Klasické plánování II

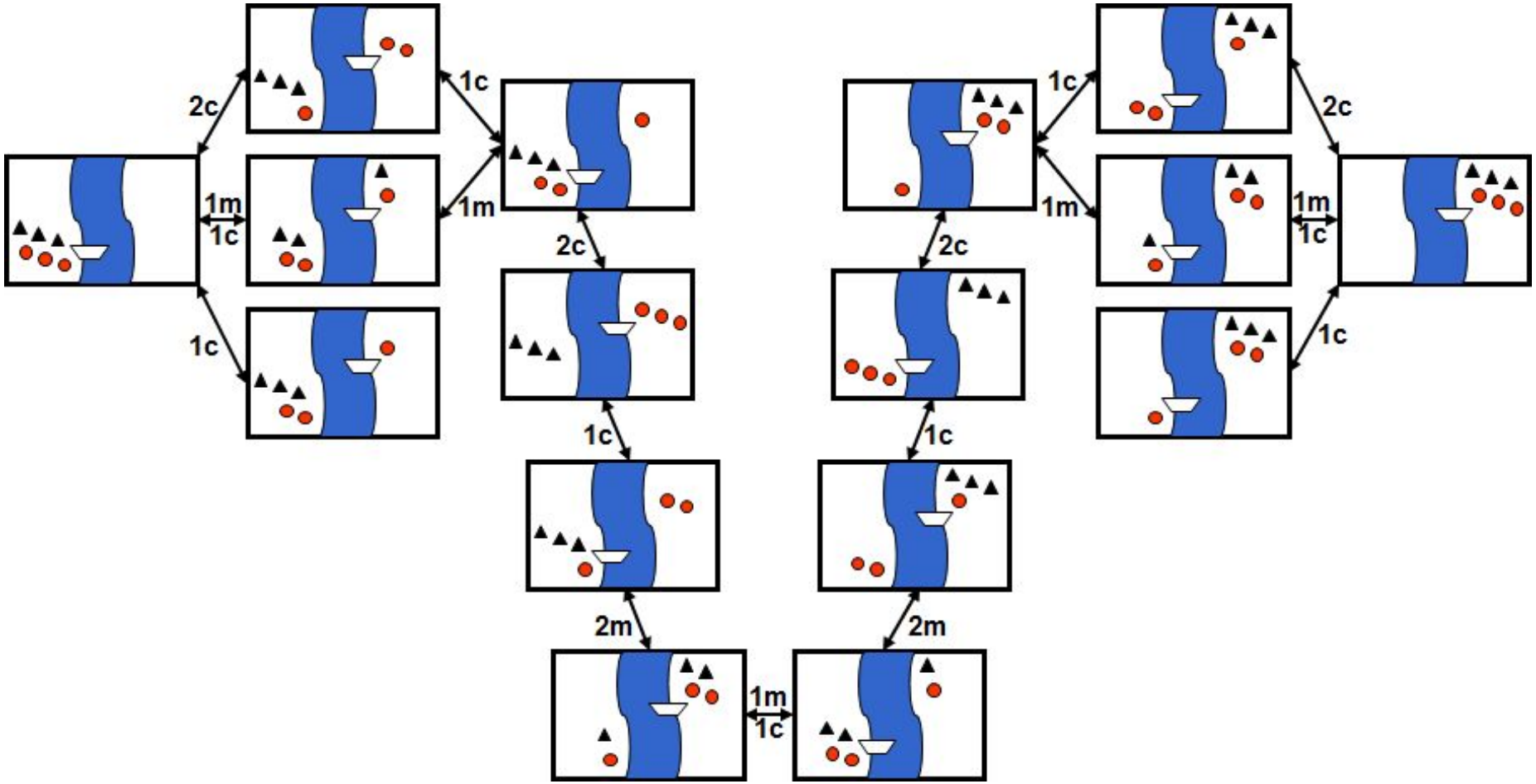
Radek Mařík

CVUT FEL, K13133

10. dubna 2012

Obsah

Prohledávání stavového prostoru [Wic11]



hra Misionáři a kanibalové

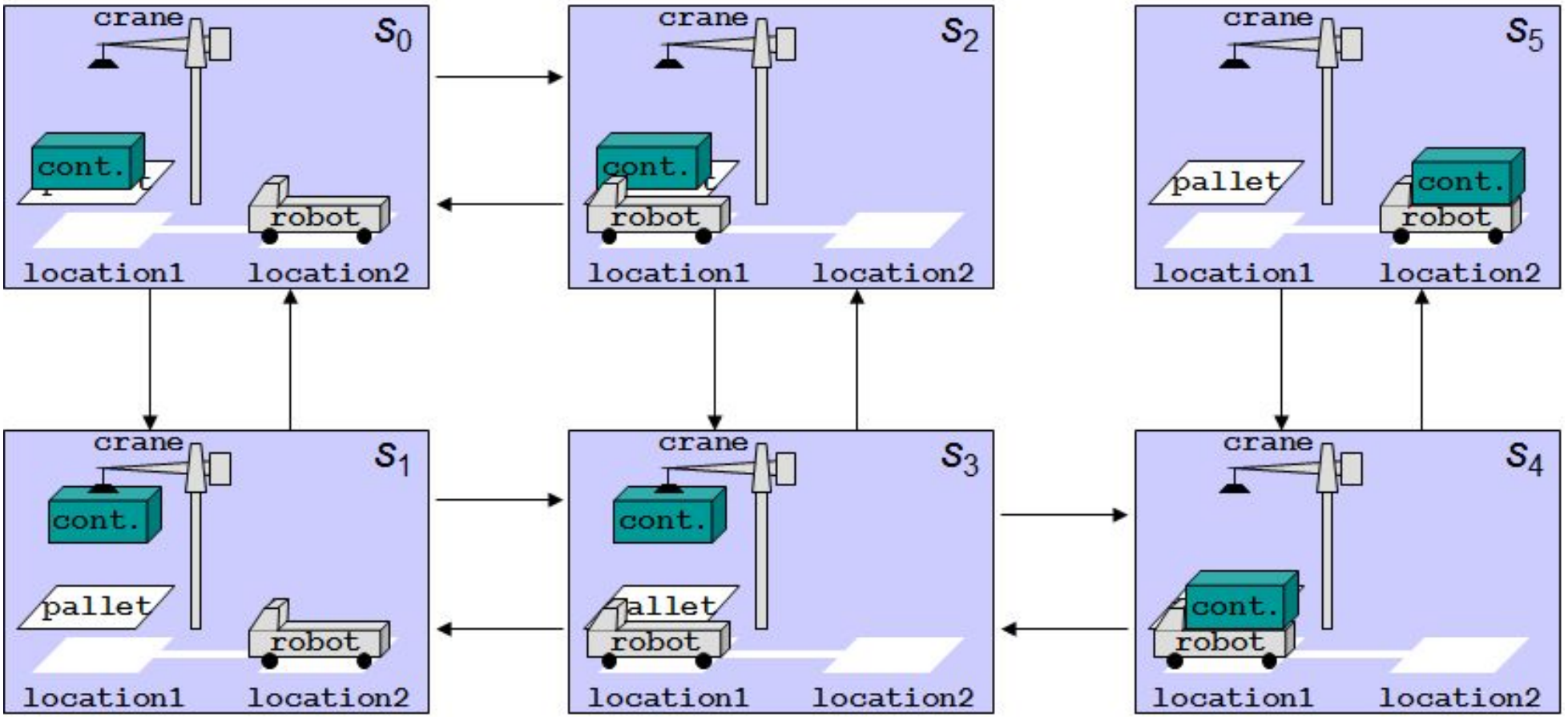
- přepravit 3 kanibaly a 3 misionáře přes řeku
- jakmile je někde více kanibalů než misionářů, jsou misionáři sněděni

Plánování ve stavovém prostoru [Wic11]

Myšlenka:

- aplikace standardních prohledávacích algoritmu
  - do šířky,
  - do hloubky,
  - A^* . . . po dlouhou dobu chyběla vhodná heuristika,
  - atd.
- k řešení plánovací úlohy
  - prohledávaný prostor je podmnožina stavového prostoru
  - uzly odpovídají stavům světa
  - hrany korespondují přechodům mezi stavy
  - cesta v prohledávaném prostoru odpovídá plánu

Plánování ve stavovém prostoru - příklad [Wic11]

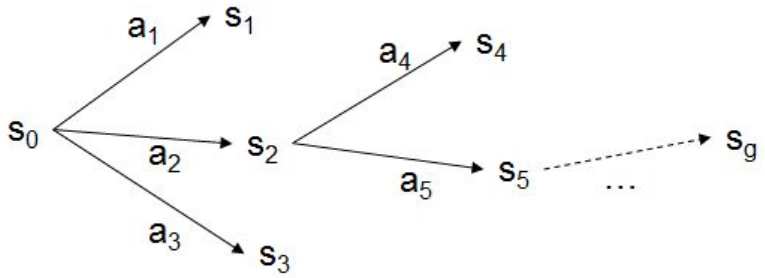


- uzly: zavřené atomy
- hrany: akce (uzavřené instance operátorů)

Plánování ve stavovém prostoru [Wic11]

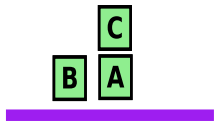
- Dáno: zadání plánovací úlohy $P = (O, s_i, g)$
- Plánování ve stavovém prostoru jako úloha prohledávání:
  - počáteční stav: s_i
  - test na cíl ve stavu s: $s \models g$
  - funkce ocenění cesty plánu π: $|\pi|$
  - funkce následníka stavu s: $\Gamma(s)$

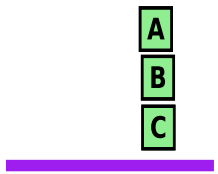
Progresivní hledání [Wic11]



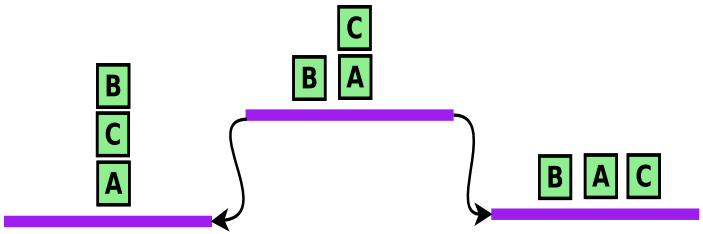
1. Forward-search(O, s_0, g)
   1. $s \leftarrow s_0$
   2. $\pi \leftarrow$ the empty plan
   3. loop
      1. jestliže $s \models g$, potom vrať π
      2. $E \leftarrow \{a | a$ je uzavřená instance operátoru $\in O$ a $precond(a)$ je pravdivá v $s\}$
      3. jestliže $E == \emptyset$, potom vrať $FAILURE$
      4. nedeterministicky vyber akci $a \in E$
      5. $s \leftarrow \gamma(s, a)$
      6. $\pi \leftarrow \pi.a$

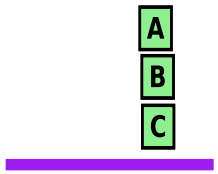
Příklad prohledávání prostoru stavů 1 [Wic11]



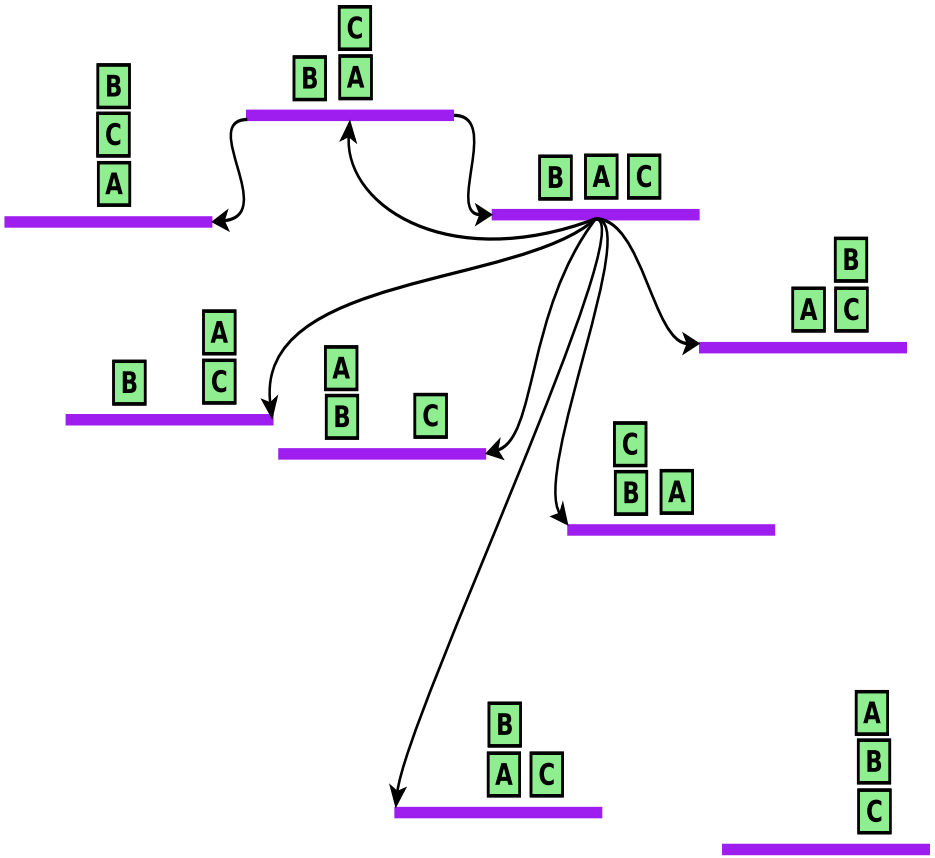


Příklad prohledávání prostoru stavů 2 [Wic11]

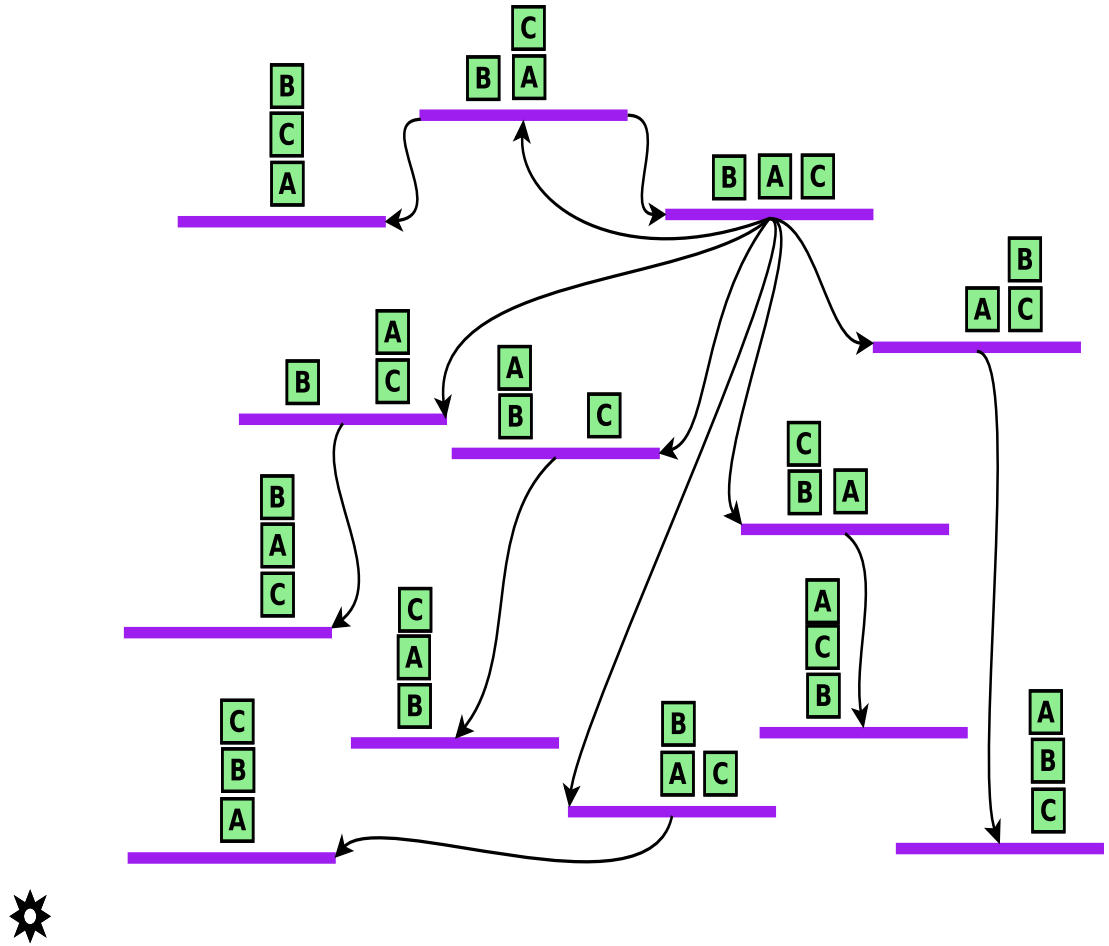




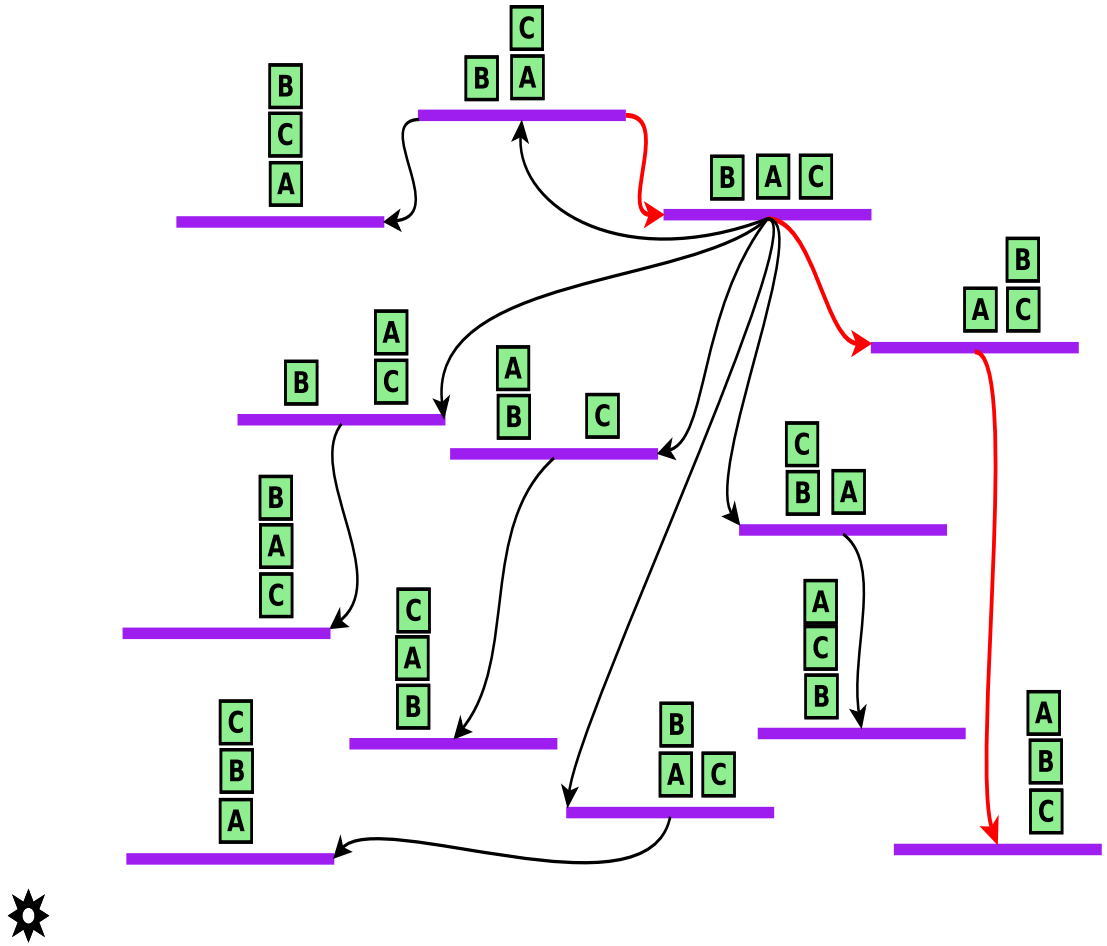
Příklad prohledávání prostoru stavů 3 [Wic11]



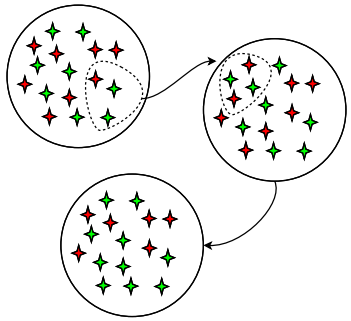
Příklad prohledávání prostoru stavů 4 [Wic11]



Příklad prohledávání prostoru stavů 5 [Wic11]

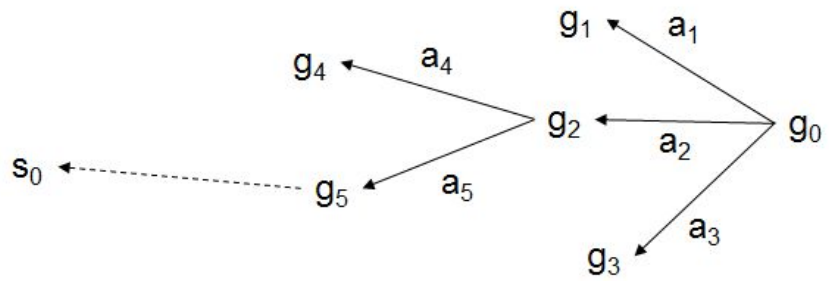


Relevantní akce [Nau09]



- Nechť $\mathcal{P} = (\Sigma, s_i, g)$ je STRIPS plánovací úloha.
- Akce a je **relevantní** pro cíl g, jestliže
  - a způsobí, že alespoň jeden z literálů g je pravdivý
    - $g \cap effects(a) \neq \emptyset$
  - a nezpůsobí, že ani jeden z literálů g je nepravdivý
    - $g^+ \cap effects^-(a) = \emptyset \bigwedge g^- \cap effects^+(a) = \emptyset$
- **Regresní množina** cíle g pro relevatní akci $a \in A$ je:
  - $\gamma^{-1}(g, a) = (g - effects(a)) \cup precond(a)$

Zpětné hledání [Wic11]



1. Backward-search(O, s_0, g)
   1. $\pi \leftarrow$ the empty plan
   2. loop
      1. jestliže $s_0 \models g$, potom vrať π
      2. $A \leftarrow \{a | a$ je uzavřená instance operátoru $\in O$ a $\gamma^{-1}(g, a)$ je definováno $\}$
      3. jestliže $A == \emptyset$, potom vrať $FAILURE$
      4. nedeterministicky vyber akci $a \in A$
      5. $\pi \leftarrow a.\pi$
      6. $g \leftarrow \gamma^{-1}(s, a)$

STRIPS plánovač [Nau09]

1. $\pi \leftarrow$ the empty plan
2. modifikované zpětné prohledávání z g
   1. $precond(a)$ je nová množina podcílů (místo $\gamma^{-1}(g, a)$)
   2. vyber jeden podcíl, který se má dosáhnout
   3. jestliže podcíl není dosažen
      1. vyber akci, která splní podcíl
      2. zajisti dosažení vstupních podmínek akce
      3. proveď akci
   4. zajisti zbytek cílů
3. Algoritmus STRIPS nemusí najít plán, i když existuje.
4. Dosažení podcíle může porušit již splněné cíle.

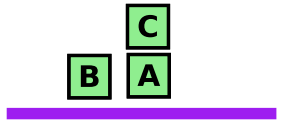
Sussmanova anomálie

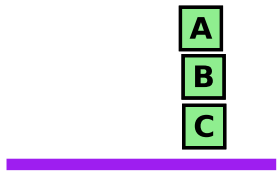
- interakce podcílů, jejichž řešení se musí proložit, aby je bylo možné splnit současně,

Sussmanova anomálie - příklad s kostkami I [Nau09]

START

On(C,A) On(A, Table) Cl(B) On(B,Table) Cl(C)

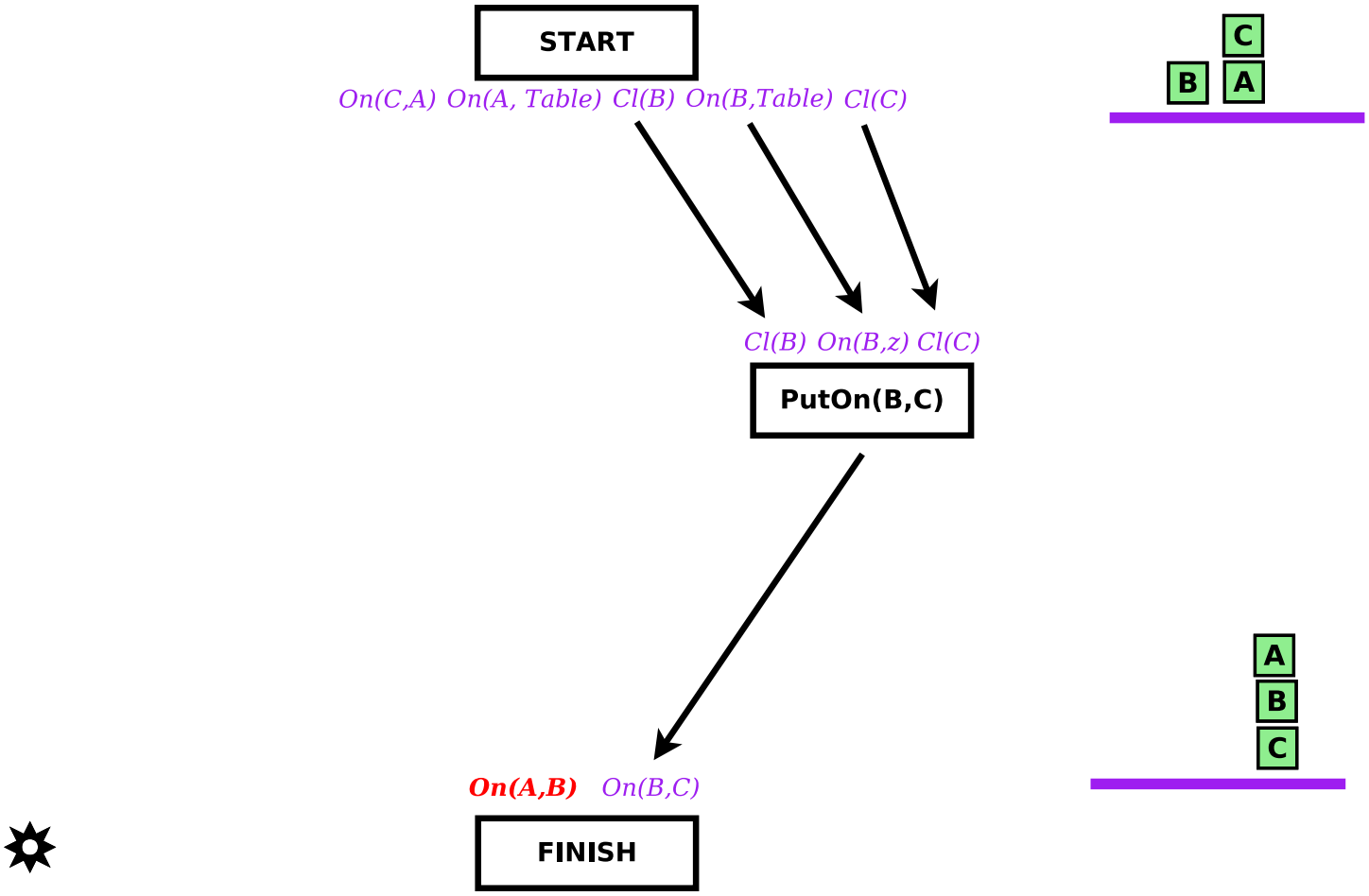




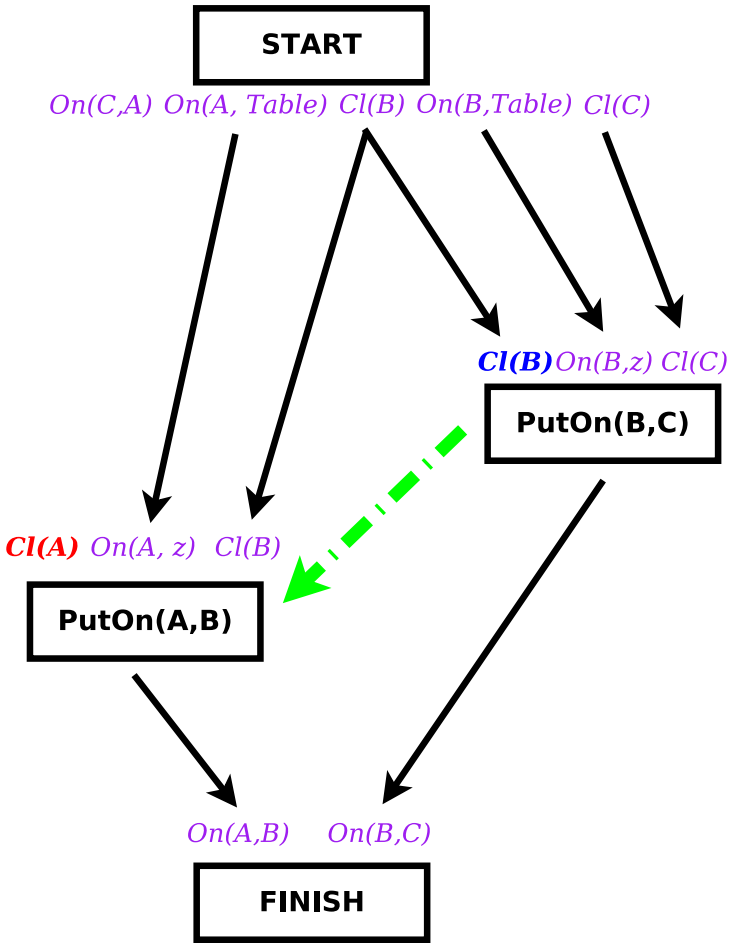
On(A,B) On(B,C)

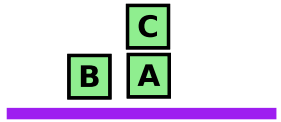
FINISH

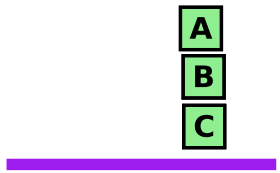
Sussmanova anomálie - příklad s kostkami II [Nau09]



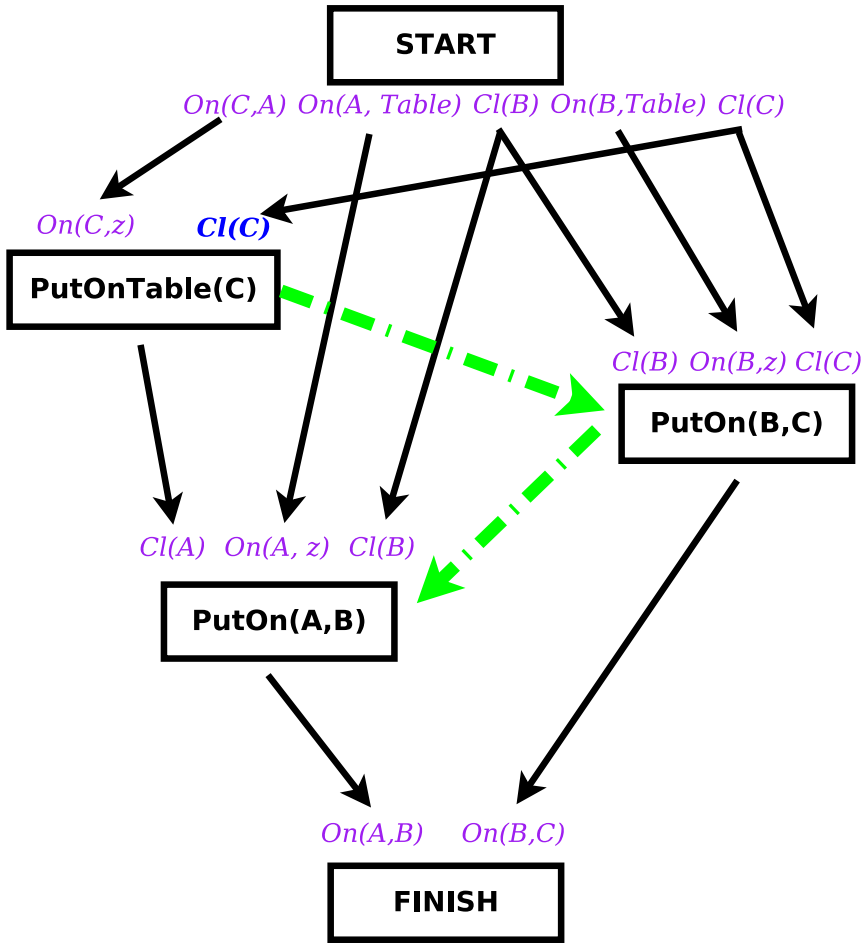
Sussmanova anomálie - příklad s kostkami III [Nau09]

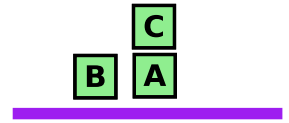


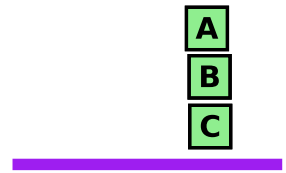




Sussmanova anomálie - příklad s kostkami IV [Nau09]





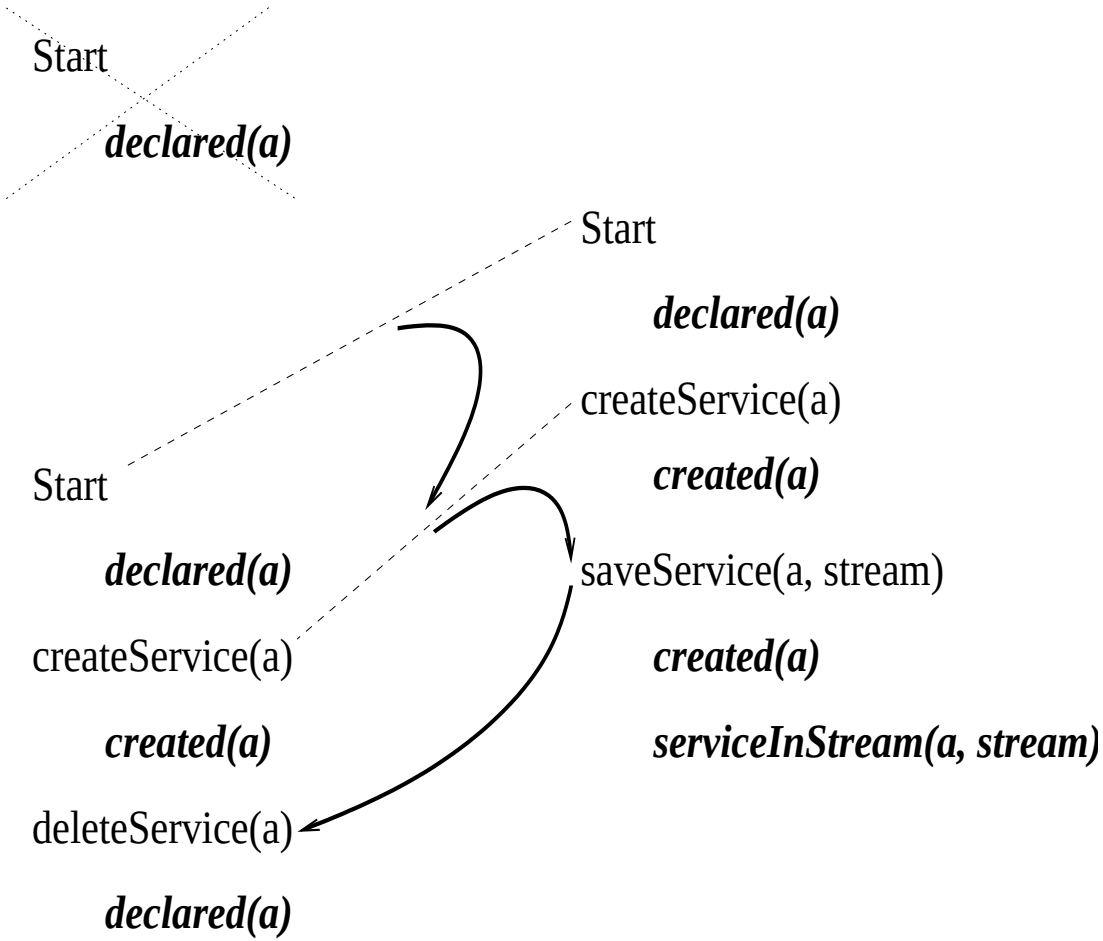


Sussmanova anomálie - Příklad služeb

```
Akce:
loadService(A, S): PRE [declared(A), serviceInStream(A, S)]
saveService(A, S): PRE [created(A)]
                   ADD [serviceInStream(A, S)]
createService(A):  ADD [created(A)]
                   DEL [declared(A)]
deleteService(A):  ADD [declared(A)]
                   DEL [created(A)]
start:             ADD [declared(a)]

Plan:
1.  start
2.* createService(a)
3.* saveService(a, stream)
4.  deleteService(a)
5.  loadService(a, stream)
```

Sussmanova anomálie - schéma



WARPLAN plánovač

- 1973
- k řešení Sussmanovy anomálie se používá strategie dopředných a zpětných posuvů operací,
- lineární plánovač,
  - Plán je representován jako lineární sekvence.
- Jestliže podcíl nelze splnit, plánovač se navrací s úplným prohledáváním prostoru, tj. zkouší všechny možné lineární sekvence.
- do 12-16 volání metod.

Prohledávání prostoru stavů vs. plánů [Wic11]

- prohledávání stavového prostoru
  - prohledávání grafu, jehož uzly reprezentují stavy světa
- prohledávání prostoru plánů
  - prohledávání grafu, jehož uzly reprezentují částečné plány
  - uzly: částečně určené plány
  - hrany: operace zjemnění plánů
  - řešení: částečně uspořádané plány

Částečné plány [Wic11]

- plán
  - množina akcí organizované do nějaké struktury
- částečný plán:
  - podmožina akcí
  - podmnožina organizační struktury
    - časové uspořádání akcí
    - zdůvodnění: co akce znamená pro plán
  - podmnožina vazeb proměnných

Plánování v prostoru plánů - omezující podmínky [Nau09]

- podmínka předcházení
  - a musí předcházet b
- vazební podmínka
  - podmínky nerovnosti, např. $v_1 \neq v_2$ nebo $v_1 \neq c$
  - podmínky rovnosti a substituce, např. $v_1 = v_2$ nebo $v_1 = c$
- kauzální vazby
  - použij akci a k vytvoření podmínky p potřebné pro akci b

Řešení nedostatků - nesplněné cíle [Nau09]

- Otevřený cíl
  - Akce a má podmínku p, která nebyla dosud vyřešena
- Řešení nedostatku
  - nalezni akci b
    - ať ji existující v plánu nebo vložením
  - která může být použita k vytvoření p
    - může předcházet a a produkovat p
  - proveď instanci proměnných
  - vytvoř kauzální vazbu

Řešení nedostatků - hrozby kauzálních vazeb [Nau09]

- Kauzální vazba:
  - vzhledem k vlastnosti relace uspořádání
  - $\forall \alpha_1, \alpha_2 \in \pi : \exists x : x \in \text{pre}(\alpha_2) \wedge x \in \text{eff}(\alpha_1) \Leftrightarrow \alpha_1 \prec \alpha_2$
  - vložíme kauzální vazbu jako relací splnitelnosti mezi operátory
  - $\alpha_1 \xrightarrow{x} \alpha_2$, kde $x \in \text{eff}(\alpha_1) \wedge x \in \text{pre}(\alpha_2) \wedge \alpha_1 \prec \alpha_2$
  - čteme: α_1 poskytuje x pro α_2
- hrozba kauzální vazby
  - **negativní hrozba kauzální vazby**: $\alpha_1 \xrightarrow{q} \alpha_3$ je kauzální vazba v plánu a $\alpha_1 \prec \alpha_2$, $\alpha_2 \prec \alpha_3$, jsou konzistentní s plánem a existuje efect $p \in \text{eff}(\alpha_2)$ takový, že může smazat q
  - **positivní hrozba kauzální vazby**: . . . podobně přidání q
- řešení hrozby
  - degradace (angl. demotion) $\alpha_2 \prec \alpha_1$
  - povýšení (angl. promotion) $\alpha_3 \prec \alpha_2$
  - omezením vazby proměnných

TOPLAN [Pec10]

- angl. Total Order Planning

1. inicializace: $\mathbf{\Pi} \leftarrow \{\{s_{goal}\}\}, \mathbf{S} \leftarrow \{s_{goal}\}$

$\underline{toplan}(s_0, \mathbf{\Pi}, \mathbf{S})$

1. jestliže $\exists s_n \in \mathbf{S}, \pi_n \in \mathbf{\Pi} : s_{goal} == s_n$, pak $return(\pi_n)$
2. jestliže $\mathbf{S} = \emptyset$
   - pak $return(failure)$
   - jinak
     1. $\underline{remove}$ s_i from $\mathbf{S}$ a $\underline{remove}$ π_i from $\mathbf{\Pi}$
     2. $A \leftarrow \{\alpha | eff(\alpha) \in s_i\}$
     3. $S \leftarrow \{s | \forall \alpha \in A : successor(\alpha, s) = s_i\}$
     4. $\Omega \leftarrow \{\pi | \forall \alpha \in A : \pi = \alpha \cup \pi_i\}$
     5. $return(\underline{toplan}(s_0, \underline{append}(\mathbf{\Pi}, \Omega), \underline{append}(\mathbf{S}, S)))$

POPLAN [Pec10]

- angl. Partial Order Planning (simplified)

1. inicializace: $\mathbf{\Pi} \leftarrow \{\text{actions}, \{s_0 \prec s_{goal}\}, \{\}, \{\text{pre}(s_{goal})\}\}$
2. akce α, β

$\underline{poplan}(\mathbf{\Pi})$

1. *if* complete($\mathbf{\Pi}$) then $return(\mathbf{\Pi})$
2. *if* $\exists p \in \text{eff}(\beta) \wedge \beta \in \text{openGoals}(\mathbf{\Pi})$ a $\exists \alpha$, že $p \in \text{eff}(\alpha)$
   - *then* append($\mathbf{\Pi}, \{\{\alpha \xrightarrow{p} \beta\}, \{\alpha \prec \beta\}\}$) a remove($p, \beta, \text{openGoals}(\mathbf{\Pi})$)
   - *else* $return(fail)$
3. *if* existuje kauzální vazba $\alpha_1 \xrightarrow{x} \alpha_2$ ohrožená akcí α_3
   - *then* udělej jeden z následujících kroků
     - Promotion: $return(\underline{poplan}(\mathbf{\Pi} \uplus \{\alpha_3 \prec \alpha_1\}))$
     - Demotion: $return(\underline{poplan}(\mathbf{\Pi} \uplus \{\alpha_2 \prec \alpha_3\}))$
   - *else* $return(\underline{poplan}(\mathbf{\Pi}))$

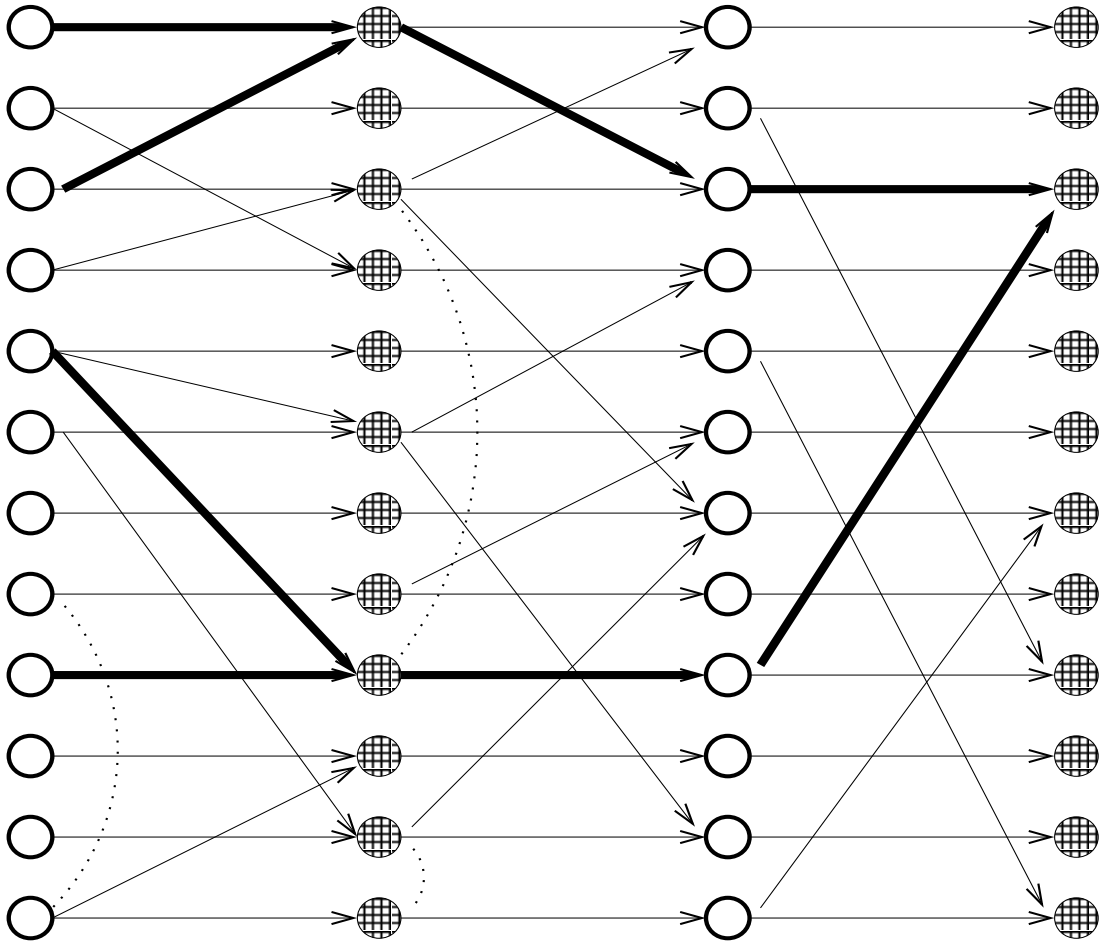
TWEAK, PWEAK plánovače

- 1987
- plánovač s částečným uspořádáním:
  - Plány jsou reprezentovány jako částečné uspořádané sekvence.
  - Závislosti mezi akcemi se zaznamenávají explicitně.
  - Konečný plán se získá linearizací částečného plánu.
- Jestliže dojde k selhání podcíle, prohledávají se pouze hraniční podmínky.
- Je rychlý.
  - sekvence s 20 metodami v několika sekundách,
  - Bohužel, existují situace, kdy se plánovač chytí do nekonečné smyčky:
    - smyčky při vytváření a rušení objektů.

GRAPHPLAN plánovač

- 1997
- plány jsou reprezentovány pomocí *plánovacího grafu*,
  - myšlenka je velmi podobná dynamickému programování či řešení toku sítí,
- Všechny plány jsou konstruovány souběžně.
  - rozšiřování grafu (dopředný běh)
  - vyhledání plánu (zpětný běh)
- Plánovač udržuje relaci binární vzájemné výlučnosti (*mutex*) mezi uzly representující aplikované akce a výroky popisující stav.
- Problém s cyklením odstraněn.
- Nelze používat parametrizované akční schémata (instance).
  - Vytváří obrovský prostor výroků.
- Existuje řada podpůrných strategií, které podstatně urychlují plánovaní.
- Implementace jsou schopny zvládnout plány s 50-100 voláními metod do minut.

GraphPlan - plánovací graf



Plánovací grafy [Nau09]

Plánovací graf

Střídání dvou úrovní

- S_i obsahuje všechny literály, které bý mohly platit v čase i
- A_i obsahuje všechny akce, které mohou mít splněny předpoklady v čase i

Mutex

- mezi dvěma akcemi
  - **nekonzistentní efekty** . . . jedna akce neguje efekt druhé akce,
  - **interference** . . . efekt jedné akce je negací podmínky druhé akce
  - **konkurenční potřeby** . . . podmínka jedné akce je neslučitelná s podmínkou druhé akce
- mezi dvěma literály
  - **nekonzistentní podpora** . . . dva literály jsou vzájemně svou negací nebo každý pár akcí produkucí tyto dva literály je vzájemně neslučitelný.

STRIPS reprezentace - příklad

- pocatecni stav: *start*
- *cil: writeValue(a, 3)*
- *akce:*

```
startPlan:      PAR [] 
                PRE [start]
                ADD [declared(server, a), declared(server, b)]
                DEL [start]
connectServer:  PAR [A]
                PRE [declared(server, A)]
                ADD [bound(server,A)]
                DEL [declared(server, A)]
writing:        PAR [a, 3]
                PRE [bound(server,a)]
                ADD [writeValue(a, 3)]
                DEL []
```

Výsledek příkladu

- cíl: *writeValue(a,3);*
- *výsledná sekvence:*
  1. *start*
  2. *connectServer(a)*
  3. *writing(a, 3)*

Příklad - Specifikace API

```
pweakDefine(action(saTag::loadTags),
  domain(saTag),
  comment('\%(TagsId)s are loaded from the stream
           \%(StreamId)s into \%(TagGroupId)s'),
  body(
    parameters::(ComId #com; StreamId #stream;
      TagGroupId #tagGroup; TagsId # tags;
      CounterId #counter),
    constraints::[],
    precondition::[
      (object(com, open, ComId), assumed),
      (object(stream, filled, StreamId, tags, CounterId),
         validated),
      (stream(StreamId, start), validated),
      (object(tagGroup, open, TagGroupId), assumed),
      (object(tags, tagDeclaration, TagsId), assumed),
      (size(tags, TagsId, CounterId), assumed)],
    positive_effect::[
      operation(loadTags),
      object(tags, open, TagsId),
      activeFlags(tags, TagsId, satActiveNone),
      related(tags, TagsId, tagGroup, TagGroupId)],
    negative_effect::[
      object(tags, tagDeclaration, TagsId)])).
```

Příklad - abstraktní testovací skript

```
TESTCASEMARK
TESTCASE: [86, simpleMethodUsage]
Tested:   [step(loadTags(comA, streamA, tagGroupB, tagsA, counterA),
 %(TagsId)s are loaded from the stream %(StreamId)s into %(TagGroupI
 begin(start)
 step(declare(counter, counterA), An object must be declared)
 step(declare(server, serverA), An object must be declared)
 step(initilizeCom(comA), COM environment is initialized)
 step(declare(tagGroup, tagGroupA), An object must be declared)
 step(declareTagNames(tagNamesA, counterA), TagNames must be declare
 step(createServer(comA, serverA), SATag %(ServerId)s is created)
 step(createTagNames(tagNamesA), Names reference %(TagNamesId)s is e
 step(declareTags(tagsA, counterA), An object must be declared)
 step(createTagGroup(comA, tagGroupA, serverA), SATag %(TagGroupId)s
 step(declare(stream, streamA), An object must be declared)
 step(releaseServer(comA, serverA), SATag %(ServerId)s is released)
 step(addTags(comA, tagGroupA, tagNamesA, tagsA, counterA), Tags %(T
 step(setActivationFlags(tagGroupA, satActiveBound, tagsA), Activati
 step(createStream(streamA), A stream %(StreamId)s in memory is crea
 step(declare(tagGroup, tagGroupB), An object must be declared)
 step(createServer(comA, serverA), SATag %(ServerId)s is created)
 step(saveStream(streamA, tagGroupA, tagsA, counterA, satActiveBound
 step(rewindStream(streamA), %(StreamId)s is rewinded to its begin)
 step(createTagGroup(comA, tagGroupB, serverA), SATag %(TagGroupId)s
 step(releaseTags(comA, tagsA, counterA), Tags %(TagsId)s are releas
 step(loadTags(comA, streamA, tagGroupB, tagsA, counterA), %(TagsId)
 aTEST(loadTags(comA, streamA, tagGroupB, tagsA, counterA), %(TagsId
 step(releaseTagNames(tagNamesA), Reference %(TagNamesId)s is releas
 step(releaseFilledStream(streamA, start, counterA), %(StreamId)s is
 .....
 step(uninitilizeCom(comA), COM environment is closed)
 end
```

Příklad - výsledný C++ kód

```
// ============================================
// Test case: 441
// Created:   Wed Nov 15 21:19:35 2000
// Author:    Generated by TCG written by Radek Marik,
// Description:
//     TagsId are loaded from the stream StreamId into TagGroupId.

void Test_441_0()

  HRESULT hr = S_OK;
  ISATagServer *serverA;
  ISATagGroup *tagGroupA;
  // SATag 'serverA' is created.
  hr = CoCreateInstance(CLSID_SATagServer, NULL, CLSCTX_ALL,
    IID_ISATagServer, reinterpret_cast<void**>(&serverA));
  TG_CheckHResult(hr, NULL, __uuidof(0), __LINE__ - TestRefStart_441,
  ISATagRef * tagsA[counterA];
  ........
  // 'tagsA' are loaded from the stream 'streamA' into 'tagGroupA'.
  hr = tagGroupA->Load(streamA, counterA, tagsA);
  TG_CheckHResult(hr, tagGroupA, __uuidof(tagGroupA),__LINE__ - TestR
                  _T("tagGroupA->Load"));
  // TEST: 'tagsA' are loaded from the stream 'streamA' into 'tagGrou
  TG_PrintResult(_T("'tagsA' are loaded from the stream 'streamA' int
  streamA->Release();
  // Release tags tagsA
   for( long i = 0; i < counterA; i++) tagsA[i]->Release();
  tagGroupA->Release();
  // SATag 'serverA' is released.
  serverA->Release();
```

Hierarchické plánování [SV10]

angl. Hierarchical Task Net (HTN) Planning

Základní myšlenka

- složité plány mají často zřejmou strukturu,
- struktura se dá zachytit ve formě hierarchie abstraktních plánů.
- podplány jsou často (skoro) na sobě nezávislé.

Příklad

- "Abychom se dostali na konferenci v ?x, dojeď na letiště, nastup do letadla do ?x, potom jeď do hotelu konference."
- "Abychom se dostali na letiště, buď řiď auto nebo si vezmi taxi."
- "Jestliže máš dostatek peněz a potřebuješ si vzít taxi do ?y, buď zavolej dopředu nebo zamávej rukou, potom nasedni do taxíku, řekni "Chci je to ?y", počkej dokud nedojedete do ?y, zaplať jízdné a poté vystup z taxíku."

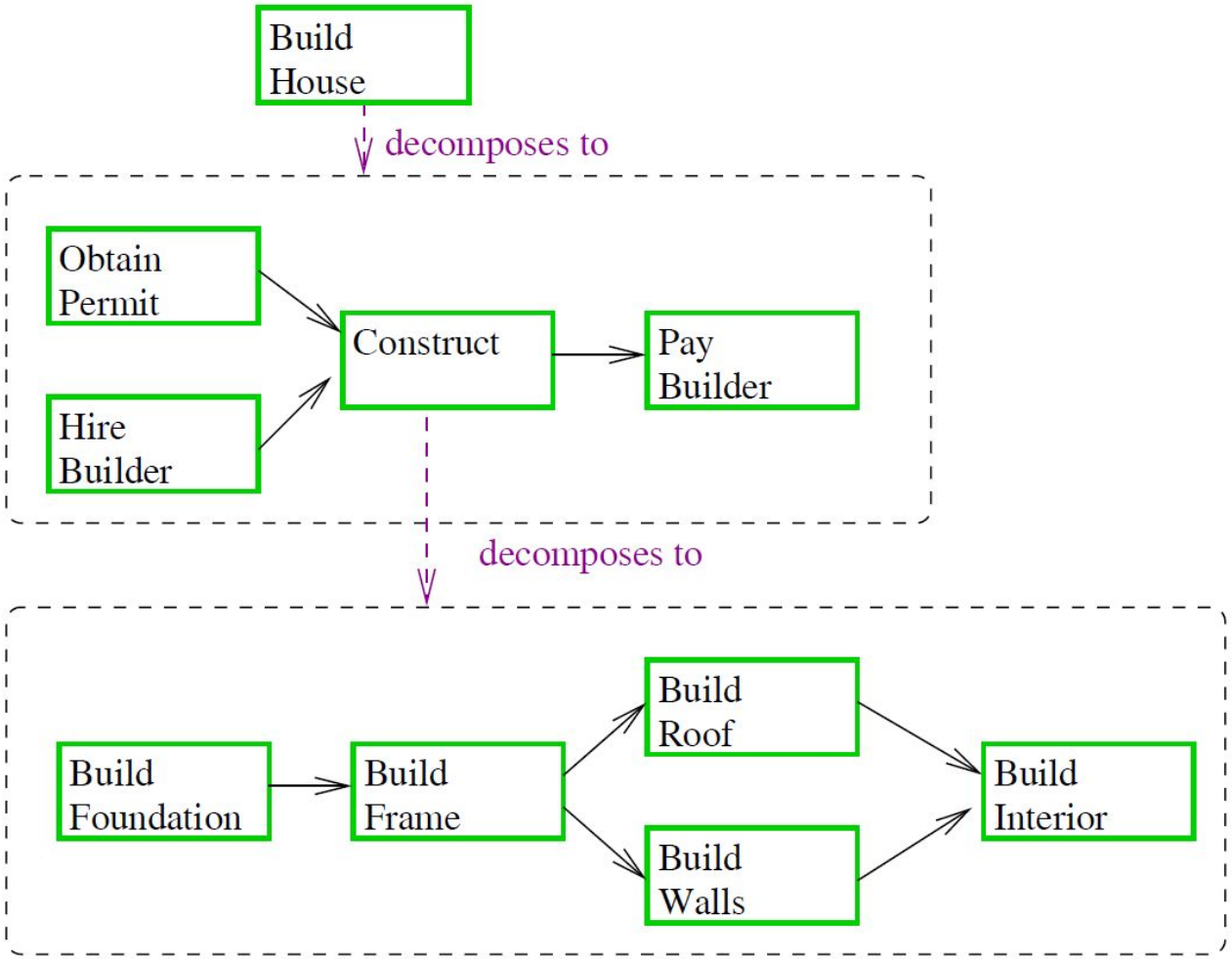
HTN plány [SV10]

- Plán = (Úlohy, Omezení)

Úlohy

- primitivní
  - standard forma STRIPS operátorů
  - "proveditelné"
- složené
  - vstupní podmínky a efekty
  - metody na kompozici operátory do podplánů
    - s detaily vnitřní struktury,
    - parametrizované,
    - podobné makrům nebo podprogramům.

HTN plány: příklad



HTN omezení a plány [SV10]

Omezení

- precedence (před, po, mezi)
- časové metriky
- zdroje
- omezení na dané úrovni platí pro všechny páry úloh nižší úrovně
  - $U = \{u_1, u_2\}$; $V = \{v_1, v_2\}$
  - $U < V \Rightarrow u_1 < v_1; u_2 < v_1; u_1 < v_2; u_2 < v_2$

Plány

- **Abstraktní plán** obsahuje složené operátory/úlohy
- **Instance plánu** obsahuje pouze primitivní úlohy
- **Plná instance plánu** je úplně uspořádaná instance plánu se všemi proměnnými vázanými.

Vyhledávací prostor [SV10]

Zadání problému

- *Cílový stav* je abstraktní úloha, které má být dosaženo
- **není** to stav světa

Operátory prohledávání prostoru

- dekomponují úlohu na podúlohy
- dosazují parametry úloh
- řeší konflikty.

Abstraktní HTN algoritmus [SV10]

HTN-plan(tasks, constraints, methods)

1. Jestliže (tasks, constraints) nemá řešení, return($\{\}$)
2. Jestliže (tasks, constraints) je instance plánu
   - **vyber** plnou instanci plánu
   - jestliže takový plán existuje, vrať jej, jinak return($\{\}$)
3. **Vyber** abstraktní úlohu $t \in$ tasks
4. **Vyber** aplikovatelnou metodu $m \in$ methods
   - kde u je seznam úloh z m a c jsou omezení z m
   - tasks $=$ (tasks $- \{t\}) \cup u$
   - constraints $=$ constraints $\cup\, c$
5. (tasks, constraints) $=$ applyCritics(tasks, constraints)
6. return(HTN-plan(tasks, constraints, methods)

Proč je tak dobré [SV10]

Specifikace

- Metody kódují doménovou znalost.
- Metody kódují znalost řešení problému
  - "jak"nežli "co"
- Abstrakce zapouzdřují vzory interakcí
- + Může být jednoduší specifikovat doménu
- − Musí se specifikovat všechny možné cíle (a jak jich lze dosáhnout)

Varování

- HTN plánování, v nejhorším případě, zůstavává NP-úplné
- Nemusí se ukončit (rekurzivní expanze metody - může být obtížné detekovat smyčky)
- Může nastat, že musí úplně expandovat, než se najde, že plán neexistuje.

Simple Hierarchical Order Planner (SHOP) [SV10]

Algoritmus

- lineární plánovač s dopředným vyhledáváním
  - plánuje v tom samém pořadí jako probíhá vykonání plánu
  - v základu se jedná o prohledávání do hloubky
- primitivní úlohy nemají podmínky
- nepovoluje souběžné akce
- representace operátorů s vysokou expresivitou (numerické výpočty)
- výkonný (ale poněkud neflexibilní) plánovací algoritmus

Příklad SHOP domény [SV10]

(:operator (!putdown ?block)
((holding ?block)) ← Delete list
((ontable ?block) (handempty))) ← Add list

(:method (make-clear ?y)
((clear ?y)) ← Applicability condition
nil) ← Task list

(:method (make-clear ?y)
((on ?x ?y)) ← Applicability condition
((make-clear ?x)
(!unstack ?x ?y) (!putdown ?x)) ← Task list

Rozšíření jednoduchého HTN plánování [SV10]

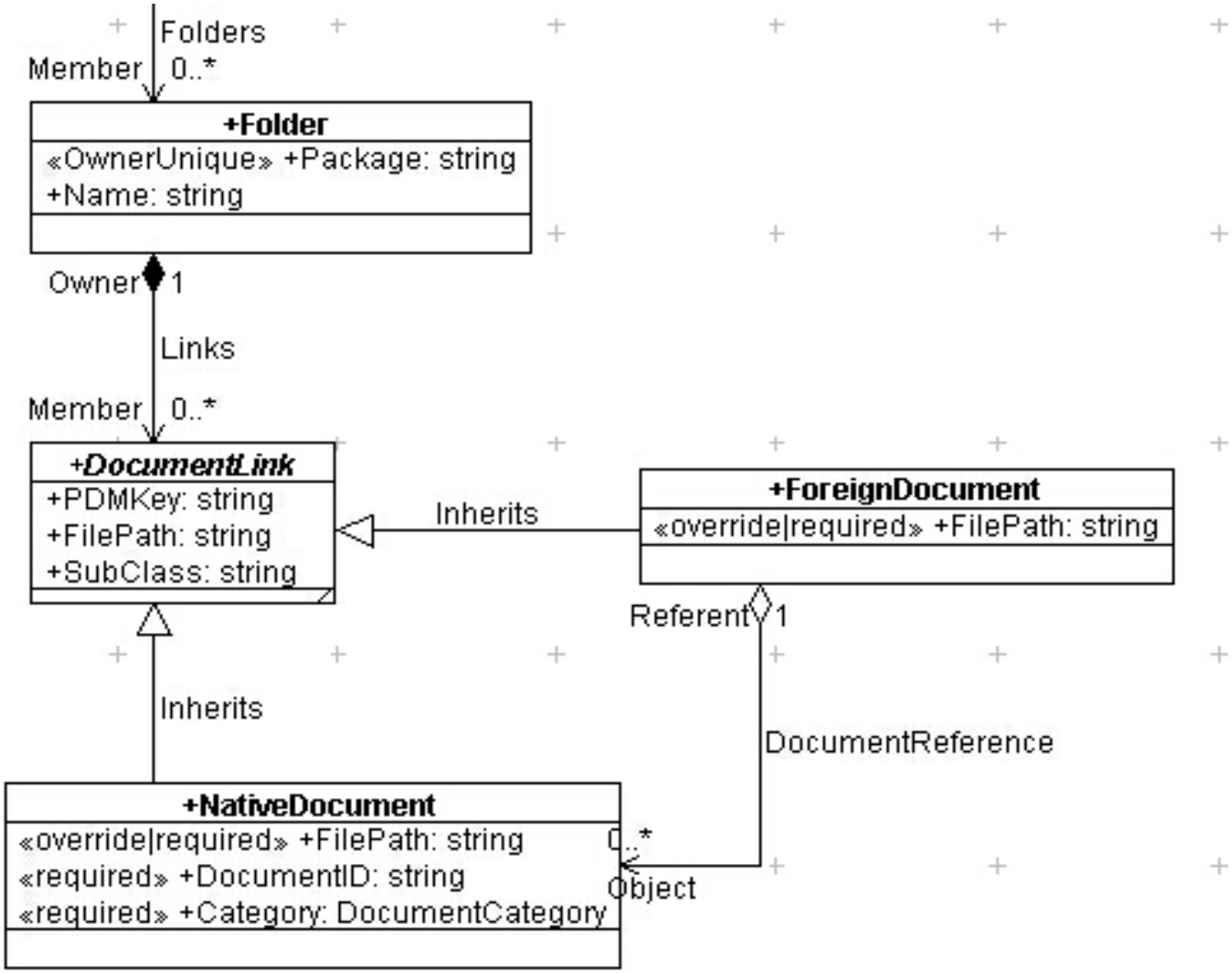
1. detekce hrozeb
   - řešení pro cíle s interakcí
   - nalezení způsobu násobného využití operátorů
2. metody mohou vkládat podmínky a efekty
   - umožňuje nalézt dříve hrozby během plánování
3. metody mohou indikovat využití zdrojů
   - je možné zahrnout některé typy rozvrhnování
4. operátory/metody mohou zahrnout otevřené podmínky ("externí podmínky")
   - je potřeba používat plánování založené na zjemňování akcí
   - umožňuje hledání "inovátorských"řešení

Příklad: stavba domu (O-Plan) [SV10]

Method Build (?house)

Precondition:	(and (own land) (have money))
Effects:	(built ?house)
Applicability:	(single-family-home ?house)
Expansion:	S1: Build-Foundation(?house)
	S2: Build-Frame(?house)
	S3: Build-Roof(?house)
	S4: Build-Walls(?house)
	S5: Build-Interior(?house)
	S6: Decorate(?house)
Orderings:	S1<S2, S2<S3, S2<S4, S3<S5, S5<S6
Links:	S1 causes (foundations laid) for S2
	S2 causes (frame erected) for S3 and S4
	S3 causes (roof built) for S5
	S4 causes (walls built) for S5
	S5 causes (interior done) for S6
TimeWindow:	start between 11:30 and 14:30 at S3
Resources:	bricklayers = between 1 and 2 men at S4

Příklad HTN: OO test design
1. UML model



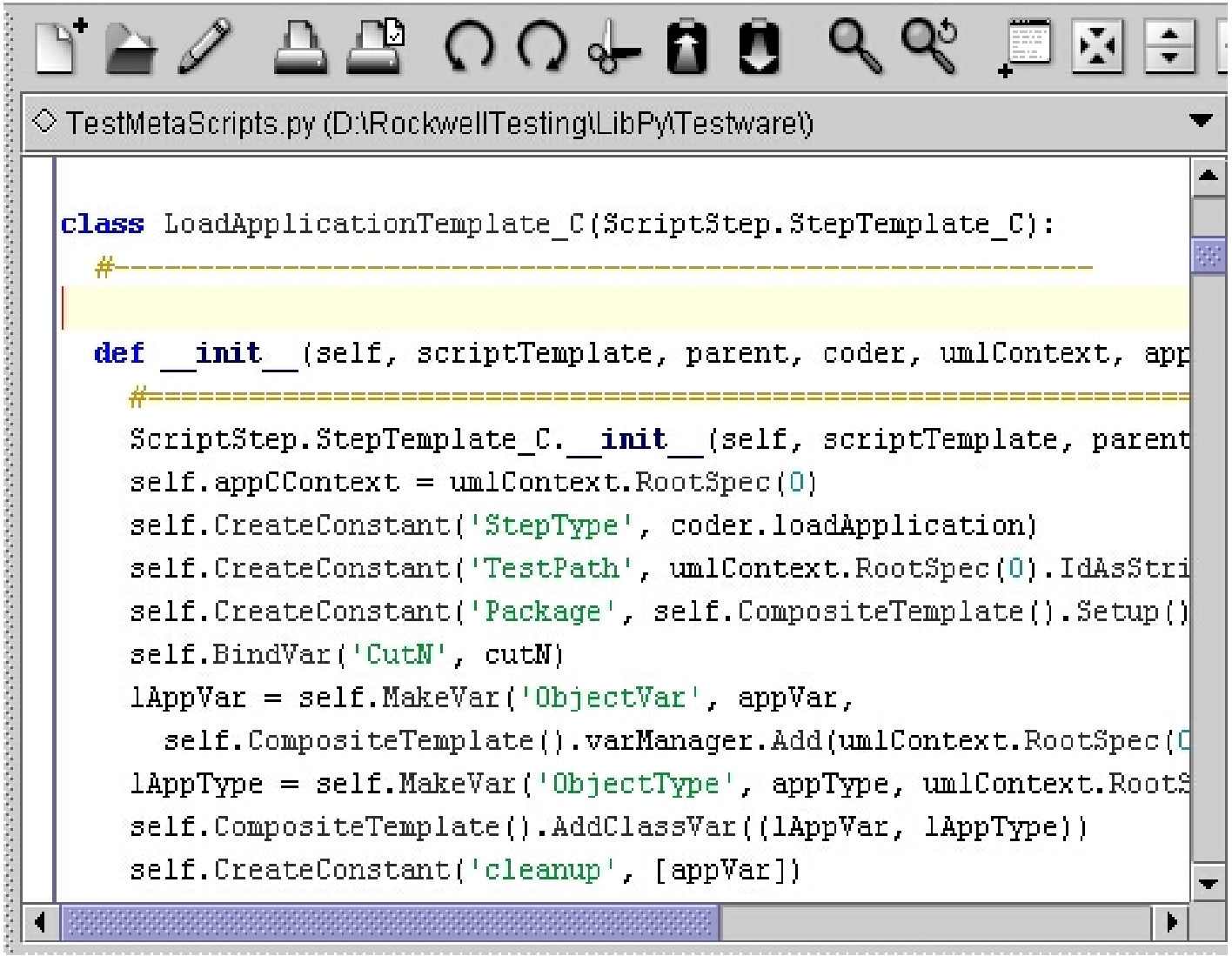
Příklad HTN: OO test design
2. Datapool with generated instances

2:Data in Table 'DBOracle.ObjectTable' in 'ECLogic_CE0_1-SULAFAT' on '...

	_CContextID	_Creation	_ClassID	_ID	_Build
	('IOTag', 0, 0)	Generated	IOTag	11295	25
	('AssemblyUDDTMember', 2, 4)	Generated	AssemblyUDDTMember	11368	67
	('AssemblyUDDTMember', 2, 0)	Generated	AssemblyUDDTMember	11458	79
	('SegmentActivity', 1, 0)	Generated	SegmentActivity	12092	283
	('PCMTEntry', 0, 0)	Generated	PCMTEntry	12425	28
	('CellCommand', 4, 0)	Generated	CellCommand	12429	72
	('AssemblyUDDTMember', 3, 6)	Generated	AssemblyUDDTMember	12991	144
	('Cell', 0, 0)	Generated	Cell	13559	108
	('Station', 1, 0)	Generated	Station	14124	35
	('SegmentActivityTemplate', 0, 0)	Generated	SegmentActivityTemplate	14732	234
	('CellCommand', 11, 0)	Generated	CellCommand	1480	259
	('AssemblyUDDTMember', 2, 1)	Generated	AssemblyUDDTMember	14854	59
	('Diagnostic', 4, 0)	Generated	Diagnostic	14933	163
	('CellCommand', 2, 0)	Generated	CellCommand	15254	180
	('SequenceType', 0, 0)	Generated	SequenceType	1527	6
	('UDDTMemberValueDint', 3, 0)	Generated	UDDTMemberValueDint	16381	114
	('CellCommand', 6, 0)	Generated	CellCommand	16998	299
	('UDDTMemberValueReal', 0, 0)	Generated	UDDTMemberValueReal	17	134
	('AssemblyUDDTMember', 1, 4)	Generated	AssemblyUDDTMember	17211	222
	('AssemblyUDDTMember', 0, 1)	Generated	AssemblyUDDTMember	17294	98
	('AssemblyUDDTMember', 1, 3)	Generated	AssemblyUDDTMember	17375	190
	('SegmentActivity', 0, 0)	Generated	SegmentActivity	17830	261
	('ControlSequence', 1, 0)	Generated	ControlSequence	18078	75
	('Diagnostic', 9, 0)	Generated	Diagnostic	18128	33
	('Cell', 5, 0)	Generated	Cell	18343	103
	('CellCommand', 4, 0)	Generated	CellCommand	18461	61

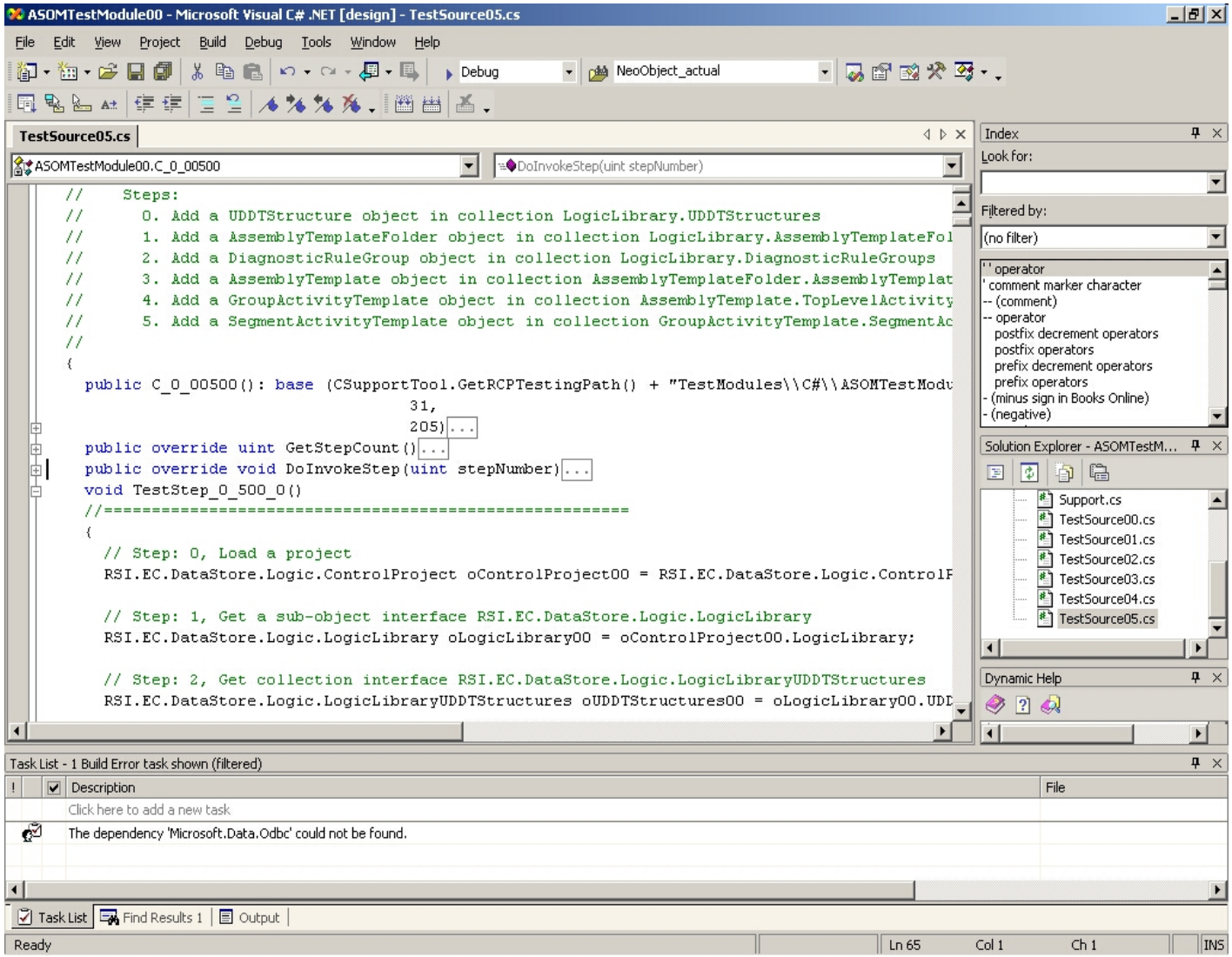
Příklad HTN: OO test design

3. Metascripts



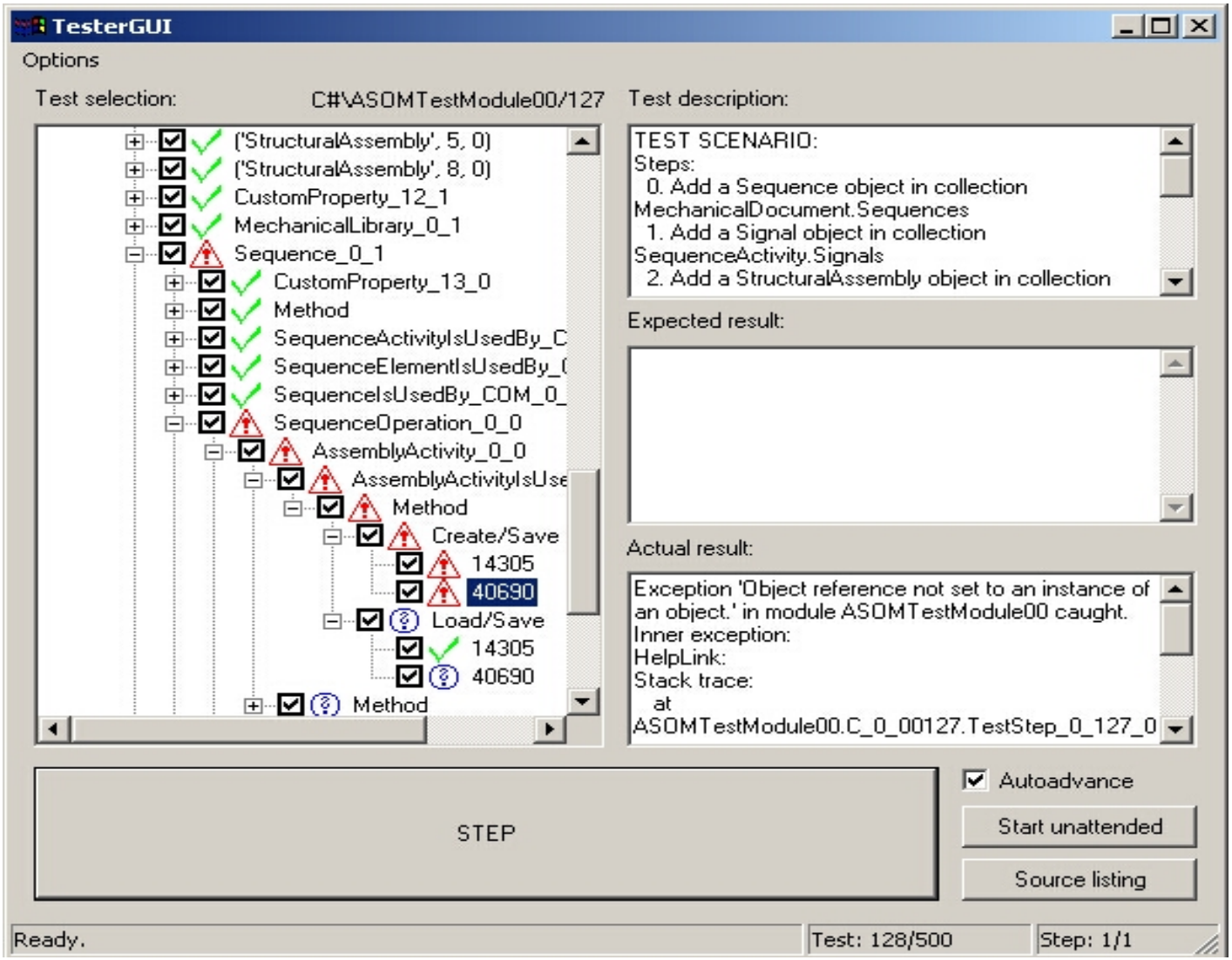
Příklad HTN: OO test design

4. Generované testovací scripty



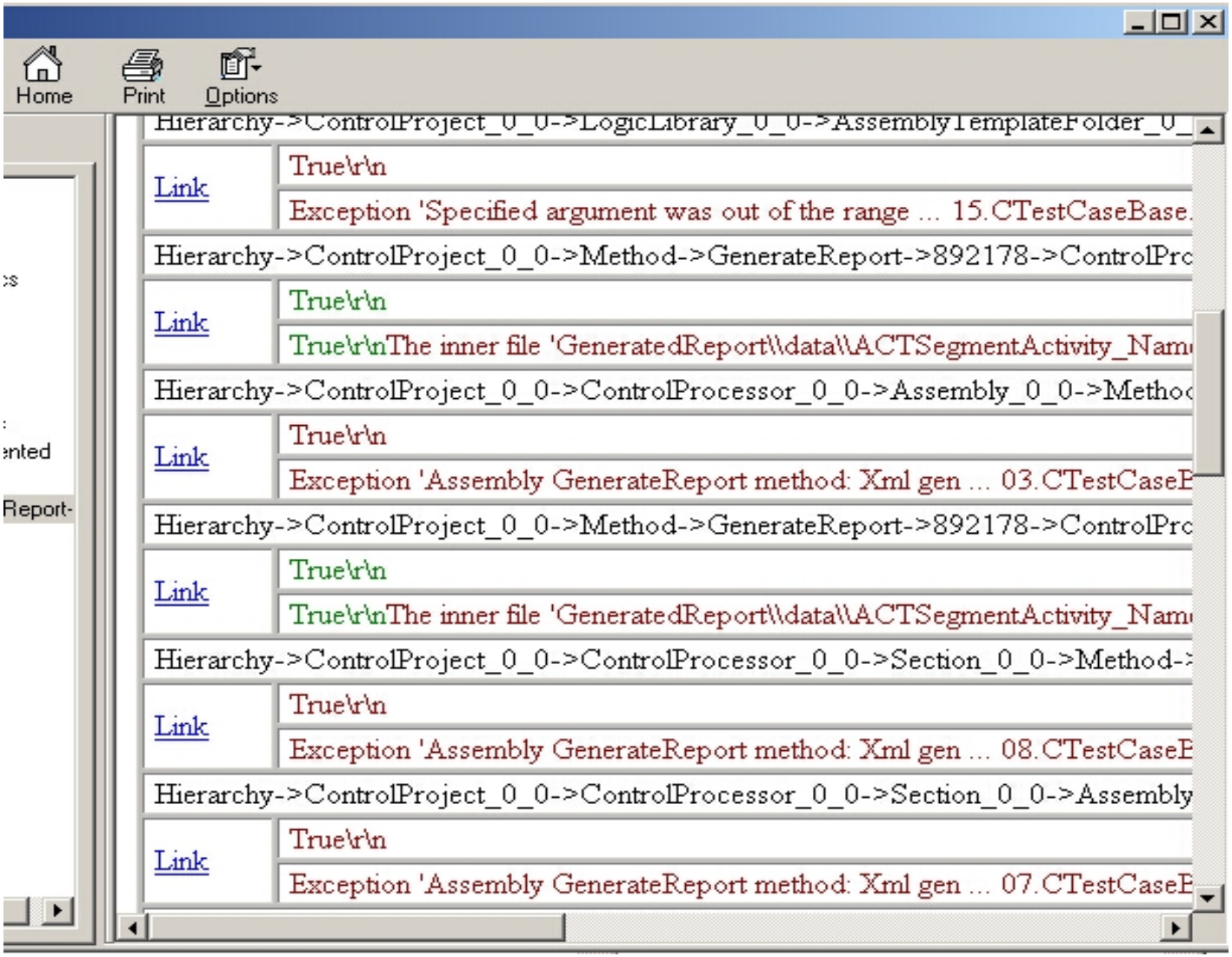
Příklad HTN: OO test design

5. Testovací sada



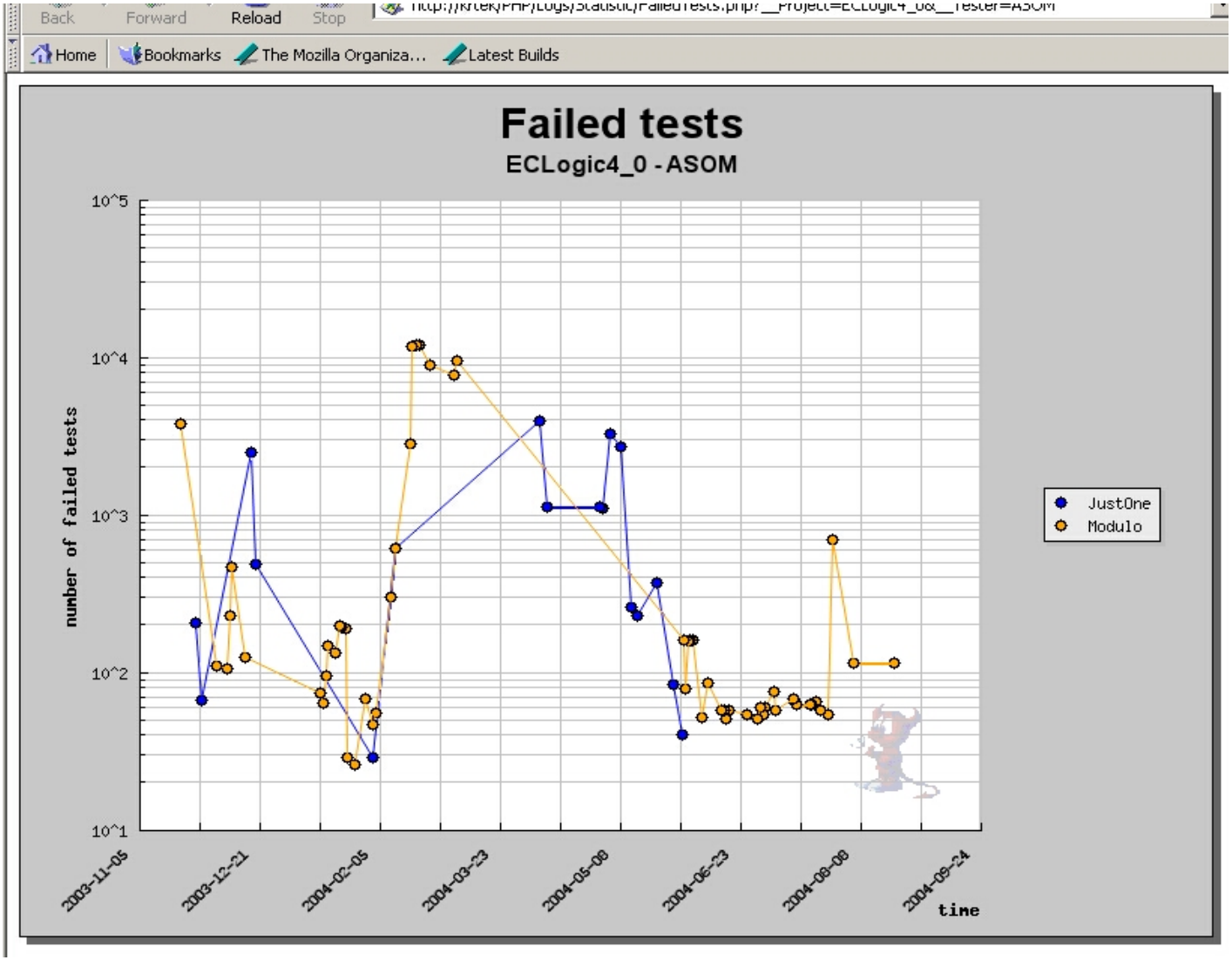
Příklad HTN: OO test design

6. Záznam testů



Příklad HTN: OO test design

7. Průběh testování



3 Příloha

- JSHOP2

JSHOP2 [?]

Atomická úloha

- ([: immediate] s t_1 t_2 ... t_n)
-

Seznam úloh

- ([: unordered] [tasklist$_1$ tasklist$_2$ ... tasklist$_n$])
-

JSHOP2 [?]

Operátory

- (: operator h P D A $[c]$)
- (: $protection$ a)

Metody

- (: method h [name$_1$] L_1 T_1 [name$_2$] L_2 T_2 $\ldots$ [name$_n$] L_n T_n)
-

JSHOP2 [?]

Plánovací doména

- (defdomain domain-name$(d_1\ d_2\ \ldots\ d_n)$)

Plánovací úloha

- (defproblem problem-name domain-name
  $([a_{1,1}\ a_{1,2}\ \ldots\ a_{1,n}])\ T_1\ \ldots\ ([a_{m,1}\ a_{m,2}\ \ldots\ a_{m,n}])\ T_m$)
-

Literatura I

Dana Nau.

CMSC 722, ai planning (fall 2009), lecture notes.
http://www.cs.umd.edu/class/fall2009/cmsc722/, 2009.

Michal Pechoucek.

A4m33pah, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a4m33pah/prednasky, February 2010.

Reid Simmons and Manuela Veloso.

Planning, execution, and learning, lecture notes.
http://www.cs.cmu.edu/ mmv/planning/, September 2010.

Gerhard Wickler.

A4m33pah, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a4m33pah/prednasky, February 2011.