typ závady.

Krok 2: *Izolace závad.* Fyzické nebo logické vyloučení vadných komponentů z další účasti v poskytování služby (v případě logického vyloučení jde o vracení závad do spícího stavu).

Krok 3: *Rekonfigurace systému.* Buď přepnutí na náhradní komponenty, nebo použití správných komponent (které zůstaly v systému) a přerozdělení úkolů mezi správnými komponentami.

Krok 4: *Znovuinicializace.* Kontrola, aktualizace a zaznamenání nové konfigurace.

Po ošetření závad obvykle následuje korigující údržba, která odstraňuje závady, které byly izolovány v průběhu ošetření závad.

Využití mechanismu ošetření závad se však liší podle zvoleného principu implementace. V některých případech mohou být využívány jen některé kroky ošetření, resp. nemusí vůbec dojít k ošetření závad. Například v případě odrolování je běžně využita jen rekonfigurace systému (v podobě například tzv. obnovovacích bloků), ale v případě jednoduchého návratu k předem uloženému stavu, a tudíž i prostému opakování části programu, k ošetření závad vůbec nedochází (předpokládá se, že k opakovanému vzniku chyby nedojde).

Při využití mechanismu maskování lze využít diagnostiku závad pro lokalizaci závady, což umožňuje následné vyloučení závadné komponenty z poskytování služby (službu pak poskytují jen správné komponenty, viz obrázek 4.9 na straně 130). K lokalizaci chybné komponenty však nemusí vůbec dojít, neboť postačující je zjištění

správné komponenty, resp. podmnožiny správných komponent, které budou následně poskytovat službu. To znamená, že i v případě maskování nemusí být provedeno kompletní ošetření závad a závady mohou být důsledně ošetřeny a odstraněny až po delší době (např. u vestavěných systémů s dlouhodobým autonomním provozem). Dokončení diagnostiky se tak přesouvá až do fáze korigující údržby (kdy je nejčastěji zapotřebí účast externího agenta).

Při volbě konkrétní metody implementace odhalení a ošetření chyb, resp. ošetření závad je nutno vycházet z předpokladů o typu a charakteru selhání. Nejčastěji se používají následující (omezující) předpoklady:

systémy s ovladatelnými selháními — systémy, jejichž návrh a implementace zajišťují, že tyto systémy selžou specifickým způsobem popsaným v požadavcích dependability a pouze do přijatelné míry. Příkladem ovladatelných selhání jsou:

- nesprávná ale stálá výchozí hodnota (opakem je nahodilá výchozí hodnota)
- absence výchozí hodnoty resp. signálu (ticho na rozdíl od „blábolení“)
- konzistentní selhání (opakem je nekonzistentní selhání).

systémy typu „selhání ⟶ zastavení“ — selhání se vždy (resp. do přijatelné míry) jeví jako zastavení systému (např. projevující se jako „mlčení“ systému).

systémy s neškodnými selháními — systémy, jejichž selhání jsou do přijatelné míry méně závažná.

Situaci dále komplikují problémy spojené s rekurzívností pojmu „odolnost proti závadám“. Je totiž důležité, aby i mechanismy, které zajišťují odolnost systému proti závadám, byly samy chráněny proti svým závadám, které samozřejmě mohou ovlivnit jejich činnost. Tento rekurzivní problém známý již od starověku v podobě otázky: „*Kdo stráží stráže*?“[1] musí být uvažován při výběru, resp. návrhu mechanismů zajišťujících odolnost systémů, neboť odolnost samotných mechanismů může do značné míry ovlivnit dependabilitu celého systému.

Důležitým rysem, resp. východiskem, většiny schémat obnovy v oblasti softwarových systémů je použití tzv. diverzity (rozmanitosti) návrhu. Princip diverzity návrhu je založen na používání několika (nadbytečných) variant návrhu a implementace (u softwaru kódu), čehož se využívá pro odhalení chyb a obnovení aplikace.

[1] v originální podobě „Quis custodiet ipsos custodes?“ , Juvenalis

Jednotlivé varianty jsou vytvořeny nezávisle (např. různými týmy), ale musí vyhovovat společné specifikaci služby. Jinak řečeno varianty musí poskytovat stejnou službu, ale musí být implementovány různými způsoby (což eliminuje nebo alespoň snižuje pravděpodobnost existence stejné závady). Adjudikátor (rozhodčí algoritmus) následně určí jeden výsledek (jenž je považován za správný) na základě výsledků různých variant.

Pro **neparalelní softwarové systémy** se používají následují tří hlavní schémata:

1. obnovovací bloky (OB) [43]
2. n-variantní programování (NVP) [44]
3. n-samokontrolní programování (NVP) [45].

Další schémata jsou (povětšinou) jen kombinacemi těchto schémat nebo jejich modifikacemi. Uvést lze např. distribuované obnovovací bloky [46], konsenzní obnovovací bloky [47], samokonfigurující optimální programování [48], certifikační stopy [49] nebo t/(n-1) variantní programování.

U **paralelních systémů** (kooperačních i konkurenčních) se navíc používají mechanismy jako atomické akce [50], atomické transakce [51], rozšířené konverzace [52] a koordinované atomické akce [53].

Přehled hlavních rysů těchto mechanismů najdete v tabulce 4.1.

metoda	souběžnost	závady	obnovení
konverzace [1976]	kooperační	závady návrhu	odrolování (rozman. SW)
atomické akce [1986]	kooperační	závady prostředí, závady návrhu	přerolování (výjimky), odrolování (rozman. SW)
atomické transakce [1965]	konkurenční	závady hardwaru	odrolování (opakování)
rozšířené konverzace [1991]	konkurenční kooperační	závady návrhu	odrolování (rozman. SW)
koordinované atomické akce [1995]	konkurenční kooperační	závady prostředí závady návrhu závady hardwaru	přerolování (výjimky) odrolování (rozman. SW, opakování)

Tabulka 4.1.: Porovnání mechanismů odolnosti proti závadám

V rozsáhlých webových aplikacích je nutno uvažovat i nekonzistentní selhání způsobené napadením systému. U těchto systémů lze pro zajištění odolnosti využít *byzantských algoritmů* [54].

Byzantské algoritmy řeší tzv. byzantský problém, který lze obecně definovat za využití vojenské terminologie:

Armáda řízená skupinou generálů obklíčí pevnost nepřátel. Každý generál je velitelem své vlastní vojenské jednotky a může se rozhodnout, jaké akce podnikne a jaké rozkazy vydá ostatním generálům (může tedy jednat nezávisle na ostatních generálech, může ale nemusí uposlechnout jejich rozkazů). Generálové se domlouvají prostřednictvím kurýrů (poslů) a kromě rozkazů si mohou vyměňovat i další informace.

Generálové se musí dohodnout na společné (koordinované) akci. Klasickým příkladem takovéto akce je společný útok. Situace je však komplikována existencí zrádců, tj. generálů, jejichž cílem je narušení koordinace (tj. mohou provádět opačné akce a chybně informovat své partnery). Systém však navzdory aktivitě zrádců musí zajistit koordinaci mezi loajálními generály.

Přesněji řečeno všichni loajální (= nezrazující) generálové musí mít algoritmus, který by garantoval, že :

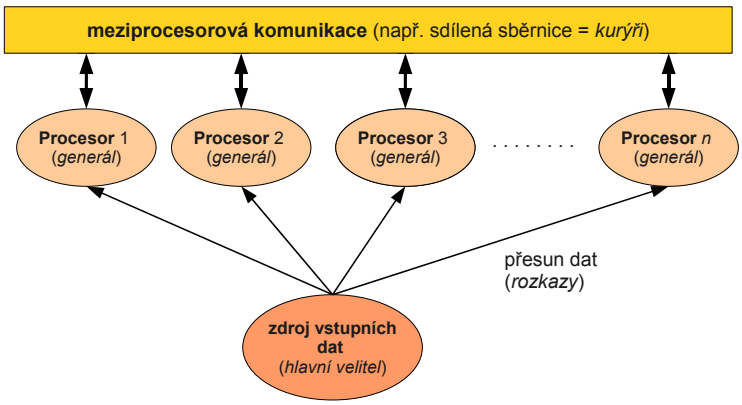
1. všichni loajální generálové podniknou stejnou akci (zrádci mohou podniknout akci libovolnou)
2. malý počet zrádců nesmí ohrozit plán, který provádějí loajální generálové.

Speciálním případem je situace, kdy rozkazy vydává pouze jediný generál (hlavní velitel). I ten však může být zrádcem. Pokud je však loajální, musí být akce vykonána loajálními generály v souhlasu s jeho rozkazem (jinak je vykonána akce opačná, resp. žádná). Je nutné si uvědomit, že loajalita hlavního velitele je definována jako konzistentní chování (velitel vydává stejné rozkazy a v rámci komunikace se chová konzistentně). Proradnost rozkazu (hlavní velitel může být ovlivněn nepřítelem), resp. jeho nezáměrnou chybnost, nelze v byzantském systému kontrolovat. Jinak řečeno loajální generálové musí vždy uposlechnout rozkaz konzistentně se chovajícího velitele.

Aplikace byzantského algoritmu ve výpočetních systémech může například spočívat v dosažení dohody o vstupních datech, která poskytuje vstupní komponenta (funkčně odpovídající hlavnímu veliteli), přičemž koordinovanou akcí je uložení (resp. nastavení) těchto dat u komponent-příjemců (odpovídají generálům). Komponentami mohou být např. procesory, servery replikovaných služeb, včetně např. replikovaných databází. Komponenty těchto systémů podléhají selháním, která jsou ve většině případů nekonzistentní (označuje se často jako tzv. byzantské selhání) a která odpovídají zrádcovskému chování ve vojenské terminologii.

Předpokládejme nejdříve systém s procesory, jež vzájemně komunikují a získávají data například ze vstupního portu, resp. IO procesoru (dále označeno jako

zdroj dat), a produkují výstup, který by měl být u všech modulů identický (viz obrázek 4.16). Tento systém může obsahovat redundantní (nadbytečné) procesory, které by nebyly nutné při běžném zpracování dat v systému bez byzantských selhání.



Obrázek 4.16.: Využití byzantských algoritmů na úrovni procesorů

Toto schéma je obdobou běžnějšího schématu *většinového hlasování*, které zajišťuje, že je zvolen výstup, který převládá. Toto schéma však předpokládá, že získaná data jsou shodná, a tudíž správná (jinak řečeno zdroj dat je v kontextu byzantského problému loajální generál). Naproti tomu systém užívající byzantských algoritmů uvažuje i možnost selhání zdroje dat, a proto musí tudíž zahrnovat i vzájemnou komunikaci procesorů .

Podobně lze uvažovat i systémy s replikami serverů, kde vstupními daty jsou zprávy od primárního serveru. Zprávy v tomto systému mohou být pozměněny, resp. dokonce podvrženy útočníkem, mohou být duplikovány, resp. mohou docházet, se zpožděním a samozřejmě mohou být chybně interpretovány servery-replikanty. I přesto se servery musí dohodnout na společném výstupu, tj. na shodně poskytované službě.

Vlastní popis byzantských algoritmů je mimo rámec této knihy, je však možno uvést alespoň základní charakteristiku. Lze totiž dokázat, že pokud je n počet generálů (počet komponent) a f počet zrádců (selhavších komponent), pak řešení byzantského problému existuje jen v případě, pokud platí $n \geq 3f + 1$.

Ve většině případů je však pro řešení problémů nekonzistence výhodnější použít **samodiagnostiku**, jíž jsme se zabývali v předchozích kapitolách. Samodiagnostika má potenciál odhalit větší počet různých chybových situací v systému a především rozlišit a identifikovat větší počet různých druhů selhání. Například v systému popsaném obrázkem 4.16 může samodiagnostika rozlišit selhání zdroje vstupních dat

(tj. selhání IO procesoru) od selhání komunikačního kanálu mezi procesory. Navíc může identifikovat, zda se jedná o selhání stálé, nahodilé nebo jen intermitentní. Za určitých podmínek může poskytovat důvěryhodný výsledek i pro systémy s větším počtem chybných komponentů (např. v případech, kdy je počet chybných komponentů větší než $\lfloor \frac{n-1}{2} \rfloor$). Další výhodou je rychlejší reakce na změnu v architektuře systému např. v případě omezení komunikace nebo poklesu počtu komponent. Pružnější je i při využití nadbytečnosti (redundance), kterou může, ale nemusí využívat.

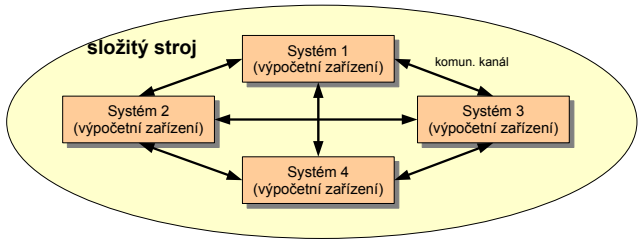
4.6. Možnosti využití samokontroly a samodiagnostiky ve výpočetních systémech

Konkrétní využití samodiagnostiky závisí v prvé řadě na charakteru systémových modulů a charakteru atomických kontrol, které se mezi nimi provádějí. Následující tabulka ukazuje základní typy diagnostických systémů z tohoto pohledu, včetně příslušné sémantiky atomických kontrol.

typ	modul (komponenta)	atomická kontrola
1	hardwarové výpočetní zařízení např. procesor	kontrola hardwaru a správnosti jím poskytovaných služeb
2	softwarový modul např. komponenta, třída.	porovnání výsledků jednotlivých softwarových komponent
3	softwarový agent	ohodnocení vlastností (schopností) agenta provedené jiným agentem
4	server v Internetu	ohodnocení stavu serveru provedené jiným serverem

Případ 1 — Hardwarová výpočetní zařízení

V tomto modelovém případě budeme uvažovat diagnostický systém, jehož moduly jsou hardwarová výpočetní zařízení a atomické kontroly představují kontroly jednotlivých zařízení. Tento typ diagnostických systémů se využívá u složitých strojů (letadla, automobily), které obsahují několik elektronických subsystémů, z nichž každý obsahuje výpočetní zařízení (procesor, vestavěný software). Subsystémy jsou obvykle propojené a vyměňují si informace. Na obrázku 4.17 je znázorněno strukturální schéma složitého zdroje, v němž může být každý subsystém uvažován jako modul v kontextu samodiagnostiky. Každý subsystém musí mít prostředky na provedení kontroly jakéhokoliv jiného subsystému v rámci stroje.



Obrázek 4.17.: Strukturální schéma složitého stroje

Atomické kontroly v tomto systému mohou být zaměřeny na:

1. odhalení hardwarových závad (kontrola technického stavu modulu)
2. odhalení softwarových závad výpočetního zařízení (kontrola spících závad)
3. kontrolu správnosti služby (tj. výsledku výpočtů provedených modulem).

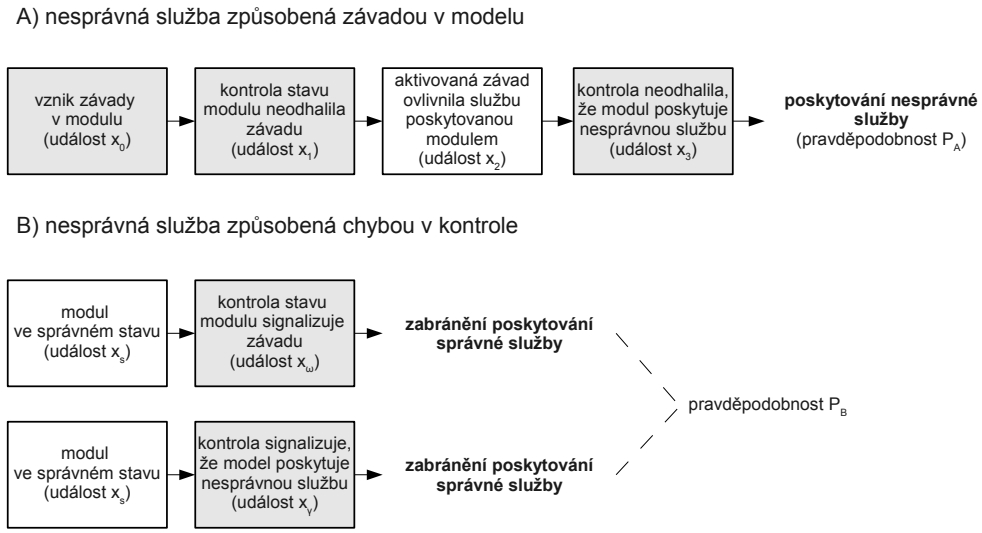
Jak již bylo výše uvedeno, existují dvě základní možnosti organizace atomických kontrol v systému. Kontroly mohou být prováděny v době, kdy je systém pozastaven, nebo souběžně s poskytováním služby. V prvém případě se jedná o kontrolu připravenosti výpočetního systému k poskytování služby, ve druhém pak o kontrolu správnosti poskytované služby. V další části se soustředíme na první případ, neboť je v něm přínos samodiagnostiky nejvýraznější.

Úplná kontrola správnosti poskytované služby může být dosažena jen za využití redundance a majoritních struktur. Službu (identickou) v tomto případě poskytuje několik systémů, správná služba se odvodí z jejich porovnání. Bohužel použití majoritních struktur je finančně náročné (redundance, tvorba hardwarové a softwarové podpory) a především může vnést do systému nové druhy závad. Proto je mnohdy výhodnější provést kontrolu neúplnou, která je výrazně jednodušší. Sníženou důvěryhodnost neúplné kontroly lze kompenzovat kontrolou připravenosti komponent systému.

Zajímavou strategií je proto kombinace kontroly připravenosti modulů, jež je prováděna prostředky samodiagnostiky, a jednodušší neúplné kontroly poskytované služby. Alternativní možností je stálá kontrola připravenosti modulů k poskytování služby spojená s kontrolu poskytované služby, jež může být přepínána mezi jednoduchou a úplnou kontrolou (v kritických fázích).

Efektivitu takovéto kombinace lze ohodnotit na základě rozboru událostí, které vedou k nesprávné funkci systému. Nejdříve je nutno zohlednit řetězec událostí, kdy nesprávnou službu systému způsobila závada v modulu. Řetězec událostí vedoucí k poskytování nesprávné služby je uveden na obrázku 4.18 na následující straně v části A.

Je však nutno si uvědomit, že i prostředky, které provádějí kontrolu modulu nebo kontrolu služby, jsou také nedokonalé, a mohou tudíž chybně vyhodnotit stav modulu nebo služby a tím zabránit v poskytování služby. Tato situace proto musí být též zohledněna při výpočtu pravděpodobnosti poskytování nesprávné služby (viz řetězce událostí obrázek 4.18, část B).



Obrázek 4.18.: Řetězce událostí vedoucí k poskytování nesprávné služby

Označme pravděpodobnost událostí, že poskytování služby se liší od plánovaného (tj. buď služba není správná anebo je poskytování služby přerušeno) jako P_{NS}. Tuto pravděpodobnost lze vypočíst ze vztahu:

$$P_{NS} = P\{A \cup B\} = P_A + P_B$$

kde:

$$P_A = P(x_0)P(x_1/x_0)P(x_2/x_0x_1)P(x_3/x_0x_1x_2) \tag{4.2}$$

$$P_B = P(x_s)[1-(1-P_\omega)(1-P_\gamma)] \tag{4.3}$$

Lze snadno nahlédnout, že pravděpodobnosti uvedené ve vztazích (4.2) a (4.3) lze vyjádřit takto:

$$
\begin{aligned}
P(x_0) &= 1 - P(x_s) \\
P(x_1/x_0) &= \beta \\
P(x_2/x_0x_1) &= P_A \\
P(x_3/x_0x_1x_2) &= P_{KSS} \\
P_\omega &= \alpha_1 \\
P_\gamma &= \alpha_2
\end{aligned}
$$

kde:

P_A	—	pravděpodobnost aktivace závady
P_{KSS}	—	pravděpodobnost události, že kontrola správnosti poskytované služby neodhalí odchylky služby
α_1	—	chyba prvního druhu kontroly stavu modulu
α_2	—	chyba prvního druhu kontroly služby (zdánlivá chyba)
β	—	chyba druhého druhu (nedetekování chyby)

Po dosazení:

$$P_{NS} = (1 - P(X_s)) \cdot P_{KSS} \cdot \beta \cdot P_A + P(X_S)[1 - (1 - \alpha_1)(1 - \alpha_2)]$$

V případě, kdy se kontrola připravenosti k poskytování služby provádí prostřednictvím samokontroly, je možno předpokládat, že $\alpha_1 = 0$, $\alpha_2 = 0$, $\beta = 1 - P_{AT}$, kde P_{AT} je důvěryhodnost atomických kontrol. Chyby prvního druhu α_1 a α_2 jsou nulové, neboť v systému, kde jsou všechny moduly správně, nemůže atomická kontrola provedená správným modulem označit správný modul za chybný.

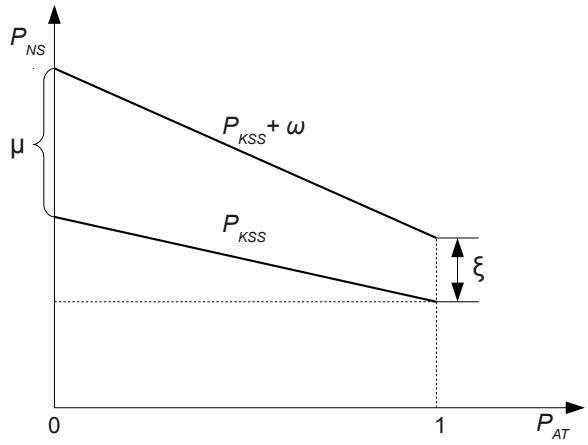
Po zohlednění tohoto předpokladu lze výraz dále zjednodušit:

$$P_{NS} = P(X_S) + k - kP_{AT}$$

kde $k = P_{KSS}(P_A - P_A P(X_S))$

Schéma funkcionální závislosti $P_{NS} = \varphi(P_{AT})$ pro dvě různé hodnoty P_{KSS} je zobrazeno na obrázku 4.19 na následující straně.

Z obrázku lze odvodit, že v případě systému s vysokou důvěryhodností samodiagnostiky ($D \to 1$) mohou být sníženy požadavky na „drahou“ kontrolu správnosti poskytované služby (zde je například pravděpodobnost neodhalení P_{KSS} zvýšena o hodnotu ω) za cenu nepříliš velkého zvýšení pravděpodobnosti poskytování nesprávné služby P_{NS} (viz přírůstek ξ na obr. 4.19).



Obrázek 4.19.: Závislost chybného poskytování služby na důvěryhodnosti AT

Případ 2 — Softwarové moduly

Rozmanitost (diverzita) návrhu, jež se používá pro zajištění odolnosti systému proti závadám, vyžaduje pro svou implementaci v reálných systémech porovnání výsledků poskytovaných různými variantami softwarových modulů (může to být např. aplikace, softwarová komponenta, server, databáze atd). Pro mnohé softwarové moduly je procedura porovnání výsledků netriviálním úkolem využívajícím komplexní algoritmy (tj. nestačí například porovnávání na úrovni bytů). Z tohoto důvodu nemohou být porovnávače provádějící porovnání výsledků různých variant softwarových modulů považovány za úplně spolehlivé.

Vzhledem k této situaci je důležité zredukovat celkový počet porovnávačů v adjudikátorovém mechanismu. Tradiční mechanismy (adjudikátory) využívající porovnávačů nejsou příliš vyhovující vzhledem k poměru „složitost × efektivita“.

Například adjudikátor používaný ve schématu NVP je mnohem složitější než jednoduché schéma většinového (majoritního) hlasování. Naproti tomu adjudikátor používaný ve schématu NSCP je příliš jednoduchý, a není tudíž schopen odhalit korelované závady, které mohou vzniknout v aktivních samokontrolujících komponentech. V poslední době byl navržen velmi jednoduchý adjudikátor na základě t/(n-1)–diagnostiky (viz kapitola 1.7, [6]). Je však nutno poznamenat, že výsledkem činnosti tohoto adjudikátoru je pouze nalezení správného SW modulu, a nelze jej tudíž použít pro detekci modulů chybných.

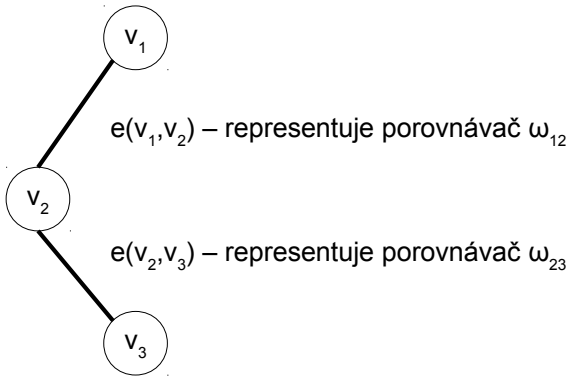
Jak již bylo řečeno výše má samodiagnostika na systémové úrovni potenciál nejen detekovat správný softwarový modul, ale také odhalit všechny nesprávné moduly

stejně jako nesprávné porovnávače. Kromě toho v určitých situacích umožňuje samodiagnostika nalézt softwarový modul, který může být s velkou pravděpodobnosti považován za správný, i když je celkový počet nesprávných modulů větší než celkový počet modulů správných (za cenu zvýšení složitosti adjudikátoru). Tudíž v každém konkrétním případě je nutno uvážit poměr „složitost × efektivita“ a pak učinit rozhodnutí, v jakém rozsahu je možno použít samodiagnostiku při návrhu adjudikátoru.

Při využití samodiagnostiky v adjudikátorovém mechanismu je nutno vycházet z modifikovaného diagnostického modulu, který začleňuje podporu porovnávačů.

Diagnostický model systému s porovnávači

Množinu modulů systému S označme $U = \{u_1, u_2, \ldots, u_n\}$. Každý modul u_i, $u_i \in U$ produkuje výstup, který je předáván porovnávačům. Porovnání výstupů dvou modulů u_i a u_j je prováděno porovnávačem, jenž je označen jako ω_{ij}. Strukturu všech porovnání lze znázornit podobně jako u diagnostického modelu z prvé kapitoly pomocí diagnostického grafu $G = (V, E)$. V grafu je každý modul systému u_i, $i \in 1 \ldots N$ representován vrcholem $v_i \in V$ a každý porovnávač ω_{ij} pak hranou $e(v_i, v_j) \in E$. Na obrázku 4.20 je ukázka diagnostického grafu systému se třemi moduly a dvěma porovnávači.

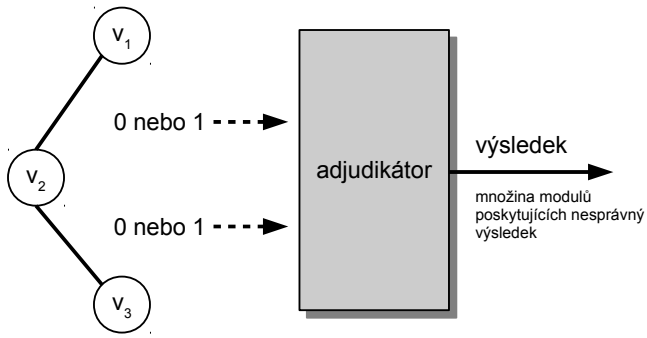


Obrázek 4.20.: Diagnostický graf modelu systému s porovnávači

Výše uvedený model je obecný, což znamená, že modul diagnostického modelu může odpovídat různým softwarovým entitám (od softwarových modulů, resp. tříd, přes databáze až po komplexní webový server).

Výsledkem činnosti porovnávače je buď hodnota 0 (výstupy jsou stejné) nebo 1 (výstupy jsou rozdílné, neúspěch). Tento výsledek lze v diagnostickém grafu vyjád-

řit jako ohodnocení hrany, jež odpovídá danému porovnávači. Výsledky porovnání se ode všech porovnávačů předávají do adjudikátoru jako jeho vstupní data (adjudikátor je softwarový modul nebo program), viz obrázek 4.21.



Obrázek 4.21.: Role adjudikátoru v diagnostickém modelu systému

Na základě těchto vstupních dat adjudikátor určí, které z modulů poskytují nesprávné výsledky (tj. detekuje událost selhání modulu). Příčina selhání modulu (tj. závada) se v rámci popisovaného diagnostického modelu nezjišťuje.

Obecně může být nesprávný výsledek modulu způsoben buď hardwarovou závadou nebo závadou návrhu. V případě hardwarových závad se rozlišují závady permanentní, intermitentní nebo krátkodobý jednorázový výpadek. Pokud modul neposkytuje žádný výsledek, což se může stát v případě kolapsu hostitelského prostředí, pak se tento stav interpretuje jako nesprávný výstup modulu.

Při návrhu adjudikátoru je možno vzít v potaz různé předpoklady ohledně situace, kdy porovnávač vrací výsledek 0, tj. označí výstupy modulu za shodné. V jednoduchém případě je možno předpokládat, že dva vadné moduly nemohou produkovat shodný výstup. Pokud za tohoto předpokladu vrací porovnávač hodnotu 0, pak lze moduly s jistotou označit za správné. Takový předpoklad však není ve většině případů platný, neboť chybné moduly mohou běžně produkovat stejný nesprávný výstup. Příčinou může být například stejná závada v obou modulech, tzv. *korelovaná závada* (angl. *related*). Z pozitivního výsledku porovnání (= 0) nelze v tomto případě přímo vyvodit správnost obou modulů.

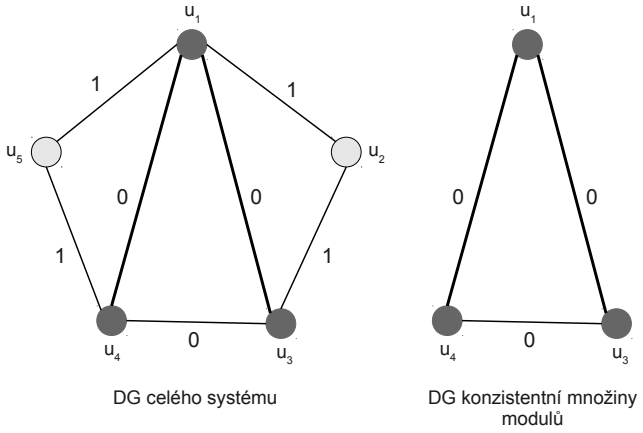
Při návrhu diagnostického algoritmu pro adjudikátor je možno použít koncepce tzv. „konzistentních množin modulů“.

Konzistentní množiny modulů

Libovolnou (pod)množinu modulů systému označíme jako **konzistentní**, pokud pro ni platí následující tvrzení :

1. výsledek porovnání výstupů libovolné dvojice modulů z této množiny je rovno 0 (= shoda).
2. výsledek porovnání výstupů kteréhokoliv modulu z množiny s výstupem modulu, který není prvkem této množiny, je roven 1 (= neshoda).
3. v odpovídajícím diagnostickém grafu je tato množina představována souvislým podgrafem.

Příklad konzistentní množiny modulů pro systém pěti modulů je znázorněn na obrázku 4.22 .



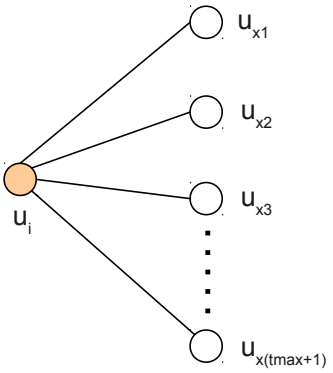
Obrázek 4.22.: Diagnostický graf s konzistentní množinou modulů

Východiskem pro návrh tohoto diagnostického algoritmu je základní předpoklad, že správný výsledek diagnostiky může být garantován jen v případě, pokud je celkový počet správných modulů v systému větší než počet modulů chybných.

Pokud tento předpoklad přijmeme, můžeme snadno odvodit, že pokud je celkový počet modulů konzistentní množiny větší nebo roven $\lceil N/2 \rceil$, lze všechny moduly této množiny považovat za správné. Proto se konzistentní množina k modulů, kde $k \geq \lceil N/2 \rceil$, označuje jako *konzistentní množina správných modulů* Y_{SM}.

Podle výše uvedeného základního předpokladu je nejvyšší přípustný počet nesprávných modulů t_{max} roven $N - \lceil \frac{N}{2} \rceil$. Lze snadno prokázat, že správné ohodnocení kteréhokoliv modulu systému je možné pouze tehdy, pokud se výstup modulu po-

rovná s alespoň $(t_{max}+1)$ moduly (obr. 4.23). Jen tak je totiž zajištěno, že bude porovnán s výstupem alespoň jednoho správného modulu.



Obrázek 4.23.: Minimální počet porovnání v DG

Diagnostický graf, ve kterém má každý modul právě $t_{max}+1$ porovnání, se nazývá *diagnostický graf se základní strukturou porovnání*. Minimální počet porovnávačů P_{min}, které jsou nutné, aby měl graf základní strukturu porovnání, je roven:

$$P_{min} = \lceil N(t_{max}+1)/2 \rceil$$

Je zřejmé, že důvěryhodnost výsledků diagnostiky systému bude větší, když budou všechny správné moduly součástí konzistentní množiny správných modulů Y_{SM}. V článku [55] je dokázáno, že pro systémy s $N < 7$ pro to postačuje diagnostický graf se základní strukturou porovnání. Pro systémy s $N \geq 7$ není základní struktura porovnání postačující, tj. některé správné moduly mohou ležet mimo Y_{SM}. V tomto případě je nutné přidat další porovnávače.

Vztahy pro zjištění počtu dodatečných porovnávačů včetně algoritmů pro vytvoření příslušných diagnostických grafů jsou uvedeny v [55]. Tabulka 4.2 na následující straně převzatá z tohoto zdroje uvádí celkový počet porovnávačů nutných pro zajištění základní struktury porovnání (případ 1), pro zajištění stavu, kdy jsou všechny správné moduly prvky Y_{SM} (případ 2), a pro srovnání počet porovnávačů nutných pro zajištění majoritního hlasování (porovnávač pro každou dvojici modulů, případ 3).

Koncepci „konzistentní množiny modulů“ lze využít i v případě, pokud počet nesprávných modulů překročí hodnotu t_{max}. Jestliže lze navíc předpokládat, že většina závad *není* korelována, pak se největší konzistentní množina modulů bude s vysokou pravděpodobností skládat ze správných modulů. U menších množin je tato pravděpodobnost výrazně nižší.

N	případ 1	případ 2	případ 3
3	3	3	3
5	8	8	10
7	14	15	21
9	23	24	36
11	33	35	55
13	46	48	78
15	60	63	105
17	77	80	136

Tabulka 4.2.: Počet porovnávačů pro různé algoritmy adjudikátoru

Případ 3 – Softwarový agent

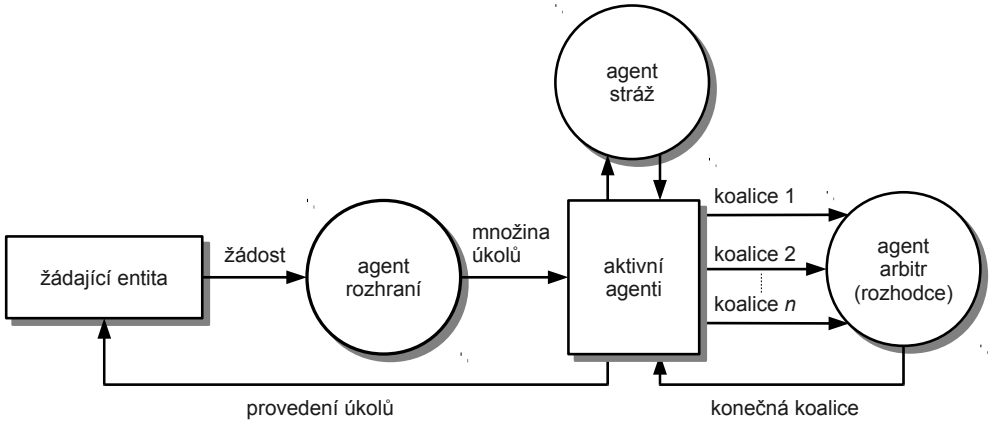
Použití jednotlivých izolovaných softwarových aplikací, resp. jiných inteligentních jednotek (používá se zde obecný termín agent), nemusí stačit pro řešení skutečně komplexních problémů. V takových případech je však možno spojit více agentů a vytvořit **multiagentní systém** (MAS). Kromě nejčastěji uvažovaných softwarových agentů lze jako agenty uvažovat i roboty, podniky, organizace, týmy (včetně vojenských jednotek) resp. jednotlivé lidi.

Jednotlivé prvky organizace samodiagnostiky (kapitola 2) mohou být využity i v oblasti multiagentních systémů například v procesu formování koalic agentů. Níže budeme předpokládat tzv. *kooperační multiagentní systémy*, tj. systémy, ve kterých mají agenty větší zájem na společném cíli než na vlastním prospěchu (podrobnosti viz např. [56]).

Formování koalic agentů je komplexní proces, který vyžaduje účast několika typů specializovaných agentů. V modelu znázorněném na obrázku 4.24 na následující straně je to agent rozhraní, agent stráž a agent arbitr.

V roli žádající entity obvykle vystupuje agent, který vyjadřuje (resp. zastupuje) zájem určité entity např. jednoduchého klienta či celé velké organizace. Aktivními agenty mohou být jak bylo již výše řečeno nejen softwarové aplikace a moduly ale i skupiny lidí (např. záchranářské týmy).

Koalice se formuje jako reakce na žádost oprávněné entity. V závislosti na úkolech, které mají být provedeny pro splnění žádosti, rozhoduje každý jednotlivý agent o prostředcích, které by mohl poskytnout (tj. jak může přispět ke splnění úkolů). Je zřejmé, že je velmi důležité, aby každý aktivní agent obdržel správnou a přesnou



Obrázek 4.24.: Součinnost agentů při formování koalice

informaci ohledně úkolů, které musí být provedeny. Pro vyloučení nesrovnalostí v pochopení úkolů jednotlivými agenty lze použít specializovaného agenta rozhraní. Úkolem tohoto agenta je převedení žádosti na množinu konkrétních úkolů. Takový postup zaručuje, že všechny agenty obdrží stejnou informaci ohledně úkolů a tudíž mohou tyto úkoly správně interpretovat.

V průběhu formování koalice si agenty vyměňují informace o službách a prostředcích, které mohou poskytnout pro vznikající koalici. Agent může v průběhu formování koalice tyto informace měnit za účelem zvýšení vlastních šancí na účast ve finální koalici. Agent-stráž je zodpovědný za to, že tyto informace budou dostupné jen agentům, které předběžně souhlasily se vzájemnou komunikací (sociální systémy) resp. jsou komunikačně kompatibilní (technické systémy). Kromě toho zajišťuje, že (potenciálně citlivé) informace nepřekročí hranice multiagentního systému a nemohou tak být vně systému zneužity.

Agenty v rámci tohoto procesu navrhnou několik potenciálních koalic, z nichž každá je schopna splnit veškeré požadované úkoly. Pro volbu koalice, která bude vposledku pověřena provedením úkolů (konečná resp. finální koalice), se využívá nezávislý agent rozhodce (resp. arbitr). Tento agent využívá informací o způsobilosti každé koalice-kandidáta a kritéria, která jsou stanovena pro plán provedení úkolů koalicemi. Agent rozhodce je též zodpovědný za informování agentů zúčastněných ve finální koalici o svém rozhodnutí, čímž povoluje zahájit jejich vykonávání. Obě specializované funkce, tj. funkci stráže i rozhodce, může samozřejmě vykonávat jediný agent (viz Meta-agent [57]).

V průběhu formování koalicí se musí každý aktivní agent rozhodnout, kdy a s kte-

rým agentem bude komunikovat. Toto rozhodování závisí na jeho znalosti způsobilostí ostatních agentů a na globální strategii agentů. Tyto strategie i komunikační protokoly vycházejí z výše uvedeného předpokladu, že agenty v kooperujícím multiagentním systému mají zájem především na tom, aby byly úkoly provedeny s maximální efektivitou a za nejkratší možný čas, nikoliv na dosažení svých individuálních zájmů.

To mimo jiné znamená, že:

1. agenti se chovají v rámci vzájemné komunikace poctivě (např. uvádějí jen pravdivé údaje o své způsobilosti)
2. pokud agent přijme nabídku na účast v koalici, pak nesmí toto rozhodnutí následně měnit
3. agent musí co nejdříve odpovědět na dotazy ostatních agentů.

V případě, kdy agenti tyto požadavky nedodržují, bude celkový proces formování výrazně zkomplikován, což může vést k výraznému zvýšení času potřebného na formování koalic, a to až na neakceptovatelnou úroveň (kritické např. u záchranářských akcí).

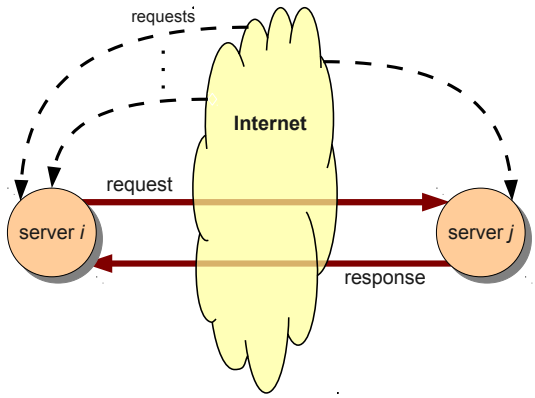
Běžně používané multiagentní systémy jsou obecně velmi zranitelné, neboť může docházet k (nezlovolným) selháním jednotlivých agentů. Závažné je především selhání specializovaných agentů (např. rozhodce). Proto se vývojáři multiagentních systémů snaží funkce specializovaných agentů distribuovat mezi běžné agenty (tj. funkce agentů není určena předem). Například aktivní agenty mohou zvolit agenta, který bude vykonávat funkci rozhodce, mezi sebou. Proces volby je přitom obdobou procesu, který probíhá při samodiagnostice složitého systému, kdy je zjišťován modul poskytující výsledek diagnostiky systému (např. pomocí algoritmu putujícího jádra). Proto je možno algoritmy uvedené v této kapitole využít i v oblasti formování koalic multiagentních systémů.

Případ 4 – Server v Internetu

Rychlý rozvoj internetových technologií na konci let devadesátých vedl ke vzniku nového typu distribuovaných systémů – webových služeb. První návrhy webových služeb se objevily na přelomu tisíciletí a od té doby bylo navrženo a implementováno obrovské množství prakticky využívaných webových služeb.

Dalším vývojovým krokem v oblasti webových služeb je skládání více méně jednoduchých webových služeb do podoby pokročilých distribuovaných aplikací (přičemž jednotlivé služby zůstávají autonomní a lze je využívat i přímo). Tento postup se označuje jako *kompozice webových služeb* (angl. *web service composition*), viz [58] resp [59].

Během poskytování této pokročilé služby spolu komunikují jednotlivé webové servery, přičemž vystupují buď jako servery (= poskytovatelé elementárních služeb) nebo jako klienti (uživatelé elementární služby), viz obr. 4.25.



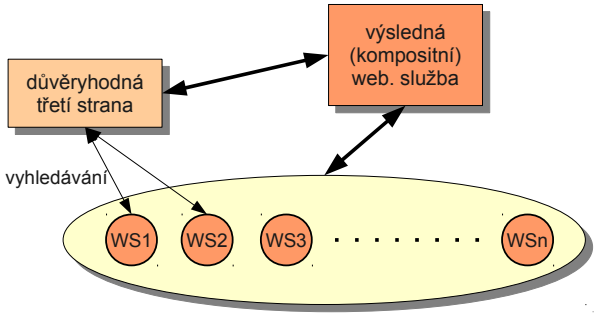
Obrázek 4.25.: Webová služba typu *request/response*

Server i vystupuje na obrázku 4.25 jako klient, tj. odesílá požadavek na server j, jenž hraje roli serveru v rámci modelu klient-server. Server j požadavek zpracuje a klientovi zašle odpověď. Komunikace se děje prostřednictvím internetu a oba servery musí souběžně reagovat i na požadavky dalších serverů, tj. v rámci poskytování kompozitní služby i mimo ni.

Po ukončení komunikace může server i provést ohodnocení zatíženosti serveru j prostřednictvím celkové doby odezvy (čas mezi vysláním požadavku a příjmem odpovědi). Zjištění celkové doby odezvy nevyžaduje žádnou změnu protokolu komunikace mezi oběma servery. V případě, že čas odezvy překročí stanovený limit, může server i o této skutečnosti informovat tzv. důvěryhodnou třetí stranu (angl. *trusted third party*), která je zodpovědná za vyhledávání a angažování jednotlivých webových služeb (tj. vytváření kompozice), viz obr 4.26 na následující straně. Při příštím sestavování komplexní služby (v terminologii webových služeb *orchestraci*) může být pomalý server eliminován či nahrazen.

Problémem je však skutečnost, že hodnocení provedené serverem i nemůže být bráno jako důvěryhodné. Důvodem je skutečnost, že ke zpoždění obou zpráv (požadavku i odpovědi) může dojít i na straně serveru i. Například odpovědní zpráva může čekat ve frontě zpráv, dokud server i nevyřídí jiné požadavky z Internetu (například od svých klientů). Překročení limitní doby odezvy tak může být důsledkem nadměrného zatížení jak serveru j tak i serveru i.

Hodnocení doby odezvy tak svým charakterem odpovídá atomickým kontrolám samodiagnostiky, a lze je tudíž i podobně zpracovávat. Třetí strana může na základě



Obrázek 4.26.: Složená webová služba a důvěryhodná třetí strana

množiny hodnocení jednotlivých serverů (odpovídá syndromu v diagnostice) s určitou úrovní důvěryhodnosti stanovit množinu zatížených serverů a tento údaj zohlednit v další kompozici. Může přitom používat algoritmů, které jsou velmi podobné algoritmům používaným při samodiagnostice složitých systémů.

A. Přehled základní notace

Následující tabulka shrnuje *základní* symboly užívané v rámci více kapitol či průběžně v celé knize.

notace	význam	stránka
M_i	i-tý modul (kontrolující)	13
M_j	j-tý modul (kontrolovaný)	13
N	počet modulů v systému	14
P_{AT}	důvěryhodnost AT ($r_{ij} = 1$ pro správné M_i a chybné M_j)	23
$P_k(t)$	pravděpod. kontroly všech modulů časovém intervalu t	75
P_D	důvěryhodnost (cyklu) samokontroly	85
R	syndrom	13
r_{ij}	výsledek atomické kontroly τ_{ij} ($r_{ij} \in R$)	13
t	maximální počet chybných modulů v DG	14
t_{max}	nejvyšší dosažitelné t pro daný systém	14
t	čas	
t_{max}^{SK}	limitní čas pro samokontrolu	108
T_{max}	limitní čas pro samokontrolu, samodiagn. a obnovu	90
t_c	doba trvání cyklu samokontroly	74
t_{AT}	doba trvání atomické kontroly	26
τ_{ij}	atomická kontrola (modul M_i kontroluje M_j)	55
τ_i	i-tá atomická kontrola (v posloupnosti AT)	26

Seznam obrázků

Literatura

[1] PREPARATA, F; METZE, G.; CHIEN, R. On the Connection Assignment Problem at Diagnosable System. *IEEE Transactions on Electronic Computers*. vol. EC-16, No. 12. 1967. pp. 848-854.

[2] BARSI, T.; GRANDONI, T.; MAESTRANI, P. A Theory of Diagnosability of Digital Systems. In *IEEE Trans. on Comp*. Vol. C-25, No. 6. 1976. pp. 585-593.

[3] CVETKOVIĆ, D.M.; DOOB, M.; SACHS, H. *Spectra of graphs*. 3rd revised and enlarged edition. Heidelberg/Leipzig: Johann Ambrosius Barth Verlag, 1995. 447 p. ISBN: 3-335-00407-8

[4] VEDESHENKOV, V. On Organization of Self-diagnosable Digital Systems. *Automation and Computer Engineering*. vol. 7. 1983. pp. 133-137.

[5] FRIEDMAN, A. A new measure of digital systems diagnosis. In *Proc. Int. Symp. Fault-tolerant computing*. Paris (France), 1975. pp. 167-170.

[6] XU, J. The t/(n-1)-diagnosability and its application to fault tolerance. *Technical report*. series No. 340. University of Newcastle upon Tyne, 1991.

[7] FUJIWARA, H.; KINOSHITA, K. Some Existence Theorems for probabilistically Diganosable Systems. *IEEE Trans. on. Comp*. Vol. C-27, No. 4. 1981. pp. 297-303.

[8] FIŠER, J.; MASHKOV, V. Generic Model For Application of Probabilistic Algorithms for System Self-diagnosis. In *Intellectual Systems for Decision Making and Problems of Computational Intelligence. ISDMCI'2010*: *Conference Proceedings*. Vol. 1. Kherson: KNTU, 2010. pp. 215-218.

[9] XU, J. *Sequentially t-diagnosable Systems*. Technical Report Series no. 481. Newcastle upon Tyne: University of Newcastle upon Tyne, 1994. 24 p.

[10] CHANG, Y-M.; YEH, J-M. A new sequential diagnosis algorithm in hypercubes with high diagnosibility. In *International Computer Symposium*. Taipei (Taiwan), 2004.

[11] KHANNA, S.; FUCHS, W.K. A graph partitioning approach to sequential diagnosis. *IEEE Trans. Comp*. 46. 1997. pp. 39-47.

[12] MALLELA, S.; MASSON, G. Diagnosable Systems for Intermittent Faults. *IEEE Trans. on Comp*. Vol. C-27. 1978. pp. 379-384.

[13] Машков, В.А. Самодиагностика бортовых вычислительных систем. Киев: КВВАИУ, 1989.

[14] MASHKOV, V.; MASHKOV, O. Providing complex Systems Fault-tolerance on the Basis of their Diagnosis on the Principle of Wandering Nucleus. *Cybernetics and Computing Technology (Argatic Systems)*. No. 116. Allerton Press, 1999. pp. 88-94.

[15] MASHKOV, V.; BARABASH, O. Self checking of modular systems under random performance of elementary checks. *Engeneering Simulation*. vol. 12. 1995. pp. 433-445.

[16] MOORE, E.F.; SHANNON, C.E. Reliable circuits using less reliable relays. *J.Franklin Institute*. 262. Sept/Oct. 1956. pp. 191-208 and 281-297.

[17] GOSTEV, V.; MASHKOV, V.; MASHKOV, O. Self Diagnosis of Modular Systems in Random Performance of Elementary Tests. *Cybernetics and Computing Technology (Discrete Control System)*. No, 105. Allerton Press, 1995. pp. 104-111.

[18] KLEINROCK, L. *Queueing systems*: *Theory*. Volume 1. New York: Wiley-Interscience, 1975. 432 p. ISBN: 0471491101

[19] DHILLON, B.S. *Engineering Reliability*. John Wiley & Sons, 1981. 318 p.

[20] COX, D. *Renewal Theory*. London: Methuen & Co, 1970. 142 p. ISBN: 041220570X

[21] MASHKOV, V.; BARABASH, O. Self-checking and self diagnosis of modular systems on the principle of walking diagnostic kernel. *Engineering simulation*. Vol. 15. 1998. pp. 43-51.

[22] FRY, C. The Theory of Probability as Applied to Problems of Congestion. In Nostrand D. van (edit.). *Probability and its Engineering Uses*. Princeton (N.Y), 1928.

[23] PALM, C. Intensitätsschwankungen im Fernsprechverkehr. *Ericsson Technics*. 44. 1943. pp. 1-189.

[24] Хинчин, А.Я. Математические методы теории массового обслуживания. Труды математического института им. В.А. Стеклова. т. 49. АН СССР, 1955.

[25] LAPRIE, J.C. *Dependability*: *Basic concepts and Terminology*. Springer-Verlag, 1992. ISBN: 0387822968.

[26] AVIŽIENIS, A.; LAPRIE, J.-C.; RANDELL, B. *Dependability and its Threats*: *A Taxonomy* [online]. [cit. 2010-09-03]. Available from WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.5946&rep=rep1&type=pdf>.

[27] AVIŽIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transaction on Dependable and Secure Computing*. Vol. 1. 2004. pp. 11-33.

[28] AVIŽIENIS, A. Design of fault-tolerant computers. In *Proc. 1967 Fall Joint Computer Conf*. AFIPS Conf. Proc. Vol. 31. 1967. pp. 733-743.

[29] NEUMANN, J. von. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Annals of Math Studies*. numbers 34. Princeton Univ. Press, 1956. pp. 43-98.

[30] AVIŽIENIS, A.; LAPRIE, J.; RANDELL, B. Fundamental Concepts of Depandability. *Research report*. No. 1145. LAAS-CNRS, April 2001.

[31] *Malicious and Accidental Fault Tolerance in Internet Applications*: *Conceptual Model and Architecture of MAFTIA*. Project IST-1999-11583, MAFTIA deliverable D21. 2003.

[32] HARROLD, M.J; ROTHERMEL, G.; WU, R.; YI, L. An empirical investigation of program spectra. In *Proc. of the SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Egineering*. Montreal (Canada), 1998. pp. 83-90.

[33] CHEN, M.; KICIMAN, E.; FRATKIN, E.; FOX, A.; BREWER, E. Pinpoint: Problem determination in large, dynamic internet services. In *Proc. of the International Conf. on Dependable Systems and Networks*. Washington: IEEE Computer Society, 2002. pp. 595-604.

[34] JONES, J.A.; HARROLD, M.J. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proc. of the 20th IEEE/ACM int. Conference on Automated software engineering*. New York (USA): ACM Press, 2005. pp. 273-282.

[35] SILVA MEYER, A.; FRANCO FARCIA, A.; PEREIRA DE SOUZA, A. Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L). *Genetics and Molecular Biology*. 27(1). 2004. pp. 83-91.

[36] DALLMEIER, V.; LINDIG, C.; ZELLER, A. Lightweight defect localization for Java. In Black A.P. (edit.) *ECOOP 2005: 19th European Conference, Glasgow, UK, July 25-29, 2005, Proceedings*. volume 3568 of LNCS. Springer-Verlag, [2005]. pp. 528-550.

[37] Hewlett-Packard. *Overview of perfmon kernel interface* [online]. c2009, [cit. 2010-09-20]. Available from WWW: <`http://www.hpl.hp.com/research/linux/perfmon/perfmon.php4`>.

[38] RENIERIS, M.; REISS, S.P. Fault localization with nearest neighbor queries. In *Proc.of the 18th IEEE International Conf. on Automated Software Engineering*. Montreal (Canada): IEEE Computer Society, 2003.

[39] AGRAWAL, H.; DEMILLO, R.A.; SPAFFORD, E.H. Debugging with dynamic slicing and backtrack. *Software Practice and Experience*. 23(6). 1993. pp. 589-616.

[40] ZELLER, A. Isolating cause-efect chains from computer programs. In *Proc. of the 10th international symposium on the Foundations of Software Engineering*. Charleston (South Carolina): ACM Press, 2002.

[41] GUPTA, Y. *Effect of Software Verification and Validation on Projects* [online]. [cit. 2010-09-06]. Available from WWW: <`http://www.buzzle.com/articles/effect-of-software-verification-validation-on-projects.html`>.

[42] LITTLEWOOD, B. The Problems of Assessing Software Reliability. In *Proceedings of SCSS-2000*. Springer-Verlag, 2000. Available from WWW: <`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9831&rep=rep1&type=pdf`>.

[43] RANDELL, B. System structure for software fault tolerance. *IEEE Trans. Software Engineering*. Vol. SE-1, No. 2. 1975. pp. 220-232.

[44] AVIŽIENIS, A.; CHEN, L. On the implementation of N-version programming for software fault-tolerance during execution. In *Int. Conf. Computer Software and Applications*. New York (NY), 1977. pp. 149-155.

[45] LAPRIE, J.C; ARLAT, J.; BEOUNES, C.; KANOUN, K.; HOURTOLLE, C. Hardware and software fault-tolerance: Definition and analysis of architectural solutions. In *Proc. Int. Symp. Fault-Tolerant Computing*. Pittsburgh, 1987. pp. 116-121.

[46] HECHT, M.; ARGON, J.; HOCHHAUGER, S. A distributed fault tolerant architecture for nuclear reactor control and safety functions. In *Real-Time System Symp.* Santa Monica, 1989. pp. 214-221.

[47] SCOTT, R.K; GAULT, J.W.; MCALLISTER, D.F. The consensus recovery block. In *Total System Reliability Symp.* 1985. pp. 74-85.

[48] BONDAVELLI, A.; DI GIANDOMENICO, F.; XU, J. Cost-effective and flexible scheme for software fault-tolerance. *Comput. Syst. Science & Engineering*. No. 4. 1993. pp. 234-244.

[49] SULLIVAN, G.F.; MASSON, G.M. Certification trails for data structures. In *21st Int. Symp. Fault-Tolerant Computing*. Montreal, 1991. pp. 240-247.

[50] LEE, P.; ANDERSON, T. Fault-Tolerance Principles and Practice. *Dependable computing and Fault-Tolerant Systems*. 2nd edition. Springer Verlag, 1990.

[51] LYNCH, N.; MERRIT, M.; WEIHL, W; FECETE, A. *Atomic Transaction*. San Mateo (California): Morgan Kaufman, 1994. ISBN: 155860104X.

[52] STRIGINI, L.; DIGINDOMENICO, F. Flexible schemes for application-level fault-tolerance. In *Proc. 10th Int. Symp. On Reliable Distributed Systems. IEEE*. Pisa (Italy): CSPress, 1991. pp. 86-95.

[53] RANDELL, B.; ROMANOVSKY, A.; STROUD, R.; XU, J.; ZORZO, A. *Coordinated Atomic Actions: from Concept to Implementation*. Technical Report 595. Newcastle upon Tyne: University of Newcastle upon Tyne, 1997.

[54] LAMPORT, L.; SHOSTAR, R. The Byzantine Generals Problems. *ACM Transactions on Programming Languages and Systems*. V. 4, N.3. pp. 382-400.

[55] MASHKOV, V.; POKORNY, J. Scheme for comparing results of diverse software versions. In *Proc. of ICSOFT conference*. 2007. pp. 341-344.

[56] FIŠER, J.; MASHKOV, V. Alliance and Coalition Formation. *Journal of applied. computer science*. Vol. 18 No. 1. 2010. pp. 19-38.

[57] MAŘÍK, V.; PECHOUČEK, M.; ŠTĚPÁNKOVÁ, O. Social knowledge in Multi-agent systems. In *ACAI 2001*. LNAI 2086. 2001. pp. 211-245.

[58] W3C. *Web Services Choreography Description Language Version 1.0*: *W3C Working Draft* [online]. 17 December 2004, [cit. 2010-09-03]. Available from WWW: <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>.

[59] OASIS. *Web Services Business Process Execution Language Version 2.0*: *Primer* [online]. 9 May 2007, [cit. 2010-09-05]. Available from WWW: <http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm>.

Mashkov V., Fišer Jiří. *Samokontrola a samodiagnostika na systémové úrovni.* Lviv: Ukrainian Academic Press, 2010. 176. stran.

MDT: 681.518.54