Programação Lógica com Restrições (CLP)

- Objetivo: resolver 2 limitações em Prolog:
  - Cada termo em Prolog precisa ser explicitamente codificado e não é interpretado (avaliado):
    - `X + 1` é um termo não avaliado em Prolog.
    - uma variável somente pode assumir um único valor definido sintática e semanticamente na linguagem: átomo, inteiro ou estrutura.
  - Computação uiforme, mas não tão poderosa: busca em profundidade, busca de soluções do tipo "generate-and-test".

Programação Lógica com Restrições

- CLP utiliza outras técnicas utilizadas em IA para tornar procedimentos de busca mais inteligentes: propagação, computação guiada pelos dados, "forward checking" e "lookahead".
- Aplicações: planning, escalonamento, alocação de recursos, computação gráfica, projeto de circuitos, diagnóstico de falhas etc.
- Clientes: Michelin and Dassault, French railway SNCF, Swissair, SAS and Cathay Pacific, HK International terminals, Eriksson alemã, British Telecom etc.

Programação Lógica com Restrições

- CLP vem de duas linhas principais de pesquisa:
  - Introdução de estruturas de dados mais ricas e poderosas em PL (ex: substituir unificação por manipulação de restrições em um domínio).
  - Técnicas de consistência: uso ativo de restrições para reduzir o espaço de busca.

    "generate-and-test" x "constrain-and-generate"

Programação Lógica com Restrições

- Sistemas:
  - CHIP, Dincbas and Van Hentenryck (ECRC)
  - CLP(R), Jaffar, Michaylov, Stuckey and Yap (Monash)
  - Prolog III, Colmerauer
  - Eclipse, Wallace (IC Parc)
  - Oz, Smolka (DFKI)
  - clp(FD), Diaz and Codognet (INRIA, France)
- O esquema CLP(X):
  - "constraint solver": substitui simples unificação.
  - 2 domínios populares: artimético e booleano.

Domínio Aritmético: Restrições Lineares

- Prolog não consegue resolver `x-3 = y+5`.
- CLP(R): 1a. linguagem a introduzir restrições aritméticas.
- Expressões aritméticas lineares compostas de: números, variáveis e operadores (negação, adição, subtração, multiplicação e divisão).
- Expressão no domínio aritmético: `t1 R t2`, com R = $\{ >, \geq, =, \leq, <, = \}$
- Procedimentos de decisão mais utilizados:
  - Método de eliminação de Gauss.
  - Método simplex (para desigualdades) é mais usado:
    - bom comportamento em média

* popular
* incremental

Domínio Aritmético: Restrições Lineares

- Exemplo:
- Uma refeição é composta de entrada, prato principal e sobremesa.
- Banco de dados de alimentos e seus valores calóricos.
- Problema: produzir um cardápio com refeições leves, i.e., refeições cujo valor calórico não exceda 10Kcal.

Domínio Aritmético: Restrições Lineares

```
ref_leve(A,M,D) :-
    I > 0, J > 0, K > 0,
    I + J + K =< 10,
    entrada(A,I),
    principal(M,J),
    sobremesa(D,K).

    carne(bife,5).
    carne(porco,7).
    peixe(linguado,2).
    peixe(atum,4).
    sobremesa(fruta,2).
    sobremesa(sorvete,6).
```

```
principal(M,I) :-
   carne(M,I).
principal(M,I) :-
   peixe(M,I).
entrada(salada,1).
entrada(sopa,6).
```

Domínio Aritmético: Restrições Lineares

- Resultados intermediários = estados da computação.
- 2 componentes: constraint store e continuação dos objetivos.
- Consulta: ◇ `ref_leve(A,M,D).`
- `I + J + K =< 10, I > 0 J > 0, K > 0` ◇ `entrada(A,I), principal(M,J), sobremesa(D,K).`
- `A = salada, I = 1, 1 + J + K =< 10, 1>0, J>0, K>0` ◇ `principal(M,J), sobremesa(D,K).`
- `A = salada, I = 1, M=M1, J=I1, 1 + J + K =< 10, 1>0, J>0, K>0` ◇ `carne(M1,I1), sobremesa(D,K).`
- `A = salada, I = 1, M=bife, J=5, M1=bife, I1 = 5, 1 + 5 + K =< 10, 1>0, 5>0, K>0` ◇ `sobremesa(D,K).`

- `A = salada, I = 1, M=bife, J=5, M1=bife, I1 = 5, D=fruta, K = 2, 1 + 5 + 2 =< 10, 1>0, 5>0, 2>0` $\diamond$.

Domínio Aritmético: Restrições Lineares

Derivação inconsistente:

- `A = pasta, I = 6, M=bife, J=5, M1=bife, I1 = 5, 6 + 5 + K =< 10, 5>0, 6>0, K > 0` $\diamond$ `sobremesa(D,K).`

Domínio Aritmético: Restrições não Lineares

- Programa para multiplicar 2 números complexos: `R1 + I*I1` * R2 + I*I2.

```
zmul(R1,I1,R2,I2,R3,I3) :-
    R3 = R1 * R2 + I1 * I2,
    I3 = R1 * I2 + R2 * I1.
```

- Consulta: $\diamond$ `zmul(1,2,3,4),R3,I3)`
- Equações se tornam lineares.
- Solução: `R3 = -5`, `I3 = 10` (solução definida)
- Consulta: $\diamond$ `zmul(1,2,R2,I2,R3,I3)`
- Solução:

```
I2 = 0.2*I3 - 0.4*R3
R2 = 0.4*I3 + 0.2*R3
yes (solucao indefinida)
```

Domínio Aritmético: Restrições não Lineares

- Programa para multiplicar 2 números complexos: `R1 + I*I1` * R2 + I*I2.
- Consulta: ◇ `zmul(R1,2,R2,4,-5,10), R2 < 3.`
- CLP(R): (não resolve equações não lineares)

```
R1 = -0.5*R2 + 2.5
3 = R1*R2
R2 < 3
Maybe
```

- aplicações de equações não lineares: geometria computacional e aplicações financeiras (vários algoritmos utilizados).

Domínio Booleano

- Aplicação principal: projeto de circuitos (verificação de hardware) e prova de teoremas.
- termos booleanos: valores verdade (F ou V), variáveis, conectivos lógicos, única restrição: igualdade.
- vários algoritmos de unificação para restrições booleanas.
- Solução provê procedimento de decisão para cálculo proposicional (NP-completo).

Domínio Booleano

- Exemplo: somador completo (operadores # (xor), * (and), + (or)

```
add(I1,I2,I3,O1,O2) :-
    X1 = I1 # I2,
    A1 = I1 * I2,
    O1 = X1 # I3,
    A2 = I3 * X1,
    O2 = A1 + A2.
```

- Consulta: $\diamond$ `add(a,b,c,O1,O2)`
- Resposta: `O1 = a+b+c, O2 = (a ∧ b) + (a ∧ c) # (b ∧ c)`

Técnicas de Consistência

- Eliminam 'labellings' inconsistentes em estágios anteriores da computação *propagando* informação sobre as variáveis.
- exemplos: arc-consistency, forward checking, propagação generalizada.
- Exemplo: escalonamento de tarefas.

```
      T2                       before(T1,T2).
     /                         before(T1,T3).
    /                          before(T2,T6).
  T1                           before(T3,T5).
    \          T6              before(T4,T5).
     \        /                before(T5,T6).
      T3 --- T5                notequal(T2,T3).
```

/

T4

Técnicas de consistência

- funcionam propagando informações sobre as variáveis.
- Exemplo: $T1 \in \{1, 2, 3, 4, 5\}$, $T2 \in \{1, 2, 3, 4, 5\}$.
- before(T1,T2) –¿ técnica de consistência:
  - $T1 \in \{1, 2, 3, 4\}$
  - $T2 \in \{2, 3, 4, 5\}$
- Valor 5 removido de T1, pois não existe nenhum outro valor em T2 que atenda T1 < T2.
- Valor 1 removido de T2, pela mesma razão.

Consistência de arco (arc consistency)

- $T1 \in \{1,2\}$, $T2 \in \{2,3,4\}$, $T3 \in \{2,3\}$, $T4 \in \{1,2,3\}$, $T5 \in \{3,4\}$, $T6 \in \{4,5\}$.
- Valor 2 de T1 é selecionado (T1 é "labelled" com valor 2).
- Por propagação: $T2 \in \{3,4\}$ e $T3 \in \{3\}$
- $T2 \in \{4\}$ por notequal(T2,T3).
- No final: $T1 \in \{2\}$,$T2 \in \{4\}$, $T3 \in \{3\}$, $T4 \in \{1,2,3\}$, $T5 \in \{4\}$, $T6 \in \{5\}$

Domínios Infinitos

- Utilização de máximos e mínimos para resolver restrições em domínios numéricos lineares.
- Ex: X,Y,Z têm domínios [1..10] com restrição 2x + 3y + 2 < z
- Removendo valores inconsistentes:
  - 10 é o maior valor para Z, logo: 2x + 3y < 8
  - 1 é p menor valor possível para y, logo: 2x < 5
  - X só pode assumir valores {1,2}
  - 3y < 6, y < 2, $y \in \{1\}$
  - z > 7, $z \in \{8, 9, 10\}$

Técnica básica de programação

- Declarar variáveis do problema e seus domínios.
- Estabelecer as restrições.
- Procurar solução.

```
?- [X,Y,Z]::1..10,
   2 * X + 3 * Y + 2 #< Z,
   indomain(X), indomain(Y), indomain(Z).
```