Mozart/Oz

Mozart je implementací multiparadigmatického jazyka Oz. Oz je funkcionální jazyk, který má vestavěnou podporu pro vícevláknové aplikace, podporu pro paralelizaci a mimo jiné obsahuje také vestavěnou podporu pro řešené problémů s omezujícími podmínkami. Jak bylo zmíněno dříve, jde o jazyk multiparadigmatický, lze v něm tedy psát i imperativní programy stejně jako logické programy (podobné prologu). Dále je možné definovat třídy včetně dědičnosti a praocvat s od nich odvozenými objekty. Tento jazyk byl navržen pro co nejvyšší variabilitu při použití, protože programátor může využít naráz kombinaci funkctionálního, logického i imperativního programování, OOP a další najednou. Ve standardní distribuci je dodáván jako samostatný překladač, který vytvoří nativní programy pro daný operacní systém. Mimo toho umí také běžet v interaktivním módu, kdy se jednotlivé bloky příkazů posílají přímo kompilátoru. Jako IDE se používá běžně systém EMACS, se kterým je Mozart/Oz dodáván.

Podobně jako v jiných funkcionálních jazycích je do proměnné možné přiřadit pouze jednu hodnotu za dobu jejího života. Proto má každá proměnná vedle hodnoty ještě stav. V případě, že má dojít k akci nad proměnnou do které není nic přiřazeno, ač by to příkaz vyžadoval dojde k pozastavení vlákna dokud nebude do proměnné přiřazeno. To umožňuje provést následující program:

a = 5  
if a > b then c = 5 else c = 6  
b = 4  
// V c nyní bude hodnota 5

Jazyk obsahuje vestavěnou podporu pro paralelní výpočty, tedy na úrovni jazyka je možné vytvářet vlákna a synchronizační primitiva. Jazyk navíc podporuje předat výpočet na jiný počítač skrze protokol TCP/IP. Lze tedy využít cluster počítačů pro zrychlení náročných výpočtů.

# Popis řešiče omezujících podmínek

Přo řešení problémů s omezujícími podmínkami obsahuje vestavěný řešič problémů s konečnou doménou (finite domain). Pro tyto účely se množinou s konečnou doménou rozumí konečná množina přirozených čísel s nulou. Řešič obsahuje omezení na nejvyšší možnou hodnotu proměnné. Výpočetní model pro propagaci podmínek se jmenuje *prostor (space)* a je tvořen několika propagátory napojenými na constraint store. Constraint store obsahuje konjunkci základních podmínek, tedy podmínek ve tvaru x=n nebo x ∈ D. Tedy může vypadat například x = 6 & y ∈ 1..12 & z = y. Propagátory obsahují ostatní podmínky, tedy například x>y nebo a2 + b2 = c2. Propagátor pro podmínku *c* je samostatný výpočetní agent, který se snaží omezit domény proměnných, které jsou obsažené v *c*. Řešením je pak takové přiřazení hodnot proměnným, které splní všechny podmínky dané propagátory.

Příklad:

Mějme proměnné X a Y a následující podmínky: X ∈ 0..9, Y ∈ 0..9, X+Y = 9, 2X + 4Y = 24.

1. Constraint store obsahuje: X ∈ 0..9, Y ∈ 0..9  
   Propagátory: X+Y = 9 a 2X + 4Y = 24
2. První propagátor nemůže udělat nic, druhý ale může změnit constraint store tak, že obsahuje X ∈ 0..8, Y ∈ 2..6
3. První propagátor nyní může změnit constraint store na X ∈ 3..7, Y ∈ 2..6
4. Druhý propagátor nyní změní constraint strore na X ∈ 4..6, Y ∈ 3..4
5. První propagátor změní constraint store na X ∈ 5..6, Y ∈ 3..4
6. Druhý propagátor nakonec změní constraint store, tak že obsahuje X = 6, Y = 3.

Propagace může být buď intervalová nebo doménová. Zatímco intervalová propagace pouze mění okraje domény, doménová také odstraňuje vnitřní hodnoty domény. Přestože doménová propagace vypadá jako lepší techhnika, používá se spíše intervalová pro její výpočetní jednoduchost.

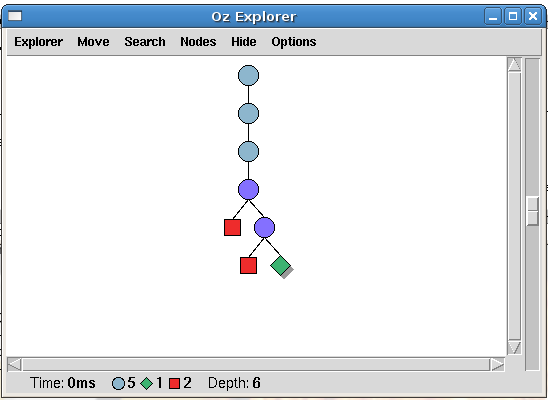
Již pro jednoduché problémy ale propagace nemusí vést k výsledku. Vezměme například problém: x ≠ y, x ≠ z, y ≠ z, x ∈ 0..1, y 0..1, z ∈ 0..1. Tento problém nelze více propagovat, všechny podmínky lze splnit ale žádná proměnné není přiřazena hodnota. V této chvíli lze použít distribuci. Problém P je distribuován pomocí podmínky C, pokud vyřešíme nové problémy P ∪{¬C} a P{C}. Alespoň v jednom z nových problémů se buď nachází řešení problému P a nebo je problém neřešitelný. V naší reprezentaci se tohoto faktu dá využít pro space. Pokud se systém dostane do stabilního stavu, kdy žádná podmínka není nesplněna a přitom není možné přiřadit žádné proměnné hodnotu, tak vybereme proměnnou x a hodnotu n takovou, že je konzistentní se všemi podmínkami nad x. Tím získáme dva nové space, jeden S ∪ {x = n} a jeden S ∪ {x ≠ n}. Na obou space pak pustíme propagaci. Pokud se dostane opět do stabilního stavu, znova distribuujeme a takto postupujeme až do vyřešení problému a nebo konstatování, že je problém neřešitelný. Tato metoda je úplná, tedy pokud existuje alespoň jedno řešení, pak budou nalezena všechna.

Je možné volit z různých distribučních strategií. Volba správné strategie může výrazně ovlivnit dobu výpočtu. Pro většinu problémů je dobré zvolit first-fail strategii, tedy volit proměnnou s nejmenší možnou doménou. Pokud ale žádná dodávaná distribuční strategie nevyhovuje, je možné implementovat své distribuční strategie, plně na míru problému.

Pro hledání řešení optimalizačních problémů se dají využít dvě techniky. Naivní technika spočívá v postupném zvyšování výsledku objektivní funkce a následném hledání řešení, které tomuto výsledku bude odpovídat. Pro tento postup existuje v jazyce konstrukce, která před splňováním ostatních podmínek postupně zvyšuje danou proměnnou. Tento postup však postrádá obecnost a Mozart nabízí lepší postup – techniku branch and bound. Tato technika vyžaduje, aby uživatel sestavil porovnávací funkci. Porovnávací funkce jako argumenty přijímá předchozí a aktuální řešení problému. Funkce pak zpravidla zavolá objektivní funkci a porovná její výsledky. Tato technika oproti naivní výrazně zrychlí výpočet.

# Nástroje pro podporu modelování

Mozart nabízí uživateli interaktivní nástroj Explorer, který umožňuje prozkoumat strom řešení včetně dílčích rozhodnutí, které řešič během výpočtu učinil. Explorer je možné používat také v interaktivním módu a ručně určovat, které větve řešení prozkoumat.



Kolečka označují uzly rozhodovacího stromu, ve kterých proběhlo rozhodnutí, kosočtverce označují nalezení úspěšného řešení a čtverec větev kde není řešení. Světlejší barvou jsou uvedené uzly, které je možné ještě dále expandovat. Na uvedeném obráýku tedy máme jedno řešení, dvě neúspěšná řešení a pět okamžiků, kdy došlo k rozhodnutí z nichž lze ještě ve třech hledat další řešení. Explorer umožňuje také export nakresleného stromu ve formátu PostScript.

# Implementace benchmarků

Každý implemetovaný benchmark je samostatný solver. Po zavolání příslušného solveru je vrácena funkce, která je posléze předána funkci realizující hledání řešení. Jde buď o funkce SearchOne resp. SearchAll, které naleznou jedno nebo všechna řešení problému a vrátí je v seznamu. Pro spuštění exploreru a interaktivní hledání řešení lze pak zavolat ExploreOne resp. ExploreAll.

## Magická sekvence

**fun {MagicSequence Size}**

**proc {$ Sol}**

**{FD.tuple magic Size 0#(Size-1) Sol}**

**{For 1 Size 1**

**proc {$ I}**

**Vector**

**in**

**{FD.tuple v Size 0#1 Vector}**

**{For 1 Size 1**

**proc {$ J}**

**Vector.J = Sol.J =: I-1**

**end}**

**{FD.reified.card Sol.I V Sol.I 1}**

**end}**

**{FD.distribute ff Sol}**

**end**

**end**