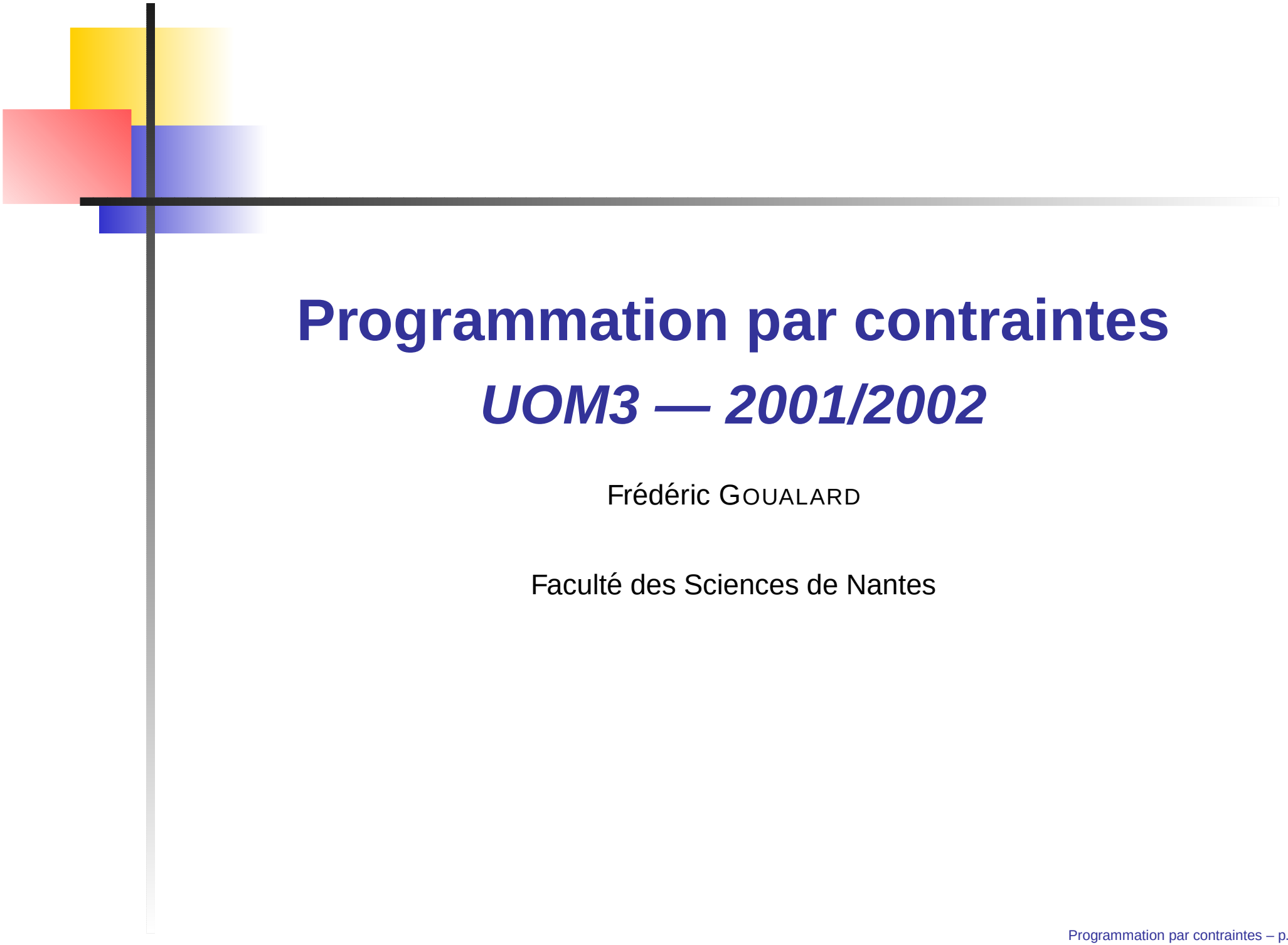


Programmation par contraintes

UOM3 — 2001/2002

Frédéric GOUALARD

Faculté des Sciences de Nantes

Organisation du module (1)

- **Cours 1 à 3 : Frédéric GOUALARD**

  Bureau 208 à l'IRIN

  Frederic.Goualard@irin.univ-nantes.fr

- **Cours 4 à 6 : Laurent GRANVILLIERS**

  Bureau 221 à l'IRIN

  Laurent.Granvilliers@irin.univ-nantes.fr

- **Cours 7 à 9 : Éric MONFROY**

  Bureau 209 à l'IRIN

  Eric.Monfroy@irin.univ-nantes.fr

- **Cours 10 : ? ? ?**

Organisation du module (2)

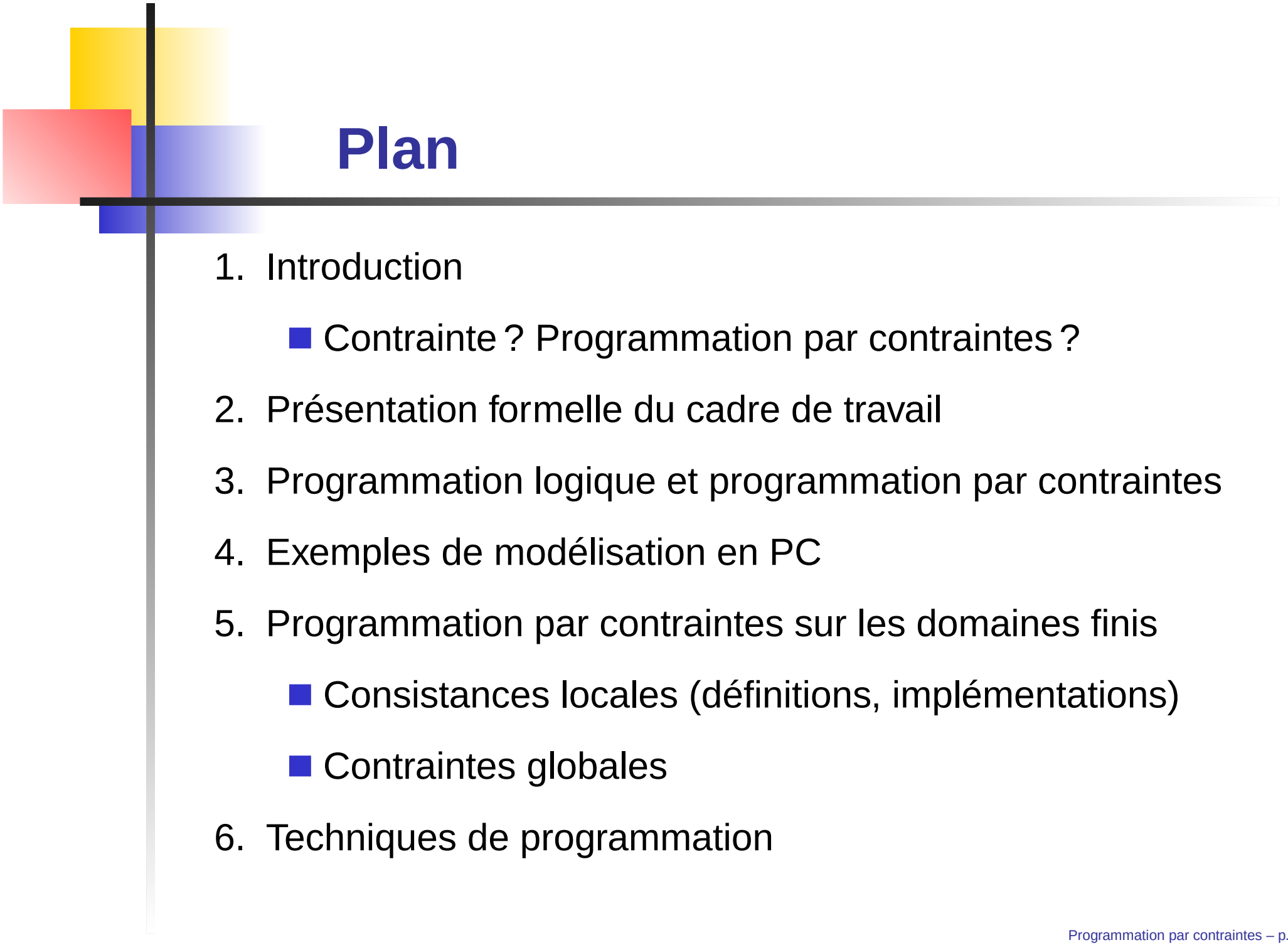
Semaine	Cours	TD	TP
14 janvier	✓	—	—
21 janvier	✓	✓	—
28 janvier	✓	✓	✓
4 février	*LG*	✓	—
11 février	*LG*	✓	✓

Transparents de cours, feuilles de tds/tps :

`www.sciences.univ-nantes.fr/info/perso/permanents/goualard/Teaching/`

Bibliographie

- *Programming with constraints : an introduction*. Kim Marriott et Peter J. Stuckey. The MIT Press, 1998
- *Program≠Program : Constraint Programming and its Relationship to Mathematical Programming*. Irvin J. Lustig et Jean-François Puget. INFORMS, 2001. www.ilog.com/products/optimization/tech/interfaces_informs.pdf
- *Online guide to constraint programming*. Roman Barták. ktiml.mff.cuni.cz/~bartak/constraints/
- *GNU Prolog*. Daniel Diaz. pauillac.inria.fr/~diaz/gnu-prolog/



Plan

1. Introduction
   - Contrainte ? Programmation par contraintes ?
2. Présentation formelle du cadre de travail
3. Programmation logique et programmation par contraintes
4. Exemples de modélisation en PC
5. Programmation par contraintes sur les domaines finis
   - Consistances locales (définitions, implémentations)
   - Contraintes globales
6. Techniques de programmation

Intuitions

Contraintes numériques

Contrainte atomique :

$$x^2 = 2 \tag{1}$$

$\Rightarrow$ le domaine des variables doit être connu :

- x rationnel : pas de solution à (1)
- x réel : deux solutions $\{-\sqrt{2}, \sqrt{2}\}$

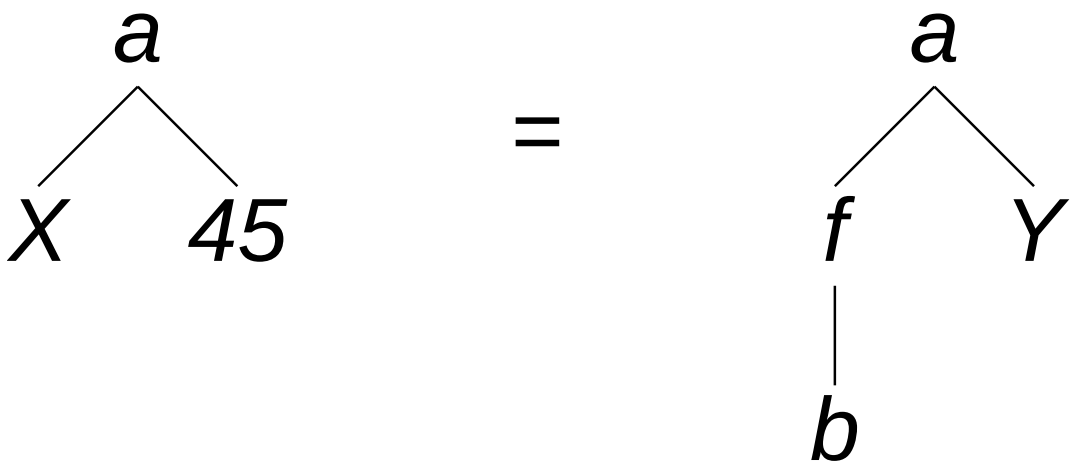
Plus généralement :

$$x^2 - y = 0 \wedge x^2 + y^2 = 1$$

Conjonctions, disjonctions, négations de contraintes atomiques

Contraintes sur les arbres

$$a(X, 45) = a(f(b), Y)$$

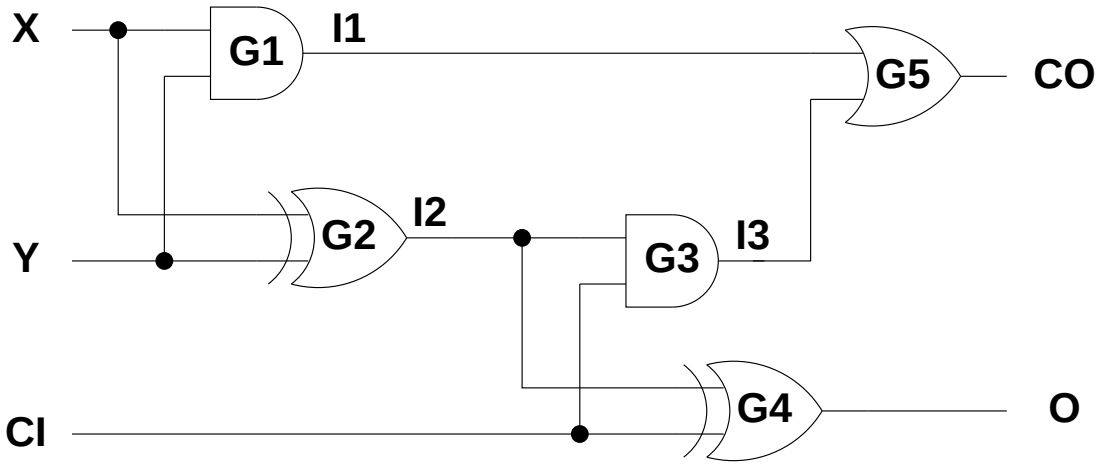


$\Rightarrow X = f(b)$ et $Y = 45$

La relation d'égalité n'est pas directionnelle

Contraintes booléennes

Additionneur :



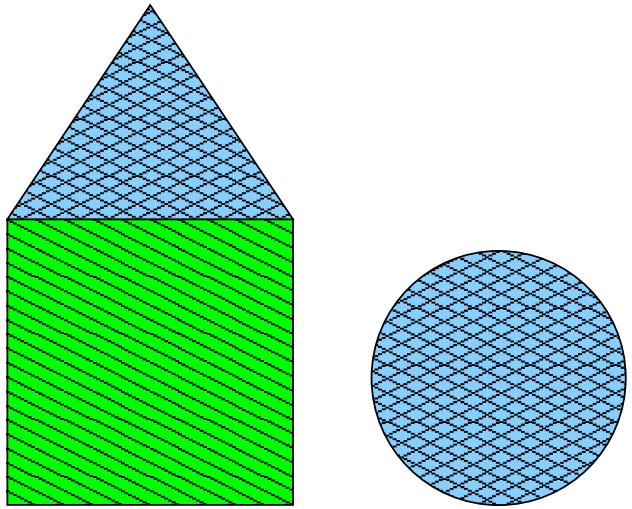
Modélisation :

$$(I1 \iff X \wedge Y) \wedge (I2 \iff X \oplus Y) \wedge (I3 \iff I2 \wedge CI) \wedge (O \iff I2 \oplus CI) \wedge (CO \iff I1 \vee I3)$$

Contraintes symboliques

Monde de blocs :

$$\textit{bleu}(X) \wedge \textit{sur}(X, Y)$$



$\Rightarrow X =$ triangle et $Y =$ rectangle

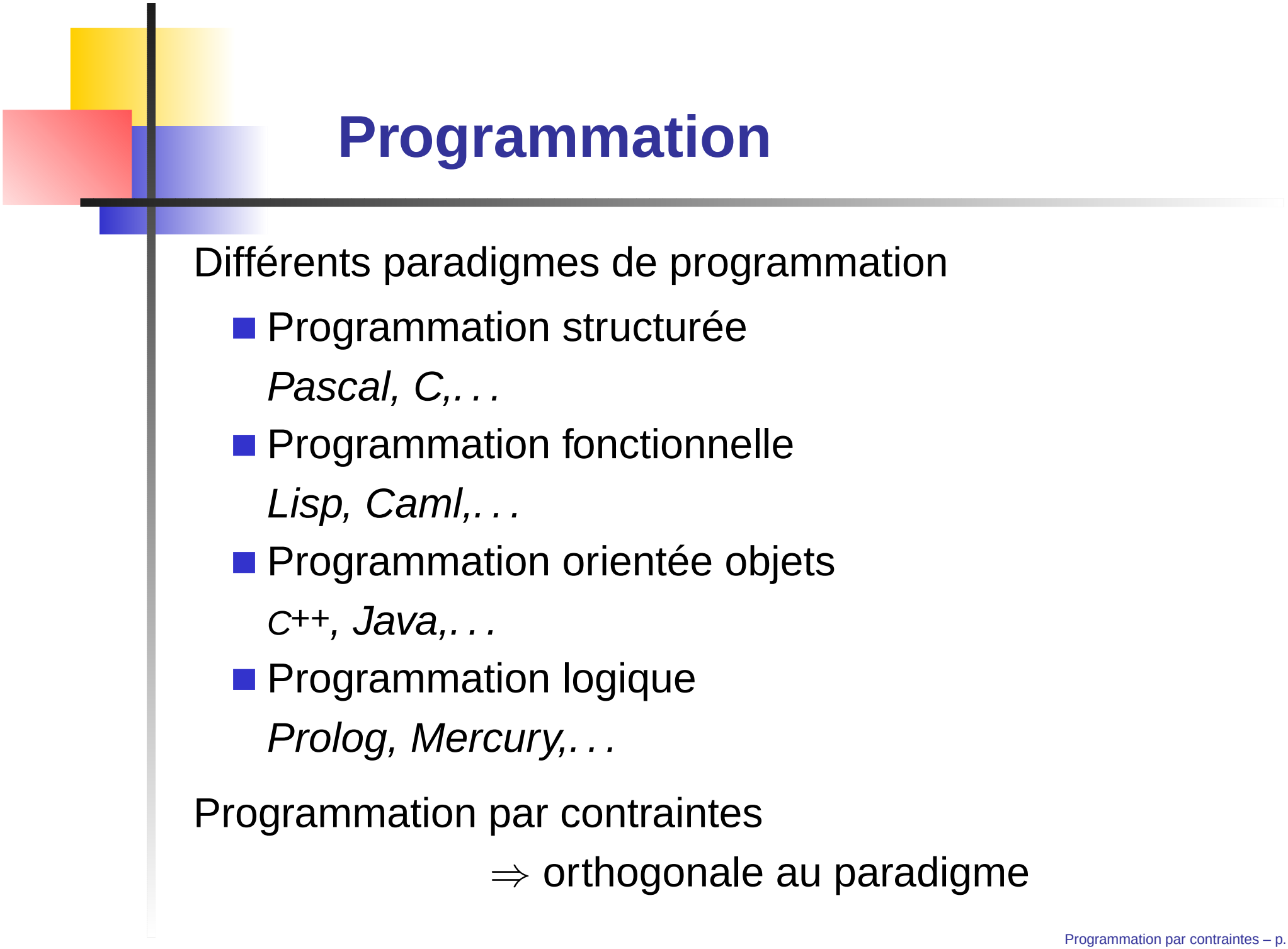
Contraintes « globales »

- *atmost*$(2, [X_1, X_2, X_3, X_4, X_5], 1)$
  Au plus 2 variables parmi $\{X_1, X_2, X_3, X_4, X_5\}$ sont égales à 1

- *alldiff*$([X_1, X_2, X_3, X_4, X_5])$
  Les variables $\{X_1, X_2, X_3, X_4, X_5\}$ ont des valeurs différentes deux à deux

Les contraintes en bref

- Une contrainte est une relation entre des entités
- Une contrainte peut spécifi er de l'information partielle
  « L'âge du capitaine est au moins supérieur à 40 »
- Une contrainte est déclarative (indépendante du processus opérationnel)
- Une contrainte est non-directionnelle (relationnelle) : $x + y = z$: si x et y sont connus, on détermine z ; si x et z sont connus, on détermine y,. . .
- L'ordre de pose des contraintes est indifférent pour la sémantique



Programmation

Différents paradigmes de programmation

- Programmation structurée

  Pascal, C,. . .
- Programmation fonctionnelle

  Lisp, Caml,. . .
- Programmation orientée objets

  C++, Java,. . .
- Programmation logique

  Prolog, Mercury,. . .

Programmation par contraintes

$\Rightarrow$ orthogonale au paradigme

Programmation par contraintes

“Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

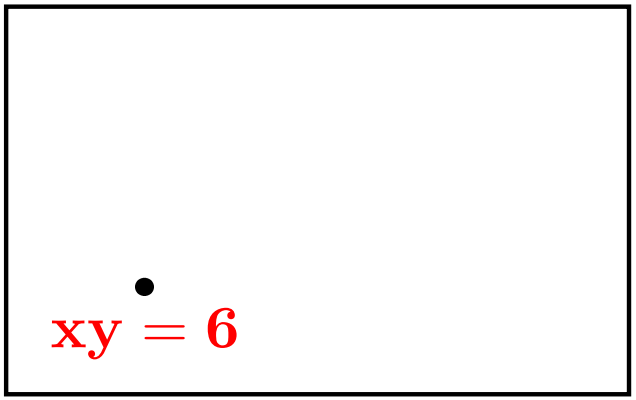
Eugene C. Freuder, CONSTRAINTS, Avril 1997

Programmation par contraintes $=$ architecture à 2 niveaux :

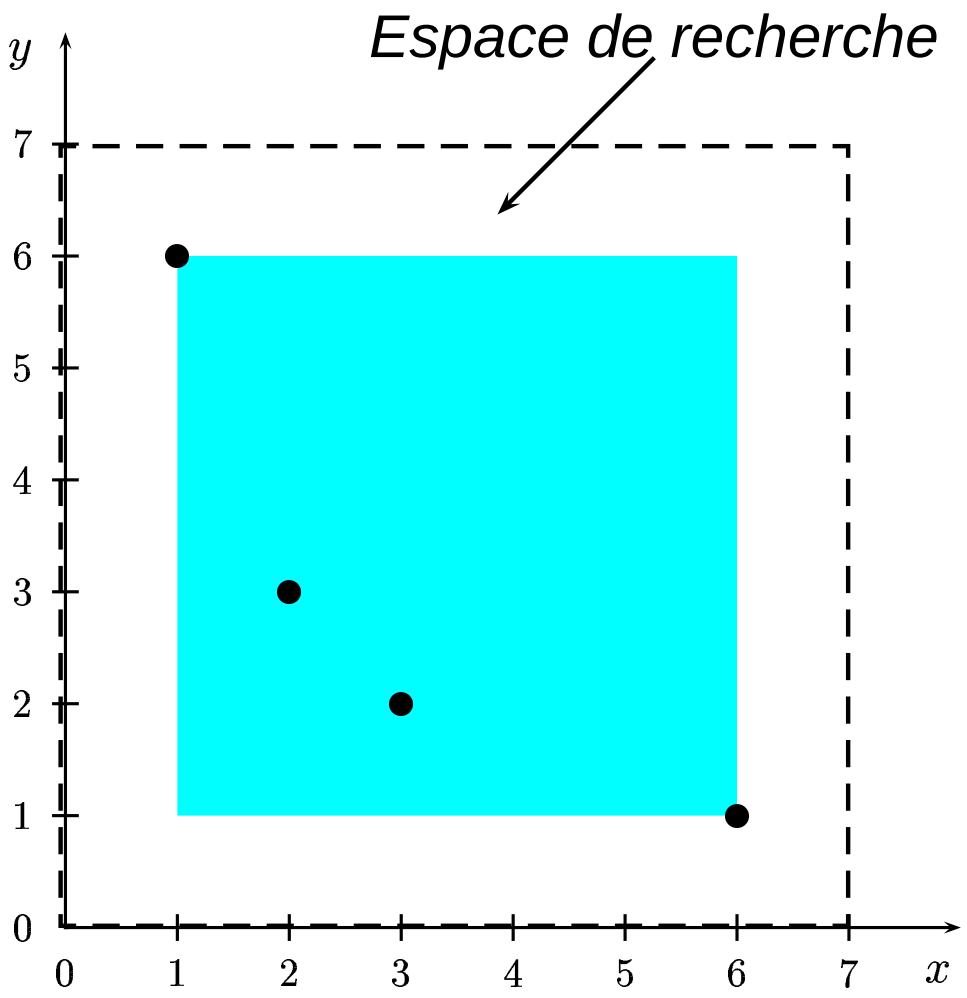
- un composant « langage »
  (indépendamment du paradigme)
  $\Rightarrow$ ajout des contraintes au *store de contraintes*
- un composant « solveur de contraintes »
  Réduction des domaines des variables par considération des contraintes du *store*

Dualité de la PC (1)

```
1 :- x in 0..7,
2    y in 0..7,
3    x*y $= 6,
4    x+y $= 5,
5    x   $< y.
```

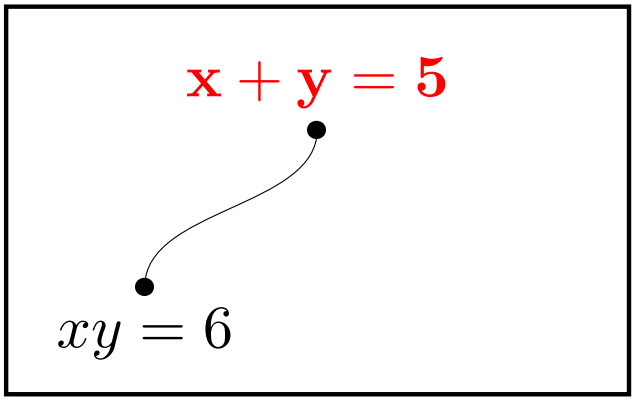


Store

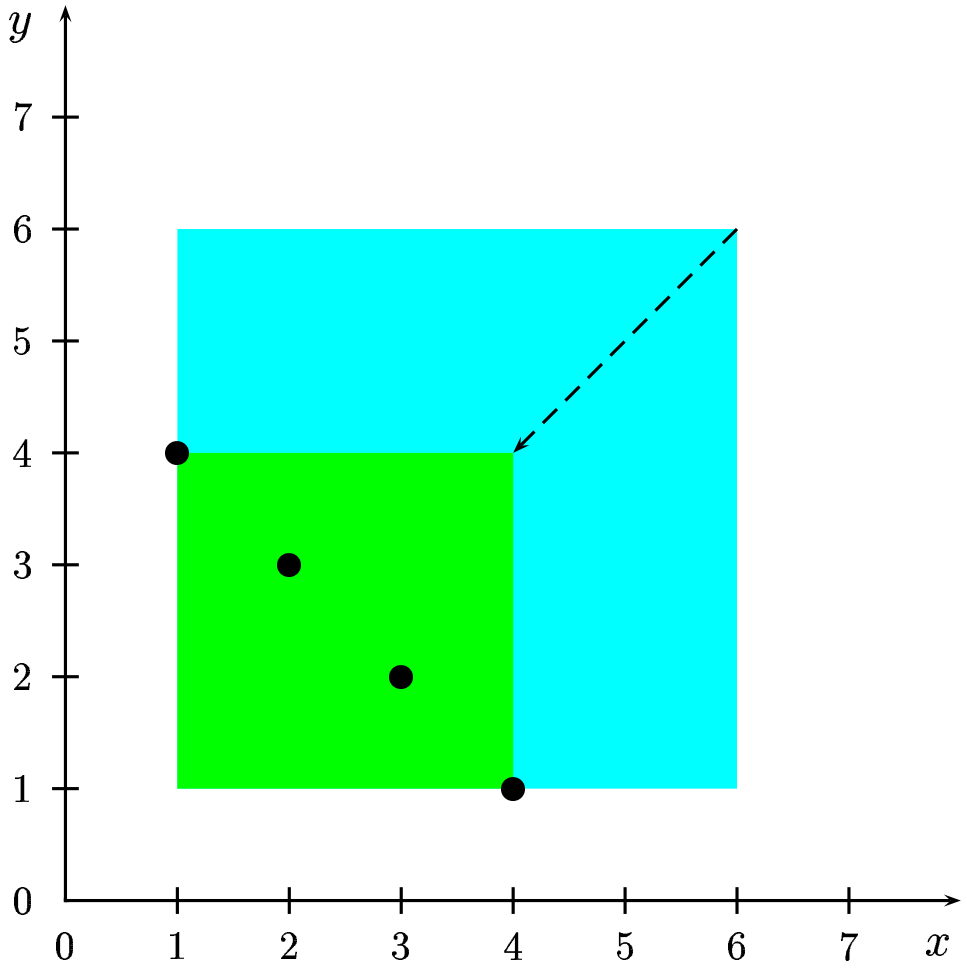


Dualité de la PC (2)

```
1 :- x in 0..7,
2    y in 0..7,
3    x*y $= 6,
4    x+y $= 5,
5    x   $< y.
```

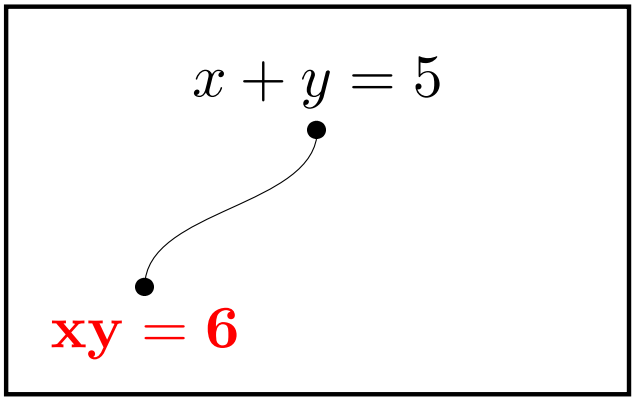


Store

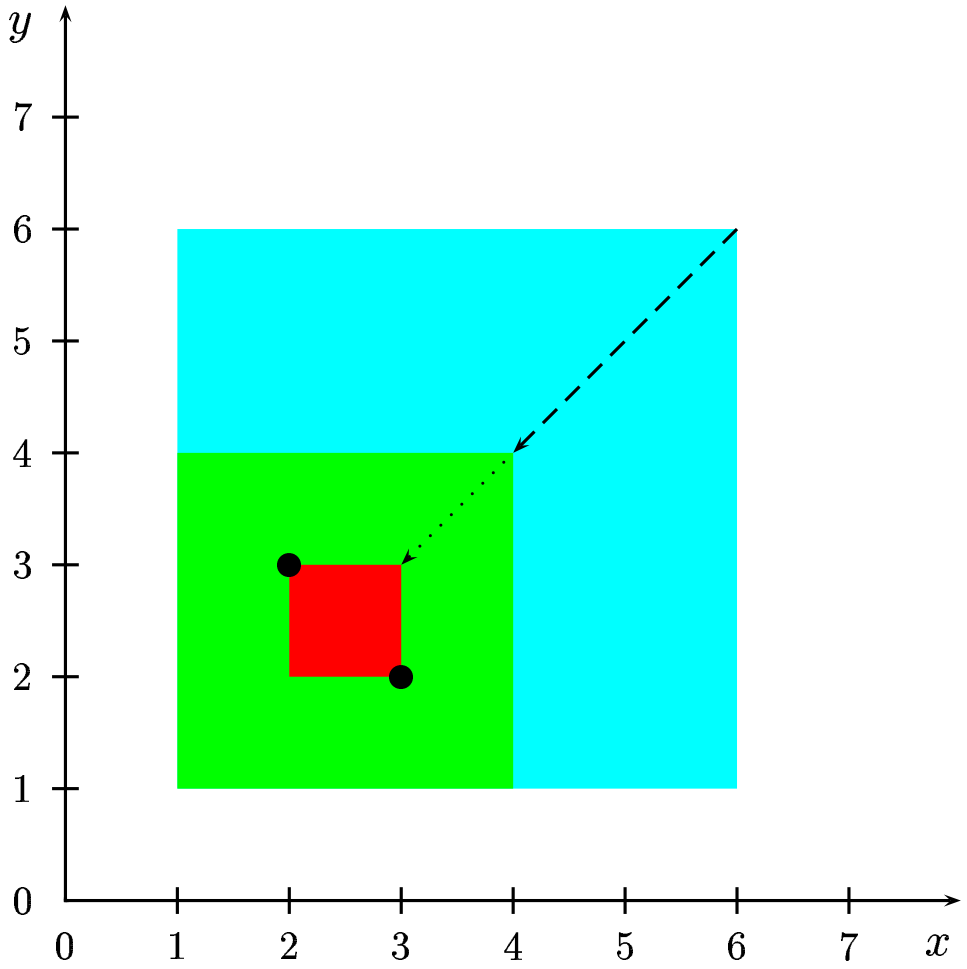


Dualité de la PC (3)

```
1 :- x in 0..7,
2    y in 0..7,
3    x*y $= 6,
4    x+y $= 5,
5    x   $< y.
```

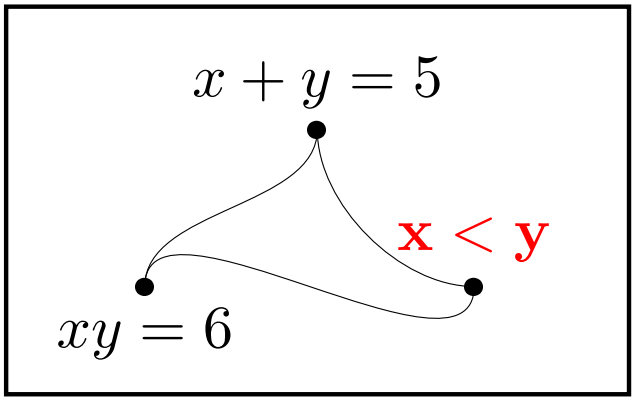


Store

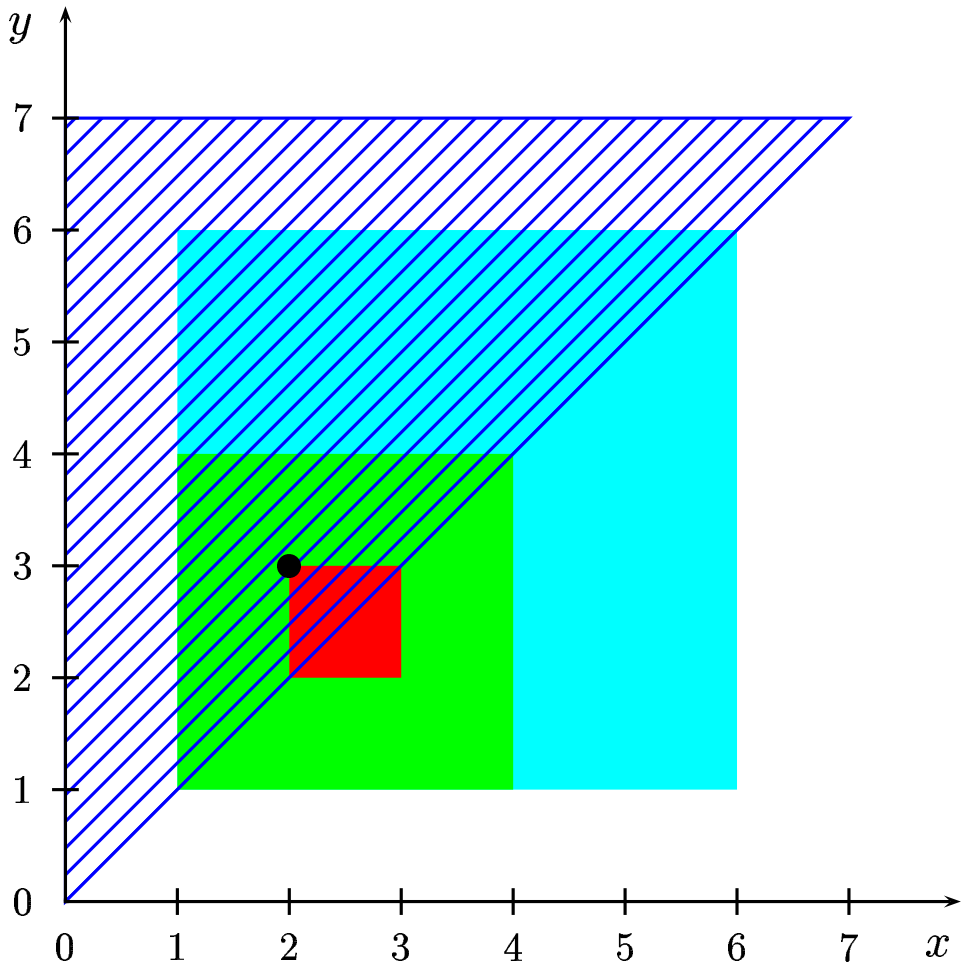


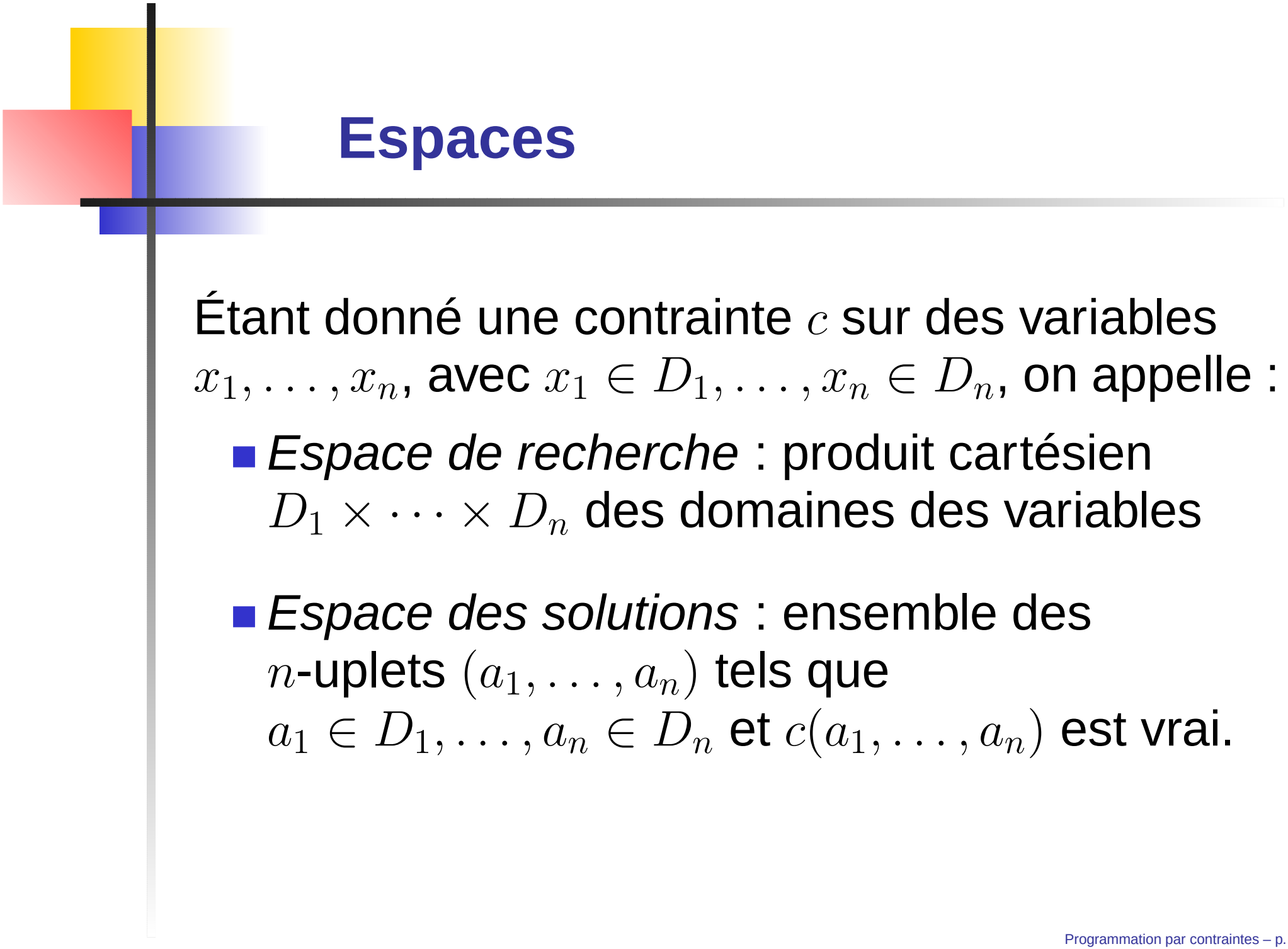
Dualité de la PC (4)

```
1 :- x in 0..7,
2    y in 0..7,
3    x*y $= 6,
4    x+y $= 5,
▶5   x   $< y.
```



Store

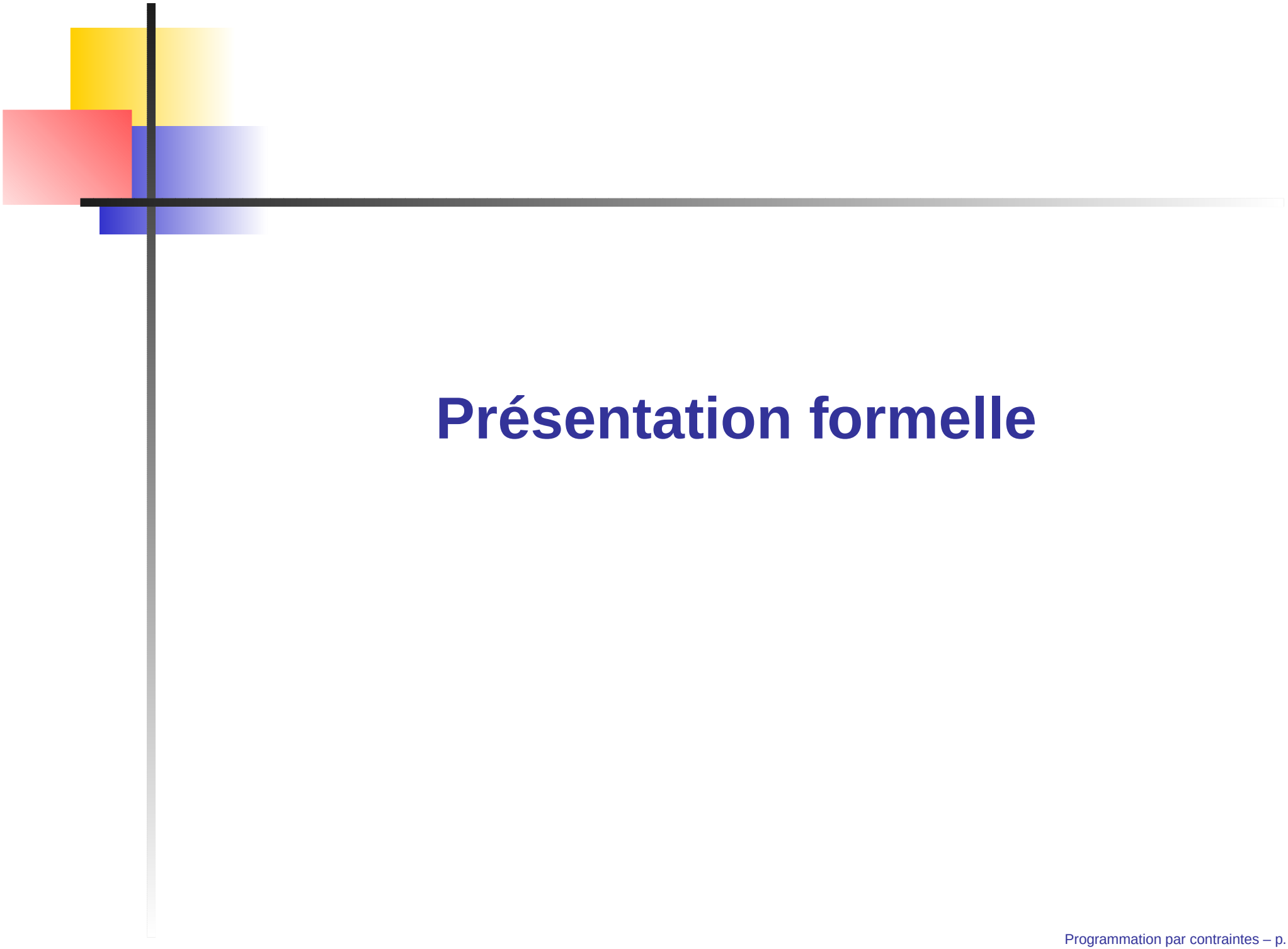




Espaces

Étant donné une contrainte c sur des variables $x_1, \ldots, x_n$, avec $x_1 \in D_1, \ldots, x_n \in D_n$, on appelle :

- *Espace de recherche* : produit cartésien $D_1 \times \cdots \times D_n$ des domaines des variables
- *Espace des solutions* : ensemble des n-uplets $(a_1, \ldots, a_n)$ tels que $a_1 \in D_1, \ldots, a_n \in D_n$ et $c(a_1, \ldots, a_n)$ est vrai.



Présentation formelle

Langage du premier ordre

Langage du premier ordre $\mathcal{L}$:

- Ensemble infi ni dénombrable de symboles de variables :

$$\mathcal{V} = \{x, y, z, X, Y, Z, \ldots\}$$

- Connecteurs $\wedge, \vee, \neg, \Rightarrow$
- Ensemble de symboles de fonctions avec arité $\mathcal{F}$
- Ensemble de constantes $\mathcal{A}$ (aussi fonctions d'arité nulles)
- Ensemble de symboles de relations avec arité $\mathcal{R}$

Terme

Défi nition récursive de l'ensemble $\mathcal{T}_{\mathcal{L}}$ des termes sur un langage du premier ordre $\mathcal{L}$:

- Une variable est un terme
- Une constante est un terme
- Si $f \in \mathcal{F}$ est d'arité k et que $t_1, \ldots, t_k$ sont des termes, alors $f(t_1, \ldots, t_k)$ est un terme

Exemple de termes :

$$x, \quad f(x_1, x_2), \quad g(f(a, x), b, h(y))$$

Formule

Défi nition récursive de l'ensemble $\mathcal{O}_{\mathcal{L}}$ des formules sur un langage du premier ordre $\mathcal{L}$:

- Si $R \in \mathcal{R}$ est d'arité k, et que $t_1, \ldots, t_k$ sont des termes sur $\mathcal{L}$, alors $R(t_1, \ldots, t_k)$ est une formule (atomique) de $\mathcal{L}$
- Si R_1 et R_2 sont des formules, alors :
  - $\neg\, R_1, R_1 \Rightarrow R_2, R_1 \wedge R_2, R_1 \vee R_2$

  sont des formules

Exemple de formules :

$$R_1(x), \quad R_2(f(x_1, x_2), a) \wedge \neg R_3(f(a, x), b, h(y))$$

Structure (réalisation de $\mathcal{L}$)

Une structure $\Sigma = \langle \mathcal{E}, \overline{\mathcal{F}}, \overline{\mathcal{R}} \rangle$, où $\mathcal{E}$ est un ensemble non vide appelé *ensemble de base*, est une *réalisation d'un langage* $\mathcal{L}$ lorsque :

- pour tout $a \in \mathcal{A}$, on peut associer $\overline{a} \in \mathcal{E}$
- pour tout $f \in \mathcal{F}$ d'arité k, on peut associer $\overline{f} : \mathcal{E}^k \rightarrow \mathcal{E}$ (*interprétation de* f)
- pour tout $r \in \mathcal{R}$ d'arité k, on peut associer $\overline{r} \subseteq \mathcal{E}^k$ (*interprétation de* r)

En pratique : identifi cation du langage à la structure dans laquelle sont interprétées les contraintes

Exemples de structures

- Structure des équations linéaires réelles :

$$\Sigma = \langle \mathbb{R}, \{+, -\}, \{=\} \rangle$$

- Structure des équations et inéquations polynômiales sur les entiers naturels :

$$\Sigma = \langle \mathbb{N}, \{+, -, \times\}, \{=, \leqslant\} \rangle$$

$\Rightarrow$ une *contrainte* est une formule sur $\mathcal{L}$ interprétée dans une réalisation Σ de $\mathcal{L}$

Notion d'interprétation

Soient k variables $x_1, \ldots, x_n$ et k éléments $e_1, \ldots, e_n$ de $\mathcal{E}$.

L'*interprétation $\overline{t}$ d'un terme t pour une* valuation $\theta = \{x_1 \mapsto e_1, \ldots, x_k \mapsto e_k\}$ est un élément de $\mathcal{E}$ défini par :

- si $t = a$, alors $\theta t = \overline{a}$
- si $t = x_i$, alors $\theta t = e_i$
- si $t = f(t_1, \ldots, t_n)$, alors $\theta t = \overline{f}(\theta t_1, \ldots, \theta t_n)$

L'*interprétation d'une formule c pour la valuation θ* est un booléen défini par :

- si $c \stackrel{\text{def}}{=} r(t_1, \ldots, t_n)$ alors θc est *vrai* (c est *satisfaite*) si $(\theta t_1, \ldots, \theta t_n) \in \overline{r}$
- si $c \stackrel{\text{def}}{=} c_1 \wedge c_2$, alors θc est *vrai* si θc_1 est *vrai* et θc_2 est *vrai*
- ... (on définit de même pour les autres connecteurs)

Exemple d'interprétation

- Soit le langage $\mathcal{L} = \langle \{a, b, d\}, \{\cdot_{/2}, \vee_{/2}\}, \{=\} \rangle$
- Soit la structure $\Sigma = \langle \mathbb{N}, \{\times, +\}, \{\} \rangle$ défi nie comme réalisation de $\mathcal{L}$ par :

$$a \rightsquigarrow 0, b \rightsquigarrow 1, d \rightsquigarrow 2, \cdot_{/2} \rightsquigarrow \times, \vee_{/2} \rightsquigarrow +$$

- Soient :

$$c \stackrel{\text{def}}{=} x \cdot (y \vee a \vee b) \cdot d = x \cdot d \qquad \theta = \{x \mapsto 2, y \mapsto 0\}$$
$$c' \stackrel{\text{def}}{=} y \cdot x \vee x \cdot x \vee b = x \vee y \qquad \theta' = \{x \mapsto 0, y \mapsto 1\}$$

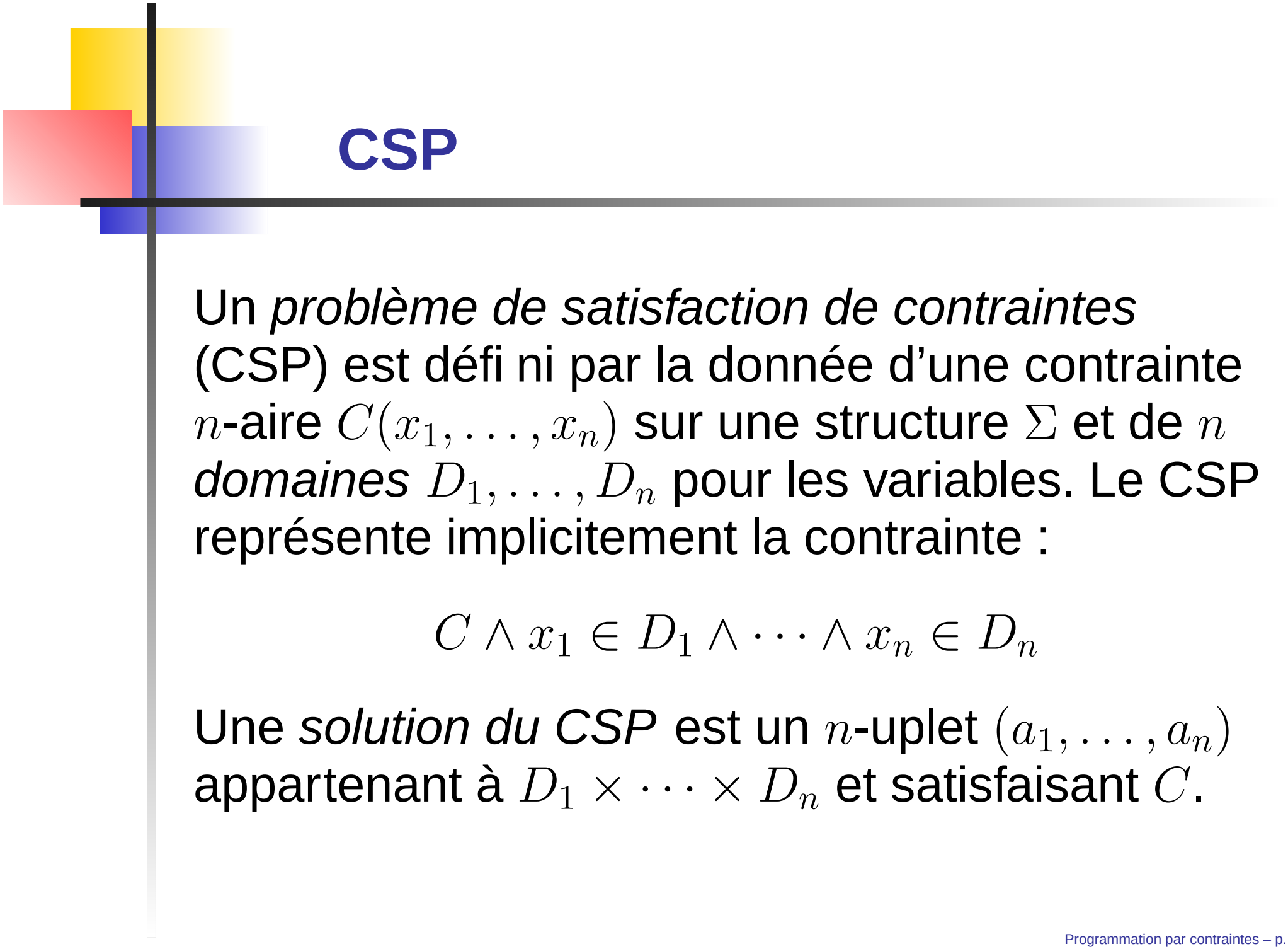
- On a : $\theta c \equiv$ vrai, $\theta c' \equiv$ faux, $\theta' c \equiv$ vrai, $\theta' c' \equiv$ vrai

Solveur de contraintes

Étant donné une contrainte c, on peut étudier les problèmes suivants :

- *satisfaction* : la contrainte c est-elle satisfiable ?
  (Existe t'il une valuation des variables de c telle que c est vraie)
- *solution* : si c est satisfiable, exhiber une ou plusieurs solutions
- *optimisation* : exhiber une solution optimale (concept à définir)
- *simplification* : transformer c en une *contrainte équivalente* (i.e. avec le même espace de solutions)

On ne s'intéressera ici qu'aux deux premiers problèmes



CSP

Un *problème de satisfaction de contraintes* (CSP) est défi ni par la donnée d'une contrainte n-aire $C(x_1, \ldots, x_n)$ sur une structure Σ et de n *domaines* $D_1, \ldots, D_n$ pour les variables. Le CSP représente implicitement la contrainte :

$$C \wedge x_1 \in D_1 \wedge \cdots \wedge x_n \in D_n$$

Une *solution du CSP* est un n-uplet $(a_1, \ldots, a_n)$ appartenant à $D_1 \times \cdots \times D_n$ et satisfaisant C.

Résolution de CSP

Résolution triviale d'un point de vue théorique : il suffi t d'explorer systématiquement l'espace de recherche !

- *Generate and test* : générer une instantiation de toutes les variables, puis tester si les contraintes sont satisfaites
- *Backtracking* : génération incrémentale d'instantiation. On teste la satisfaction de toute contrainte dont les variables sont complètement instanciées. En cas d'échec, on défait l'instantiation la plus récente.
  - *thrashing* : échec répété dû aux mêmes raisons
  - les valeurs en conflit ne sont pas conservées lors du backtracking
- *Consistances locales* : on retire des domaines les valeurs qui ne satisfont pas chaque contrainte

Propriétés des solveurs

- Un solveur est dit *complet* s'il peut toujours répondre par oui ou par non au problème de satisfaction d'une contrainte
- Un solveur est dit *correct* s'il ne calcule que des solutions
- Un solveur est dit *fiable* s'il calcule toutes les solutions d'un problème

Sur les réels : incomplétude et incorrection sont aisées

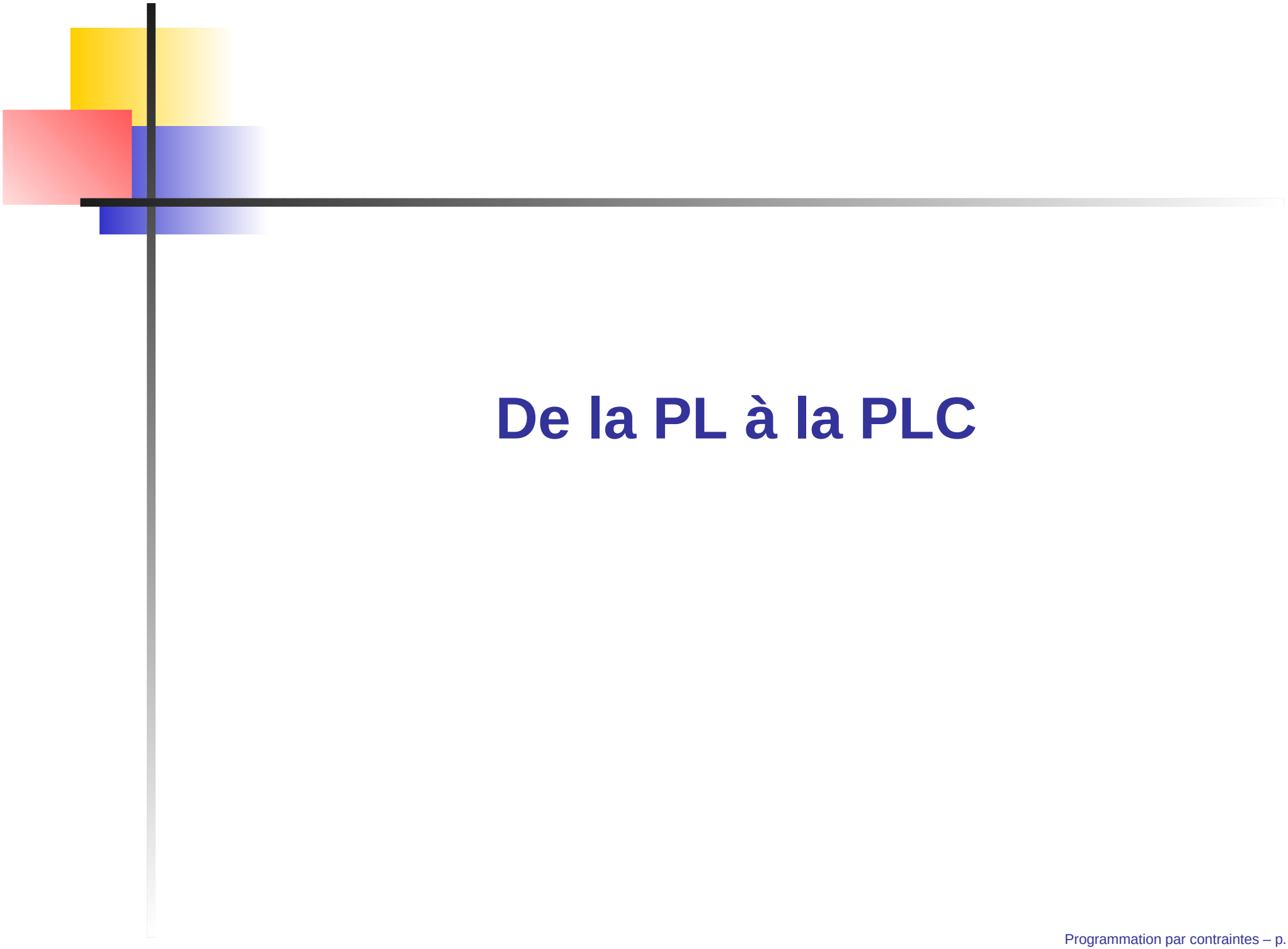
Systèmes de PC (1)

- **CLP($\Re$)**. Système basé sur Prolog. Résolution de contraintes linéaires sur les réels (équations et inéquations). Utilisation de l'élimination de Gauss et du Simplexe. Contraintes non-linéaires mises en attente. Système incorrect et incomplet
- **Prolog IV**. Système basé sur Prolog. Résolution de contraintes (non-)linéaires sur les rationnels et les réels, les contraintes sur les arbres (équations, diséquations) et sur les chaînes de caractères (équations). Système fiable et complet (suivant algos)
- **CHIP**. Système basé sur Prolog. Contraintes linéaires sur les rationnels, contraintes booléennes.

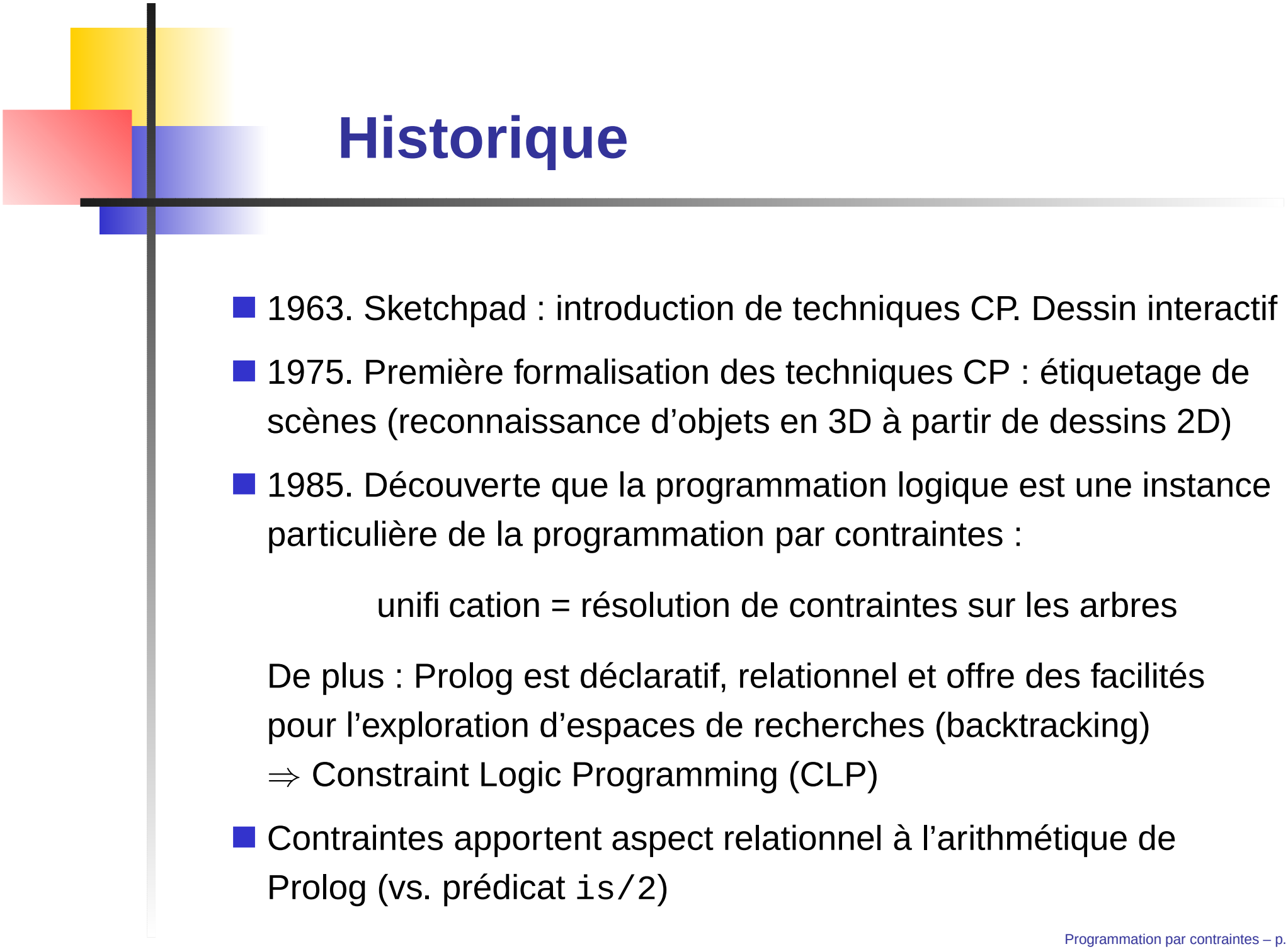
Systèmes de PC (2)

- **ECLiPSe**. Système basé sur Prolog. Contraintes sur les domaines finis, contraintes linéaires sur les réels, *Constraint Handling Rules*
- **Ilog Solver**. Librairie C++. Contraintes sur les domaines finis et les réels (fiable et complet)
- **Numerica**. Système avec son propre langage. Contraintes (non-)linéaires sur les réels. Fiable et complet
- **GNU Prolog**. Prolog avec une extension pour la résolution de contraintes sur les domaines finis.

Dans la suite : on ne s'intéresse qu'aux contraintes sur les domaines finis. Utilisation de GNU Prolog.



De la PL à la PLC



Historique

- 1963. Sketchpad : introduction de techniques CP. Dessin interactif
- 1975. Première formalisation des techniques CP : étiquetage de scènes (reconnaissance d'objets en 3D à partir de dessins 2D)
- 1985. Découverte que la programmation logique est une instance particulière de la programmation par contraintes :

  unifi cation = résolution de contraintes sur les arbres

  De plus : Prolog est déclaratif, relationnel et offre des facilités pour l'exploration d'espaces de recherches (backtracking)
  $\Rightarrow$ Constraint Logic Programming (CLP)
- Contraintes apportent aspect relationnel à l'arithmétique de Prolog (vs. prédicat `is/2`)

Rappels de syntaxe Prolog (1)

- **Variables** : commencent par une majuscule ou le blanc souligné
  `X, _e34, Variable3`
- **Constantes** : commencent par une minuscule
  `a, 'caracteres quotes', pi`
- **Structure** : `date(dimanche,X,anne(1999))`
- **Terme** : variable, constante ou structure (= structures de données du programme)
- **Atome** : expression de la forme `p(t1,...,tn)` où `p` est un **symbole de prédicat** et les `ti` sont des termes
- **Fait** : expression de la forme `p(t1,...,tn).`
  `pere(jean,X)`

Rappels de syntaxe Prolog (1)

- **Règle** : expression de la forme :
  p(t,. . .,t) :- p(t,. . .,t), . . ., p(t,. . .,t).
  Exemple :
  `menu(E,P,D) :- entree(E), plat_resistance(P), dessert(D).`
- **Tête** : partie gauche de la règle
- **Corps** : partie droite de la règle
- **Clause** : règle ou fait
- **Requête** : clause sans tête

Rappels de syntaxe Prolog (1)

- **Prédicat** : ensemble des clauses dont la tête a le même nom et la même arité

  Exemple :

```
chien(rex).
chien(X) :- animal(X), aboie(X).
chien(X) :- porte_collier(X), jappe(X).
```

Dans un corps de règle, la virgule correspond à la conjonction et le point-virgule à la disjonction

Le fait indique une vérité établie : « rex est un chien »

Sens de la deuxième clause : « X est un chien si X est un animal et si X aboie ».

Chaque clause d'un prédicat constitue une alternative (« ou » implicite)

Le nom d'une variable est local à la clause.

Exécution de Prolog

Vision procédurale des clauses, où l'appel avec passage de paramètres est remplacé par l'appel avec unifi cation des variables.
Unifi cation de deux termes : recherche de la substitution minimale pour les variables rendant les deux termes égaux.
Pas de description de l'aspect opérationnel.
Ordre de considération des clauses et des buts est arbitraire (en théorie).
Utilisation du backtracking pour explorer toutes les alternatives

LP versus CLP (1)

- Approche Prolog :

```
p(X,Y,Z):- Z is X+Y.
:- p(3,4,Z).
Z = 7
:- p(X,4,7).
INSTANTIATION ERROR
```

- Approche CLP :

```
p(X,Y,Z):- Z #= X+Y.
:- p(3,4,Z).
Z = 7
:- p(X,4,7).
X = 3.
```

LP versus CLP (2)

Arithmétique pas relationnelle en Prolog $\Rightarrow$ *generate and test* seule approche possible :

```
solution(X,Y,Z):- p(X), p(Y), p(Z), test(X,Y,Z).
p(11). p(3). p(7). p(16). p(15). p(14).
test(X,Y,Z):- Y is X+1, Z is Y+1.
:- solution(X,Y,Z).
```

458 pas pour la première solution

LP versus CLP (3)

Arithmétique relationnelle en PLC $\Rightarrow$ *constrain and generate* possible :

```
solution(X,Y,Z):- test(X,Y,Z), p(X), p(Y), p(Z).
p(11). p(3). p(7). p(16). p(15). p(14).
test(X,Y,Z):- Y #= X+1, Z #= Y+1.
:- solution(X,Y,Z).
```

11 pas pour la première solution