Návrh a verifikace řídícího systému pro nádraží

Peter Boráros

***Abstrakt*—Tento dokument popisuje řešení semestrální úlohy pro kurz a4m33au - automatické uvažování. Cílem je návrh a verifikace řídícího systému pro vlakové nádraží, s použitím automatického dokazování v logice prvního řádu.**

I. Úvod

Táto práce představuje návrh a verifikaci řídícího systému pro vlakové nádraží, s použitím automatického dokazování v logice prvního řádu.

V sekci I je popsané podrobné zadání problému (je převzaté z webových stránek kurzu). V sekci II přecházi přes jednotlivé aspekty problému, formalizuje je a přináší obecné řešení. Sekce III popisuje implementaci a způsob použití navrženého systému. V sekci IV je rozbor experimentů, nad jednoduchými instancemi problému (vid obr. 1, 2 a 3), a závěr.

A. Specifikace problému

Nádraží je souvislý orientovaný graf. Uzly, z kterých nevedou šipky, nazveme výjezdy, uzly, do kterých nevedou vedou šipky, nazveme vjezdy. Omezujeme se na grafy, u kterých z každého vjezdu existuje cesta do každého výjezdu. Každý uzel a každá hrana mají zadaný unikátní název (začínající malým písmenem).

1) Vlastnosti nádraží:

- Každý uzel s více než jednou výstupní hranou je zároveň výhybka.
- Časově variabilní prvky v nádraží jsou:
  - pohybující se vlaky,
  - na vstupních uzlech řízená návěstidla a
  - výhybky.
- V daném časovém okamžiku je každý vlak právě v jednom uzlu.
- Vlaky se pohybují pouze ve směru orientace hran grafu (tj. nemohou couvat).
- Je-li v uzlu vlak, platí, že vlak někdy do uzlu přijel (nebyl tam od nepaměti) a jednou odjede (ale není určeno, kdy - je to na „rozhodnutí strojvedoucího“).
- Na vjezdových uzlech (a pouze tam) jsou návěstidla. Ta blokují odjezd vlaků ze vstupních uzlů: Je-li návěstidlo zavřené, vlak zůstává na vstupním uzlu; je-li otevřené, může (ale nemusí) vyjet. Vlak (strojvedoucí) vždy tato návěstidla respektuje. Každý vjezdový uzel má právě jednu výstupní hranu, není tedy nikdy výhybkou.
- Každý vlak má dán výstupní uzel, do kterého chce dojet. Tento cíl se nemění celou dobu, co vlak projíždí nádražím.
- Nádraží podle tohoto cíle směruje vlak pomocí přepínání výhybek. Řídící systém nádraží může libovolně nastavovat stav výhybek.
- Pokud je vjezdový uzel prázdný, může se v něm kdykoliv objevit nový přijíždějící vlak (i hned po odjezdu předcházejícího).

2) Kritické stavy v nádraží: V nádraží rozlišujeme tyto kritické stavy:

- Vlak stojí v uzlu (který je zároveň výhybkou, viz výše), a dojde k přepnutí výhybky.
- Dva nebo více vlaků přijede do stejného uzlu.
- Vstupní návěstidlo zůstane trvale uzavřené.

3) Vstup: Graf nádraží bude zadáván v následujícím formátu (podmnožina jazyka DOT [1]):

```
digraph nadrazi_1 {
  vjezd1 -> uzel1;
  ...
  uzel1 -> vyjezd1;
  uzel1 -> vyjezd2;
}
```

4) Výstup: Úkolem je navrhnout program, který

- pro zadané nádraží navrhne řídící systém a zformalizuje podle uvedeného zadání;
- dokáže, že navržený řídící systém pracuje správně, tj. že se nádraží nemůže dostat do kritického stavu.

5) Automatické dokazování: Nádraží je modelováno v diskrétním čase. Čas je lineární, a každý časový okamžik má právě jeden následující a jeden předcházející. V každý časový okamžik si řídící systém nádraží určuje stavy výhybek a návěstidel.

Úkoly, které postupně zpracuje program pro libovolné nádraží pomocí nástrojů pro automatické dokazování:

- Formalizace nádraží:
  - Zformalizovat v logice 1. řádu v jazyce TPTP „fyzikální chování“ nádraží, tedy jak vlaky projíždějí nádražím na základě návěstidel a „rozhodování strojvedoucích“. Každý predikát p závislý na čase popisující něco, co dovedeme určit, musí být popsaný nejvýše jednou formulí tvaru $p(T+1) \Leftrightarrow \phi$, kde ϕ je formule závislá pouze na okolnostech v čase T a dřívějších (tím je syntakticky zaručena korektnost definice). Do toho spadá zejména stav výhybek, zda je v daném uzlu vlak, atd. Nespadá sem především vůle strojvedoucího, kterou neznáme (jen víme, že vždy nakonec s vlakem odjede).
  - Ukázat, že tato formalizace není sporná s přidanými podmínkami, že strojvůdce vlaku jede hned, jakmile může, a že do nádraží vjede vlak vždy, jakmile může.
- Zformalizovat návrh řídícího systému stejným způsobem jako v predešlém bodě.
- Ukázat, že je výsledná formalizace nádraží a jeho řízení bezesporná.
- Dokázat, že nikdy nenastane kritický stav.
- Nádraží musí pouštět vlaky hned, jakmile je to možné. Je třeba dokázat pro nejaké nádraží s jedním vstupem, že budou-li v čase t v tomto nádraží 2 vlaky, jeden nebo

víc vlaků na výstupu a jeden na vstupu, že se návěstidlo na vstupu v čase $t+1$ otevře.

II. Formalizace problému

A. Reprezentace grafu v logice prvního řádu

Vlakové nádraží je popsané orientovaným grafem. V níže popsané logické struktuře vrcholy grafu představují konstantné symboly (napr. vrchol a je popsaný symbolem $a/0$). Orientované hrany jsou popsane binárním predikátem $edge/2$. Term $edge(a,b)$ říká, že v grafu je přítomna hrana $\langle a,b\rangle$ a naopak neprítomnost této hrany je určena termem $\neg edge(a,b)$. Pro úplný popis grafu je potrebné specifikovat, které hrany jsou přítomny, a které přítomny nejsou, t.j.:

$$\Big(\bigwedge_{\langle a,b\rangle\in G} edge(a,b)\Big) \wedge \Big(\bigwedge_{\langle a,b\rangle\notin G} \neg edge(a,b)\Big)\ . \tag{1}$$

Dále nasleduje definice orientované cesty v grafu, reprezentované predikátem $path/2$. Hrana je zároven (elementární) cesta:

$$\forall a,b : edge(a,b) \Rightarrow path(a,b)\ , \tag{2}$$

a dále, cesta je tranzitivní:

$$\forall a,b,c : path(a,b) \wedge path(b,c) \Rightarrow path(a,c)\ . \tag{3}$$

Jestě je potřeba popsát vstupní, výstupní a divergetní uzly. Predikát $input/1$ je definován:

$$\forall x : input(x) \Leftrightarrow \bigvee_{y\in in(G)} (x=y)\ , \tag{4}$$

kde funkce in vrací množinu uzlů, do kterých nevede žádná hrana. Dále predikát $output/1$:

$$\forall x : output(x) \Leftrightarrow \bigvee_{y\in out(G)} (x=y)\ , \tag{5}$$

kde funkce out vrací množinu uzlů, ze kterých nevede žádná hrana. A koněčne predikát $diverge/1$:

$$\forall x : diverge(x) \Leftrightarrow \bigvee_{y\in more(G)} (x=y)\ , \tag{6}$$

kde funkce $more$ vrací množinu uzlů, s více než jedním potomkem.

B. Definice diskrétního času

Nejprve je potřebné definovat predikát lineárního uspořádaní $less$. Ten je definovaný následujícímy axiomy:

$$\forall x,y : less(x,y) \wedge less(y,x) \Rightarrow (x=y) \tag{7}$$

$$\forall x,y,z : less(x,y) \wedge less(y,z) \Rightarrow less(x,z) \tag{8}$$

$$\forall x,y : less(x,y) \vee less(y,x) \tag{9}$$

Vztahy (7), (8) a (9) představují antisymetrii, tranzitivitu a úplnost lineárního uspořádání.

S pomocí výše uvedeného predikátu $less/2$ definujeme funkci $succ/1$, která představuje přímeho následníka:

$$\forall x : less(x, succ(x)) \wedge (\forall y : less(y,x) \vee less(succ(x),y)) \tag{10}$$

a dále musí platit:

$$\forall x : succ(x) \neq x \tag{11}$$

Funkci $succ$ je možné použít k vyjádření následujícího časového okamžiku. Napr. hodnota $succ(succ(T))$ představuje posunutí o dva okamžiky vpřed proti hodnote T.

C. Pohyb vlaku

Poloha a smeřování vlaku v čase jsou určeny predikátem $at/3$; term $at(t,y,u)$ říká, že v čase t, v uzlu y je vlak směřující do uzlu u. Víme, že vlak směřující do uzlu u je v nejakém uzlu y v nejakém čase $succ(t)$ tehdy a jen tehdy, byl-li v daném uzlu v předešlém okamžiku a „nechtěl“ nebo nemohl z neho vyjet, anebo byl v předešlém uzlu a mohl a zároveň „chtěl“ vyjet:

$$\begin{aligned}\forall t,y,u : {} & at(succ(t),y,u) \Leftrightarrow \\ & (at(t,y,u) \wedge \neg(want(t,y) \wedge \exists z : may(t,y,z))) \vee \\ & (at(t,x,u) \wedge want(t,x) \wedge may(t,x,y))\end{aligned} \tag{12}$$

Predikát $want/2$ představuje „vůli“ strojvůdce, t.j. $want(t,x)$ znamená, že v čase t se „chce“ posunout z uzlu x dále. Predikát $may/3$ představuje možnost pokračovat, t.j. je-li u vstupů otevřené návěstidlo, případne u divergentního spojení sepnuta výhybka v daném směru, u ostatních uzlú možnosť pokračovat není omezena - vlak může vyjet jakmile strojvůdce „chce“. Term $may(t,x,y)$ znamená, že vlak může v čase t postoupit z uzlu x, do uzlu y.

$$\begin{aligned}\forall t,y,u : may(t,x,y) \Leftrightarrow {} & edge(x,y) \wedge \\ & ((input(x) \wedge signal(t,x)) \vee \\ & (diverge(x) \wedge branch(t,x,y)) \vee \\ & (\neg diverge(x) \wedge \neg input(x)))\end{aligned} \tag{13}$$

Dle zadání víme, že vlak do uzlu jednou přišel a taky, že jednou odjede. „Vůli“ strojvůdce je tedy možno definovat vzhledem k výskytu vlaku v nejakém uzlu:

$$\begin{aligned}\forall t,x,u : {} & (at(t,x,u) \wedge input(x) \wedge signal(t,x)) \Rightarrow \\ & \Rightarrow want(t,x)\end{aligned} \tag{14}$$

$$\begin{aligned}\forall t,x,u : {} & ((at(t,x,u) \wedge \neg input(x)) \Rightarrow \\ & \Rightarrow (\exists t_1 : less(t,t_1) \wedge want(t_1,x))\ ,\end{aligned} \tag{15}$$

t.j. pokud je vlak na vstupu a může vyjet, tak tak strojvůdce „chce“ vyjet, když na vstupu není tak bude chtít pokračovat v nejaký následujíci okamžik.

Je ale taky nutné ověřit jestli je formalizace není sporná, když přidáme podmínku, že vlak vyjede jakmile to je možné. Je třeba tedy přidat axiom:

$$\forall t,x : (\exists u : at(t,x,u) \wedge \neg input(x)) \Rightarrow want(t,x)\ . \tag{16}$$

D. Kritické stavy

Kritický stav v nejakem čase t je určen termem $crit(t)$. Ke kritickému stavu může dojít v následujích případech:

1) Návěstidlo zůstane trvale uzavřené:

$$\begin{aligned}& \forall t : (\exists x,u : input(x) \wedge at(t,x,u) \wedge \\ & (\neg\exists t_1 : less(t,t_1) \wedge signal(t_1,x))) \Rightarrow crit(t)\end{aligned} \tag{17}$$

2) Výhybka se přepne v okamžiku, kdy v uzlu je vlak:

$$\begin{aligned}\forall t : {} & (\exists x,u : diverge(x) \wedge at(t,x,u) \wedge \\ & (\neg\exists y,z : y \neq z \wedge branch(t,x,y) \wedge \\ & \wedge branch(succ(t),x,z))) \Rightarrow crit(succ(t))\end{aligned} \tag{18}$$

3) Do uzlu vjede víc než jeden vlak:

$$\begin{aligned} &\forall t : (\exists y, u : \\ &\quad (at(t, y, u) \wedge \neg(want(t, y) \wedge (\exists z : may(t, y, z)))\wedge \\ &\wedge (\exists x : at(t, x, u) \wedge want(t, x) \wedge may(t, x, y)))) \Rightarrow \\ &\Rightarrow crit(succ(t)) \end{aligned} \tag{19}$$

E. Řízení

Řídící systém řídí nadraží signalizací na vstupu - predikát $signal/2$, a překlápěním výhybky - predikát $branch/3$. Vlak smí vyjet ze vstupu x a v čase t pouze tehdy, platí-li $signal(t, x)$. Platí-li $branch(t, x, y)$ vlak může projít z uzlu x do uzlu y (za předpokladu, že jsou spojené hranou).

1) Signalizace na vstupu: U řízení vstupních signálů je potřebné prihlédnout také k možnému kritickému stavu, kdy některý vstup zůstane trvale uzavřen. Tomu je možné predejít použitím časovače (predikát $flop/2$). Povolení k výjezdu na vstupu x může nastat pouze v případe platnosti $flop(t, x)$. Pokud má nádraží dva vstupy - X a Y , tak platí:

$$\forall t : (flop(t, X) \wedge \neg flop(t, Y)) \vee (\neg flop(t, X) \wedge flop(t, Y)) \ , \tag{20}$$

t.j. nanejvýš jeden vstup může být aktivní. Pro dva stejné uzly dále platí:

$$\begin{aligned} &\forall t : flop(t, X) \Rightarrow \\ &\Rightarrow (\neg flop(succ(t), X) \wedge flop(succ(t), Y)) \end{aligned} \tag{21}$$

a

$$\begin{aligned} &\forall t : flop(t, Y) \Rightarrow \\ &\Rightarrow (\neg flop(succ(t), Y) \wedge flop(succ(t), X)) \ , \end{aligned} \tag{22}$$

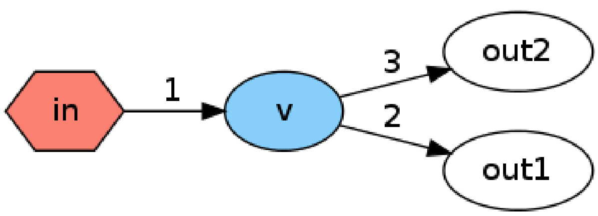
t.j. dochází k „přepínání" vstupů.

Dále je potřeba vyloučit možnost kolize vlaků. Predikát $block(t, x)$ říká, že daném čase t je v nádraží vlak v takovém místě, které je dosažitelné z daného vstupu x. V případě, že by byl vypuštěn další vlak, mohlo by dojít ke kolizi.

$$\begin{aligned} \forall t, z : &(input(z) \wedge (\exists x, u : at(t, x, u) \wedge \neg input(x)\wedge \\ &\wedge \neg(\exists y : path(x, y) \wedge path(y, z))) \Rightarrow block(t, z) \end{aligned} \tag{23}$$

Konečně můžeme napsat axiom řízení signalizace:

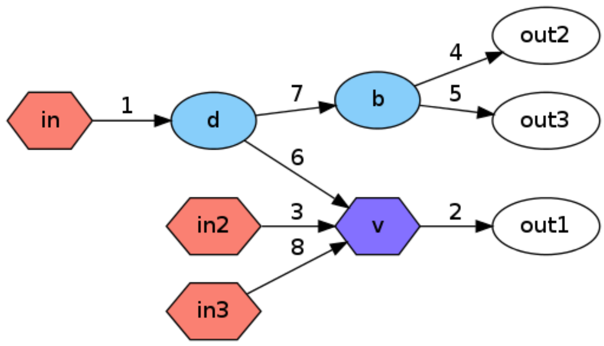
$$\forall t, x : (input(x) \wedge flop(t, x) \wedge \neg block(t, x)) \Rightarrow signal(t, x) \tag{24}$$



Obrázek 1: Vizualizace nádraží č. 1

2) Výhybky: výhybky jsou řízeny vzhledem k přítomnosti vlaku v nádraží a jeho cíle. Výhybka se sepne do polohy dle cesty v grafu.

$$\begin{aligned} \forall t, z, y : ((&diverge(z) \wedge edge(z, y)\wedge \\ &\wedge \exists x, u : (output(u) \wedge at(t, x, u)\wedge \\ &\wedge (path(x, z) \vee (x = z))\wedge \\ &\wedge (path(y, u) \vee (y = u)))) \Rightarrow \\ &\Rightarrow branch(t, z, y)) \end{aligned} \tag{25}$$



Obrázek 2: Vizualizace nádraží č. 2

F. Domněnky

Dle zadánní je možné formulovat nasledujíci domněnky, které je mozno testovat vůči výše uveděné formalizaci:

- v nádraží nikdy nenastane kritický stav,
- vlaky nejsou na vstupu zadržvány a jsou pouštěny jakmile je to možné.

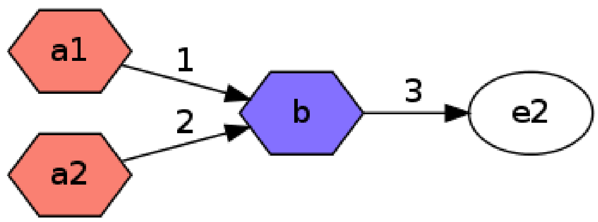
První domněnku by bylo možné formulovat následovne:

$$\begin{aligned} &\forall t : (\forall x, u : (at(t, x, u) \wedge input(x) \wedge output(u)) \Rightarrow \\ &\Rightarrow \neg \exists t_1 : (less(t, t_1) \wedge crit(t_1))) \end{aligned} \tag{26}$$

t.j. pro kazdy možný výskyt vlaku na vstupu, směřujícího do výstupu platí, že neexistuje časový okamžik v budoucnosti, kde by nastal kritický stav.

Další domněnka říká, že je-li na vstupu i výstupu vlak tak v dalsim časovém okamžiku se nádraží uvolní a čekající vlak se vpustí:

$$\begin{aligned} \forall t, x : (input(x) \wedge (\exists\, u, v : \\ output(u) &\ \wedge output(v)\wedge \\ \wedge at(t, x, u) &\ \wedge at(t, v, v))) \Rightarrow \\ &\Rightarrow signal(succ(t), x) \end{aligned} \tag{27}$$



Obrázek 3: Vizualizace nádraží č. 3

III. Implementace

V této sekci následuje popis prostředí pro automatizovanou vefifikaci popsaného problému.

Vstupem pro tento systém je jednoduchý popis nádraží ve formátu DOT [1] a výstupem jsou logické formule ve formátu TPTP [6].

Systém byl vytvořen v jazyku **Python 2.7.3** [7]. Dále je použitý nástroj **GNU Make 3.81** [5], ktorý ulehčuje tvormu skriptů. Je to potřebné kvůli automatickému spouštení různých programů, které dohromady tvoří popisovaný systém.

Pro účely této práce byl vytvořen jazyku Python modul **pytptp**. Je určen k manipulaci s derivačním stromem logických formulí. Formule a její elementy jsou reprezentovány objekty, pričem je použito přetížení aritmetických a logických operací a tím dochází k zjednodušení zápisu výrazů logiky prvního řádu v jazyce Python. Formule může být exportována do formátu TPTP, případně LaTeX. Formát TPTP pak lze použít jako vstup pro automatické dokazovače.

A. Instalace a ovládání

Po rozbalení balíku http://pborky.sk/download/au.tar.gz je kořenovém adresáři program `generator.py` a další podpůrné soubory. Po spuštění příkazu

```
python generator.py tpt < in/nadrazi0.in
```

případne

```
make a FILE="in/nadrazi0.in"
```

dojde ke zpracování souboru `nadrazi0.in` a k zobrazení logických formulí na standartní výstup.

Zároveň budou v aktualním adresári vytvořeny soubory `ltl.tpt` (obsahuje axiomy LTL), `control.tpt` (obsahuje axiomy pro řídící systém), `graph.tpt` (axiomy nádraží, pohybu vlaku a kritických stavů), `t0.tpt` (test „vůle" strojvůdce), `t1.tpt` (test, že se vlak vždy dostane ze vstupu na požadovaný výstup), `t2.tpt` (test, že nenastane kritický stav) a `t3.tpt` (test signalizace - musí pouštět vlk jakmile to je možné). Pomocí direktivy include jsou v adresáři `tests/` pripraveny testy, pro úkoly dle zadání:

- důkaz, že formalizace „fyzikálního chování" nádraží není sporná,
- důkaz, že formalizace nádraží a jeho řízení není sporná,
- důkaz, že nikdy nenastane kritický stav,
- a důkaz, že nádraží pouští vlaky jakmile je to možné.

Po správném nastavení cesty k programům **Prover9** alebo **Mace4** v souboru `Makefile` je možno spustit testy s dokazovači **Mace4** a **Prover9** pomocí příkazu

```
make verify
```

případně

```
make verify FILE="in/nadrazi0.in"
```

Vynecháním promenné `FILE` dojde ke zpracování všech soubor u v adresáři `in/`. Do adresáře `out/` se uloží soubory ve formátu TPT a v adresáři `logs/` budou výstupy programů **Mace4** a **Prover9**.

IV. Experimenty a závěr

Formalizace řešených problémů je v sekci II. Formalizováno bylo jak „fyzikální" chování nádraží tak aj jeho řízení. V této sekci se zaměřuje na automativké ověření správnosti zmíněné formalizace. Za tímto účelem byl vytvořen softvér, ktorého stručný popis je v sekci III.

Instance na, kterých byli experimenty prevedeny jsou na obr. 1, 2 a 3.

Experimenty byly provedeny s automatickými dokazovači **E** [4], **Vampire** [3] , **Prover9** [2] a hledadačem modelů **Mace4** [2].

Přiloženy jsou výsledky pouze z dokazovače **Prover9** a **Mace4** a jsou přístupné v adresáři `logs/`. Úkoly, kde je potřebné dokázat, že axiomy nejsou sporné, byly testovány pomocí **Mace4**.

V adresáři `tests/` jsou soubory `run1.tpt.mace4`, `run2.tpt.mace4`, `run5.tpt.prover9` a `run6.tpt.prover9`, které obsahují jenom direktivy `include` jazyka TPT.

Test **run1** je vytvořen pro ověření konzistence modelu nádraží a test **run2** zahrňuje i axiomy řízení. Oba jsou předány k spracování programu **Mace4**. Z teorie byl vypušten axiom (11) kvůli urychlení zpracování. Test **run1** byl proveden poměrně snadno – viď. výstupy `nadrazi?_run1.log`. Problematické ale bylo hledání modelu po přidání axiomů řízení (t.j. **run2**). Problém byl způsobený axiomama (20) a (21) resp. (22). Z nich je možno odvodit původně vyloučený axiom (11). Právě proto byl vypuštěn i axiom (20).

Pro řešení dalších stanovených úkolů – t.j. důkaz, že nikdy nenastane kritický stav a dále okamžité pouštění vlaku do nádraží – jsou určeny testy **run5** a **run6**. Test **run6** nebylo možné verifikovat v přednastaveném čase 120sec. Důkaz správnosti **run5**, t.j. že v nádraží nenastane kritický stav. je v přiloženém souboru `nadrazi?_run5.log`.

Reference

[1] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *Software- Practice and Experience*, 30(11):1203–1233, 2000.

[2] W. McCune. Prover9 and mace4. http://www.cs.unm.edu/~mccune/prover9/, 2005–2010.

[3] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *Journal of AI Communications*, 15(2/3):91–110, 2002.

[4] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[5] R.M. Stallman, R. McGrath, and P.D. Smith. *GNU make: a program for directed recompilation : GNU make version 3.81*. A GNU manual. Free Software Foundation, 2004. Also available as http://www.gnu.org/software/make/manual/make.html.

[6] G. Sutcliffe. The tptp problem library and associated infrastructure: The fof and cnf parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[7] G. Van Rossum and F.L.J. Drake. *The Python Language Reference Manual*. Network Theory Limited, 2011. Also available as http://docs.python.org/release/2.7.3/reference/index.html.