Choco

Choco je řešič omezujících podmínek implementovaný jako knihovna pro programovací jazyk Java. Choco je distribuováno jako balíček JAR s dokumentací ve formátu Javadoc. Instalace je tedy velmi snadná i pro začínajícího programátora v Javě a v běžně používaných IDE nezabere více než pět minut. Díky programovacímu jazyku Java je solver Choco dostupný na mnoha platformách a operačních systémech. Protože ale není cílem této práce popisovat vlastnosti hostitelského programovacího jazyka, nebudeme zde dále rozebírat možnosti Javy. Choco je vyvíjeno na Ecole des Mines de Nantes ve Francii a je volně dostupné na serveru Sourceforge. Aktuální hlavní verze má číslo 2. Choco dělí řešeí problému na dvě části – model a samotý řešič. Model obsahuje proměnné a podmínky dané zadáním. Řešič pak dostane na vstupu model a pokusí se v něm najít řešení. Model může obsahovat jako proměnné celá čísla, reálná čísla a množiny. Řešič je pak schopný k danému modelu najít řešení. Uživatel může od řešiče získat informaci, zda je problém řešitelný či zda neobsahuje spor. Pro samotná řešení poskytuje interface umožňující dotázat se na úrvní, následující a nebo na všechna řešení. Pokud mu určíme proměnnou s hodnotou objektivní funkce umožňuje řešič tuto proměnnou minimalizovat nebo maximalizovat. Řešič dále podporuje volbu strategií na míru řešenému problému. Proměnné modelu a řešiče jsou ve vzájemném vztahu. Přesto není možé získat řešení přímo z proměnných modelu ale je nutné je získat přes odpovídající proměnné řešiče.

# Popis řešiče omezujících podmínek

Jak bylo naznačeno v předchozí sekci, je řešení problému rozděleno na dva úkoly – nadefinovat model a předložit ho správně nastavenému řešiči. Jak model tak řešič jsou třídy, které si uživatel instancuje do příslušného objektu. Popíšeme nejprve model a poté řešič.

## Model

Model je tvořen objektem třídy CPModel. Proměnné modelu jsou v Choco reprezentovány objekty následujících typů IntegerVariable, RealVariable a SetVariable. Tyto proměnné se standardně nevytváření pomocí klíčového slova new, ale choco má pro tento účel připravené tovární metody. Takovéto proměnné je nutné v modelu zaregistrovat pomocí funkce CPModel::addVariable resp. CPModel.addVariables (pro zaregistrování celého pole proměnných). Při registraci proměnné do modelu je možné určit dodatečné vlastnosti proměnných, například zda jde o decision proměnné, nebo o proměnnou obsahující výsledek objektivní funkce. Tyto vlastnosti není nutné vždy zadávat, ale mohou výrazně urychlit výpočet. Alternativou je definovat si tyto vlastnosti v řešiči. Toto bude popsáno v následující sekci.

Po zaregistrování proměnných následuje definice omezujících podmínek. Je možné použít jak širokou nabídku podmínek dodaných s řešičem, tak si definovat vlastní podmínky. Podmínky dodané s řešičem jsou vypsané v tabulce (tabulka). Lze je rozdělit na základní podmínky (pravda, nepravda, operátory uspořádání), a základní výrazy (goniometrické funkce, mocniny, sumy), ostatní podmínky (abs, div, max, ...), reifikované podmínky (and, or, ifOnlyIf) a globální podmínky (allDifferent, occurenceMax, ...). K dispozici jsou také podmínky pro modelování geometricých podmínek, podmínky pro rozvrhování a podmínky pro sekkvenci proměnných, které může přijímat konečný automat.

Vedle dodaných podmínek je možné definovat podmínky vlastní. Jednou možností je definovat podmínku p(x,y) jako množinu kompatibilních hod not (a,b), tedy p platí, pokud x=a a y=b, případně jako množinu nekompatibilních hodnot. Tato množina je zadaná jako tabulka hodnot. Vedle zadání tabulkou je možné zadat podmínku jako predikát, který musí platit. Podmínka pak je tvořena objektem typu poděděného od třídy BinRelation s implementovanou funkcí checkCouple. Tato funkce dostane jako parametry dvojici hodnot a vrátí bool hodnotu zda byla podmínka splněna nebo ne. Vedle binárních podmínek je možné specifikovat podmínky podobným stylem na n-ticích. Pro všechny takové podmínky (jak binární tak n-ární) lze pak specifikovat požadovaný algoritmus hranové konzistence. K dispozici je AC3, AC2001, AC3rm, AC3 pro binární podmínky a AC32, AC3rm, AC2001 a AC2008 pro n-ární podmínky. Popis jednotlivých algoritmů můžete nalézt v [najít citaci pro algoritmy AC].

## Řešič

Řešič je objekt typu CPSolver, který se na základě modelu popsaného v předhoczí sekci pokusí najít řešení. Řešič nejprve přečte proměnné modelu, které převede na proměnné řešiče (IntegerVariable na IntDomainVar, RealVariable na RealVar a SetVariable na SetVar). Poté přečte podmínky z modelu a vyrobí na jejich základě podmínky řešiče. Řešič poté použije vyhledávací strategii a hledá řešení. Protože je volba správné strategie klíčovým faktorem pro rychlost řešení, je možné nastavit její různé parametry. Uživatel si může zvolit tzv. selektor a iterátor. Selektor vybírá, kterou proměnnou vybrat k příštímu rozhodnutí solveru. Iterátor pak vybírá jednotlivé možné hodnoty a iteruje přes ně. Choco ve standardní distribucí nabízí základní selektory jako proměnnou s minimální doménou, proměnnou s maximální doménou apod. Iterátory pak mohou zkoušet hodnoty v sestupném nebo vzestupném pořadí. Alternativou k iterátoru je selektor hodnoty, který na požádání vrátí další možnou hodnotu. Pro selektor hodnoty je možné použít například minimální hodnotu domény, náhodnou hodnotu domény apod. Pro různé skupiny proměnných je možné zvolit vlastní strategie tak, aby co nejlépe vystihovaly zadání problému. V tomto případě se definuje chování řešiče pomocí tzv. cílů. Cíl obsahuje definici strategie tzn. selektor nad danými proměnnými a iterátor nad hodnotami.

Protože při řešení rozsáhlých problémů může dojít k neúměrnému času výpočtu, spotřebovaných systémových prostředků apod, lze definovat omezení na řešič. Řešič je možné omezit časem, počtem uzlů vyhledávacího stromu, hloubkou backtrackingu, počtem neúspěšných řešení a nebo CPU časem. Uživatel si může vedle těchto limitů na řešení nastavit vlastní limity.

Po přečtení modelu a definici strategií přichází samotné řešení problému. Řešič poskytuje interface jak pro iterování přes řešení (solve, nextSolution), tak pro získání všech řešení naráz. Dále řešič umožňuje určit proměnnou, kterou se pokusí minimalizovat/maximalizovat. Řešení je uloženo v proměnných řešiče, ne v proměnných modelu, které byly uživatelem definovány. Je tedy nutné proměnné řešiče z řešiče získat zavoláním funkce CPSolver::getVar která jako parametr přijme proměnnou modelu a vrátí proměnnou řešiče.

# Nástroje pro podporu modelování

Choco neobsahuje nástroj na grafickou visualisaci vyhledávacího stromu podobně jako má například systémk Mozart nebo Gecode. Choco umožňuje vypsat log z průběhu řešení. Logu se dají nastavit různé úrovně podrobnosti od zcela tichého až po kompletní výpis všech událostí, které choco interně provádí. [Protyp pokud bude odpoved autoru zaporna] Mimo zmíněného logování uživatel nemá k dispozici jiné debugovací nástroje jako například interaktivní procházení prohledávacího stromu a nebo zmíněnou visualisaci. [/prototyp]

# Subjektivní hodnocení solveru

Solver Choco se ukázal jednoduchý na běžné použití. I člověk, které předtím neměl zkušenosti s javou, omezujícími podmínkami ani tímto solverem neměl problém začít používat solver a prakticky ho uplatnit. Systém je dobře zdokumentován, i když v době psaní této práce je dokumentace lehce nepřehledná. Přesto v ní pečlivvý čtenář najde zpravidla vše co potřebuje. Programátorská dokumentace je generovaná systémem JavaDoc. Díky tomu je ve většině běžných Java IDE uživateli k dispozici již během výběru funkce pomocí intellisense, což napomáhá rychlému pochopení řešiče. Vývoj řešiče je provozován na serveru SourceForge, je tedy jednoduchou standardní formou přístup jak ke zdrojovým kódům aplikace tak k historii skrze verzovací nástroj subversion. Na serveru sourceforge je také umístěné fórum technické podpory, ve kterém autoři odpovídají na dotazy uživatelů. Reakční doba je velmi nízká a úroveň odpovědí na takové úrovni, že většina problémů je rychle vyřešena.

|  |  |
| --- | --- |
| Podmínka | Popis podmínky |
| abs |  |
| allDifferent |  |
| and |  |
| atMostNValue |  |
| boolChanneling |  |
| Cumulative |  |
| disjunctive |  |
| distanceEQ |  |
| distanceGT |  |
| distanceLT |  |
| distanceNEQ |  |
| eq |  |
| eqCard |  |
| equation |  |
| FALSE |  |
| feasPairAC |  |
| feasTupleAC |  |
| feasTupleFC |  |
| geost |  |
| geq |  |
| geqCard |  |
| globalCardinality |  |
| gt |  |
| ifOnlyIf |  |
| ifThenElse |  |
| implies |  |
| infeasPairAC |  |
| infeasTupleAC |  |
| infeasTupleFC |  |
| intDiv |  |
| inverseChanneling |  |
| isIncluded |  |
| isNotIncluded |  |
| leq |  |
| leqCard |  |
| lex |  |
| lexChain |  |
| lexChainEq |  |
| lexeq |  |
| leximin |  |
| lt |  |
| max |  |
| member |  |
| min |  |
| mod |  |
| neq |  |
| neqCard |  |
| not |  |
| notMember |  |
| nth |  |
| occurenceMax |  |
| occurenceMin |  |
| occurence |  |
| oppositeSign |  |
| or |  |
| pack |  |
| precedenceReified |  |
| preceding |  |
| regular |  |
| reifiedIntConstraint |  |
| relationPairAC |  |
| relationTupleAC |  |
| relationTupleFC |  |
| sameSign |  |
| setDisjoint |  |
| setInter |  |
| setUnion |  |
| sorting |  |
| stretchPath |  |
| times |  |
| tree |  |
| TRUE |  |

|  |  |
| --- | --- |
| abs |  |
| cos |  |
| div |  |
| FALSE |  |
| ifThen |  |
| max |  |
| min |  |
| minus |  |
| mod |  |
| mult |  |
| neg |  |
| plus |  |
| power |  |
| scalar |  |
| sin |  |
| sum |  |
| TRUE |  |