LIFE Su Doku

Hassan Aït-Kaci

ILOG Research Projects
IBM Canada Ltd.

ILOG Products and Solutions

LIFE **Su Doku—Outline**

- Overview
- A quick look back on *LIFE*
- How is *LIFE* (*all* that) *different*?
- Purely declarative *Su Doku*
- It's *all different* using graphs!
- *LIFE* bonus: a declarative *Su Doku* GUI
- Epilogue

ILOG Products and Solutions

Overview

ILOG Products and Solutions

Overview

Life is *"trying things to see if they work. . . "*

RAY BRADBURY

$\mathcal{LIFE}$ stands for: $\mathcal{L}$*ogic*
$\mathcal{I}$*nheritance*
$\mathcal{F}$*unctions*
$\mathcal{E}$*quations*

$\mathcal{LIFE}$ may be viewed as a $\mathcal{CLP}$ language:

Logic Programming over (logically and functionally) constrained order-sorted labeled graphs

Overview—ctd.

N.B.: $\mathcal{LIFE}$ does not have "`alldiff`" as a built-in constraint!

However... $\mathcal{LIFE}$'s features enable a surprisingly efficient "`alldiff`" purely declaratively thanks to:

- $\mathcal{LIFE}$'s built-in constrained data-structure—the ψ-term
- $\mathcal{LIFE}$'s control strategy—(constraint) *residuation*

Residuation: Functional evaluation that proceeds as far as possible, *suspending upon unbound variables* and *resuming as they get further instantiated*

ILOG Products and Solutions
IBM®

LIFE Su Doku—Outline

A quick look back on LIFE

IBM® | ILOG Products and Solutions

A quick look back on $\mathcal{LIFE}$

> Life can only be understood looking backwards
> but it must be lived forwards.
>
> SØREN KIERKEGAARD

- $\mathcal{LIFE}$ is a $\mathcal{CLP}$ language that may be loosely defined as "Prolog over ψ-terms"
- A ψ***-term*** is a rooted graph whose nodes are typed with *sorts*, and whose arcs are labelled by feature symbols
- A ψ-term's syntax extends that of a Prolog term:
  - `f(a,X,g(X))` — same as `f(3=>g(1=>X), 1=>a, 2=>X)`
  - `person(name => "bozo", dob => date(year => 1980))`
  - `add(X,Y,result => X+Y)`
  - `X:person(spouse => person(spouse => X))`

- A ψ-term has ***no arity***—can have ***no or many features***

- Unifying the ψ-terms `f(a,3=>c)` and `f(a,b)` succeeds and results in `f(a,b,c)`.

- Unifying the ψ-term:

  ```
  person(P, dob => date(month => may))
  ```

  with the ψ-term:

  ```
  person(dob => date(year => 1980)),
  ```

  succeeds with the ψ-term:

  ```
  person(P, dob => date(month => may, year => 1980)).
  ```


ILOG Products and Solutions
IBM

A quick look back on $\mathcal{LIFE}$

- ***Everything*** in $\mathcal{LIFE}$ is a ψ-term
- $\mathcal{LIFE}$'s ***predicates*** are:
  - defined by ***Horn rules*** over ψ-terms
  - invoked using ***unification***
  - ***non-deterministic***: they use ***top-down left-right back-tracking*** (*i.e.*, like Prolog)
- $\mathcal{LIFE}$'s ***functions*** are:
  - defined by ***rewrite rules*** over ψ-terms
  - invoked using ***matching***
  - ***deterministic***: they use ***top-down committed choice*** (*i.e.*, functions do not backtrack)

A quick look back on $\mathcal{LIFE}$

- $\mathcal{LIFE}$'s logical variables are typed—*e.g.*, `X:int`
- No difference between type and value—all are sorts
- Sorts are partialy ordered in a sort hierarchy
- The ***top*** sort is `@`; the ***bottom*** sort is `{}`
- If we declare: `apple <| fruit. apple <| food.` then, the query: `X = food, X = fruit?` yields: `X = apple`
- If we also declare: `banana <| fruit. banana <| food.` then, the query backtracks to yield: `X = banana`
- ***Disjunctive sort***: `X:{ breakfast ; lunch ; dinner }`

ILOG Products and Solutions
IBM

Predicate resolution and function evaluation cooperate by residuation

- In query:

  `X = Y+1, Y = 2?`

  equation:

  `X = Y+1`

  is a *residual* constraint (or *residuation*)

- Executing `Y = 2?` ***awakens*** the residuation
- Resulting in fully resolved binding: `X = 3, Y = 2`

Feature projection extracts subterms

- Dyadic function `./2`:
  - *1st arg:* a ψ-term
  - *2nd arg:* a feature—*i.e.*, position or symbol
  - *returns:* the subterm rooted at specified feature

  That is:

  `T.f = T'` *iff* `T = s(..., f => T', ...)`

- ***N.B.:*** Feature projection residuates whenever its second argument is not ground—*e.g.*, `foo(bar => baz).X` with `X` unbound

IBM. | ILOG Products and Solutions

Feature projection may have side effects! . . .

- If a ψ-term `T` does ***not*** have feature `f`, then `T.f` ***creates the feature*** `f` ***for*** `T`

  That is, the query:

  ```
  X = foo(bar => baz), X.boo = fuz?
  ```

  yields the binding:

  ```
  X = foo(bar => baz, boo => fuz)
  ```

- ***N.B.:*** ***All*** (binding and feature creation) ***side-effects are undone upon backtracking***

ILOG Products and Solutions

How is $\mathcal{LIFE}$ (all that) different?

ILOG Products and Solutions

Life is the sum of all your choices.

ALBERT CAMUS

At first, $\mathcal{LIFE}$ feels like Prolog:

Same syntax for Horn clauses (‘`:-/2`’, ‘`,/2`’, ‘`;/2`’), logical variables, lists, . . . ; *e.g.*,

```
append([],L,L).
append([H|T],L,[H|R]) :- append(T,L,R).
```

can be used exactly as in Prolog!

But, $\mathcal{LIFE}$ also differs from Prolog:

Arity is not constrained; *e.g.*,

```
A = foo(a => 1, b => 2),
B = foo(b => X, c => 3),
A = B ?
```

succeeds, resulting in the solved form:

```
A = foo(a => 1, b => X, c => 3),
B = A,
X = 2.
```


ILOG Products and Solutions
IBM

LIFE's ***user-defined functions*** are specified as ***rewrite rules*** using infix operator '`->/2`':

```
length([]) -> 0.
length([_|T]) -> 1 + length(T).
```

and use them in relational clauses:

```
has_even_length(L:list) :- length(L) mod 2 = 0.
```

Then,

```
has_even_length([a,b])?
```

succeeds as expected.

Similarly:

```
has_even_length([a,L:list])?
```

creates the residuation:

```
(1 + length(L:list)) mod 2 = 0?
```

with incomplete solution:

```
L = list~
```

$\mathcal{LIFE}$ indicates an incomplete solution with as many *tildas* ("~") as it has ***pending residuations***

How is $\mathcal{LIFE}$ (*all* that) *different*?—ctd.

metaprogramming allows reasoning about features using feature projection

For instance, if:

```
A = foo(a => 1, b => 2, c => 2)
```

then:

```
X = { a ; b ; c }, A.X = 2?
```

succeeds first with: `X = b`

then, upon backtracking, with: `X = c`

ILOG Products and Solutions

Purely declarative Su Doku

ILOG Products and Solutions
IBM

Purely declarative *Su Doku*

The art of life is the art of avoiding pain.

THOMAS JEFFERSON

```
% Specify the Su Doku grid:
sudoku(@(@(@(X11,...,X19), ..., @(X91,...,X99)))
:- % The rows constraints:
   alldiff(X11,...,X19), ..., alldiff(X91,...,X99),
   % The columns constraints:
   alldiff(X11,...,X91), ..., alldiff(X19,...,X99),
   % The square constraints:
   alldiff(X11,...,X33), ..., alldiff(X77,...,X99).
```

Purely declarative *Su Doku*

```
% Specify the cell labels:
labels(@(@(X11,...,X19), ..., @(X91,...,X99)))
:- X11 = label, ..., X19 = label,
   ...,
   X91 = label, ..., X99 = label.

% Generate the cell labels:
label -> { 1 ; ... ; 9 }.

% The main predicate:
sudoku_solver(G) :- sudoku(G), labels(G).
```

It's all different using graphs!

ILOG Products and Solutions

It's *all different* using graphs!

If A equals success, then the formula is: A = X + Y + Z, where X is work, Y is play, and Z is keep your mouth shut.

ALBERT EINSTEIN

```
alldiff(X1,X2,X3)
:- assign(A,X1,1), assign(A,X2,2), assign(A,X3,3).
```

where:

- `A` denotes the ***global assignment***
- `X` denotes the ***constrained variable***
- `I` denotes the ***assignment's unique id***

```
assign(A,X,I) :- A.X = I.
```


ILOG Products and Solutions

It's *all different* using graphs!

For example:

```
show(X1,X2,X3)
:- alldiff(X1,X2,X3),
   X1 =  a ; b ,    % domain of X1
   X2 =  b ; c ,    % domain of X2
   X3 =  a ; d .    % domain of X3
```

Then, invoking `show(X1,X2,X3)?` yields, successively:

```
X1 = a, X2 = b, X3 = d.
X1 = a, X2 = c, X3 = d.
X1 = b, X2 = c, X3 = a.
X1 = b, X2 = c, X3 = d.
```


ILOG Products and Solutions IBM

Purely declarative *Su Doku*

Test and generate:

```
sudoku_solver(G) :- sudoku(G), labels(G).
```

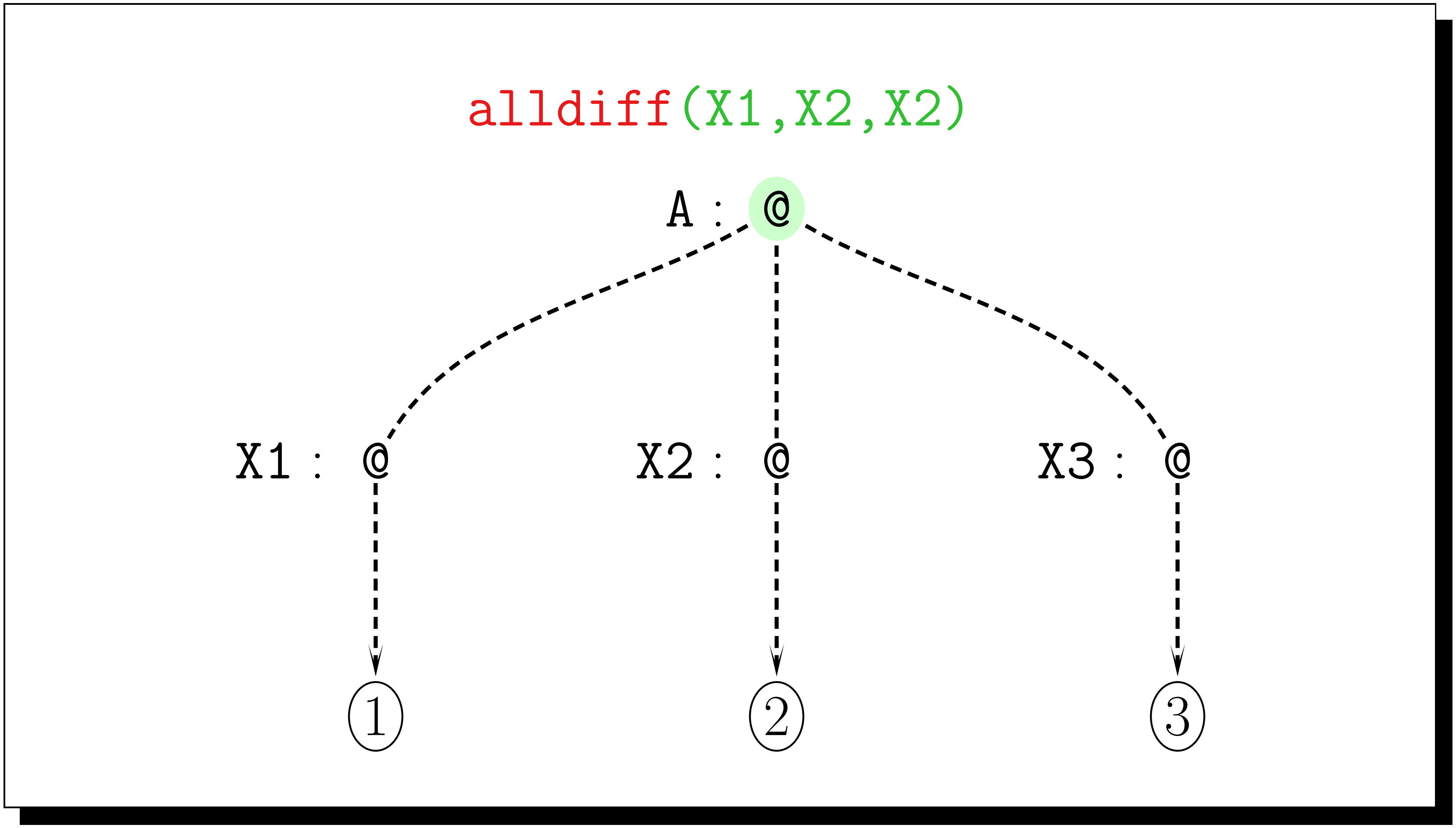
VS.

Generate and test:

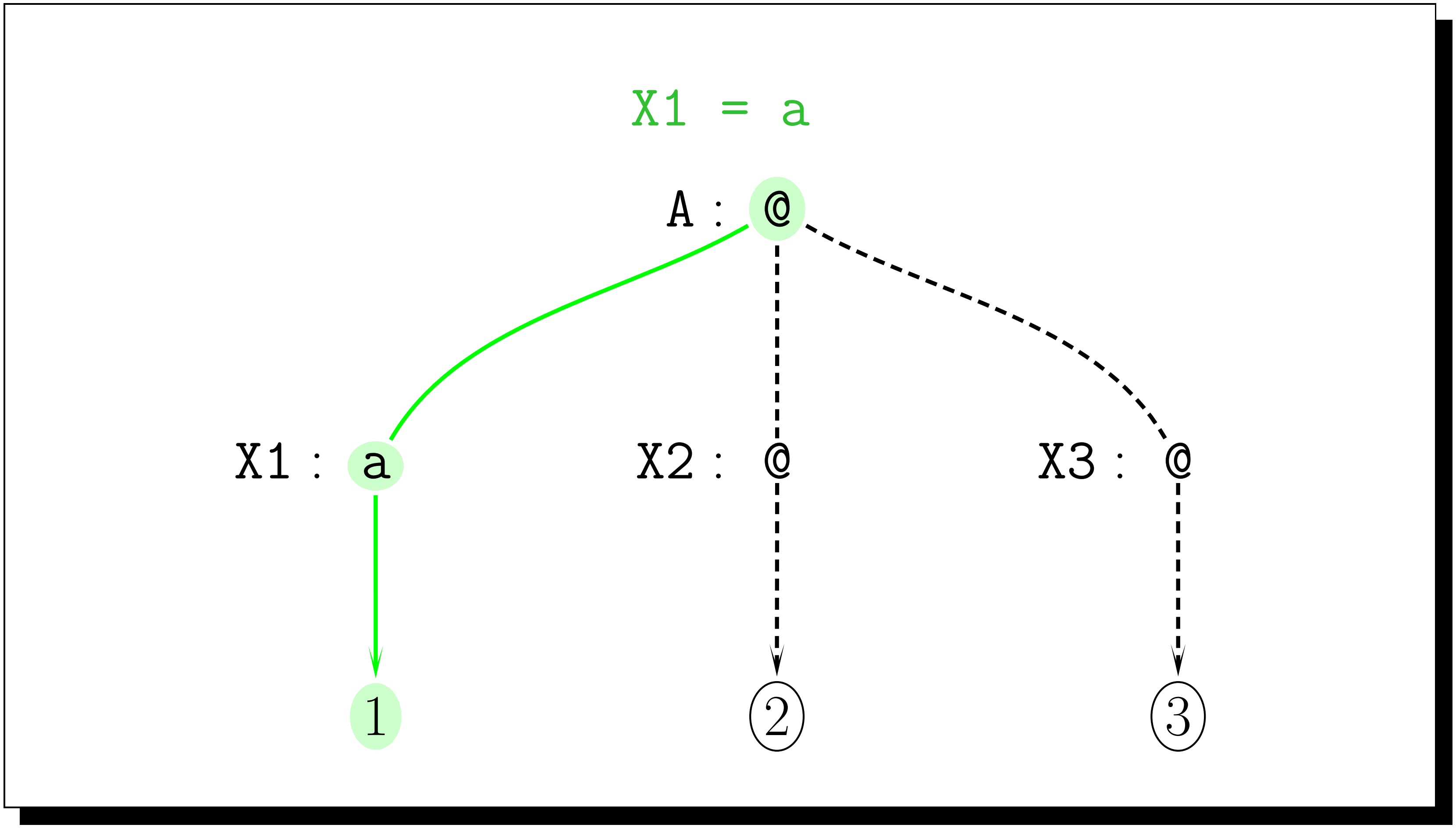
```
bad_sudoku_solver(G) :- labels(G), sudoku(G).
```


ILOG Products and Solutions

It's *all different* using graphs!

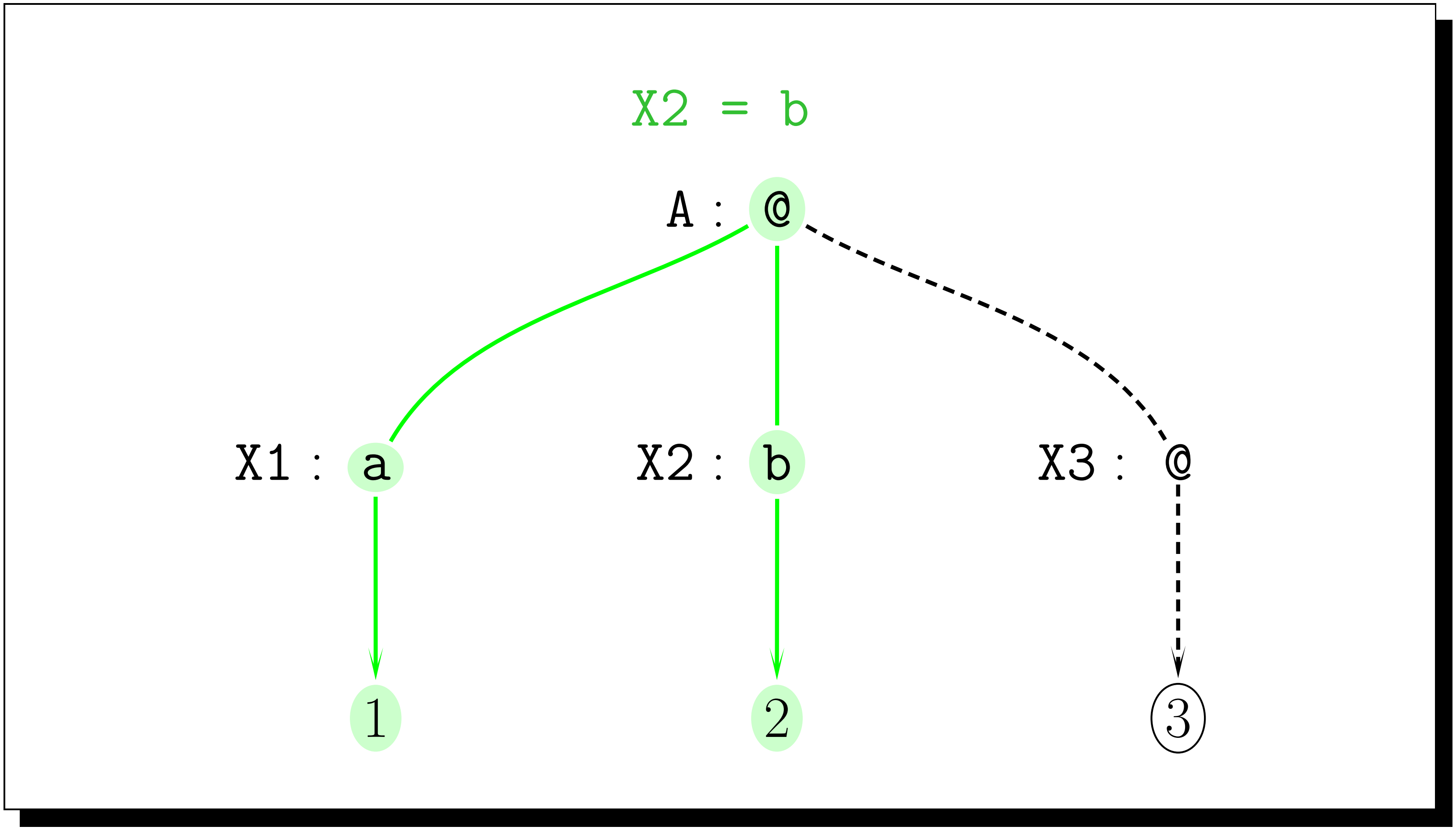


It's *all different* using graphs!

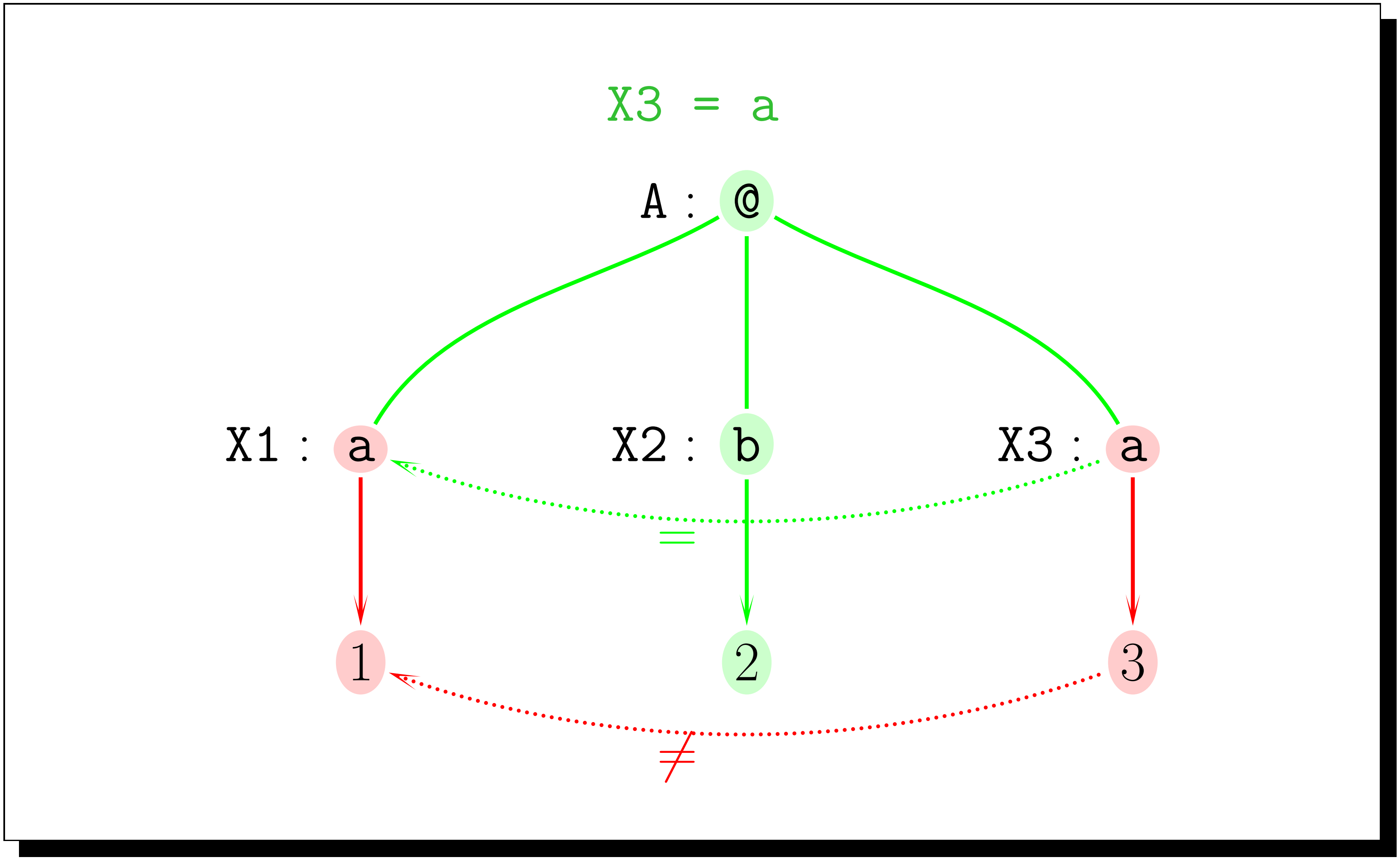


IBM | ILOG Products and Solutions

It's *all different* using graphs!



It's *all different* using graphs!



IBM | ILOG Products and Solutions

LIFE Su Doku—Outline

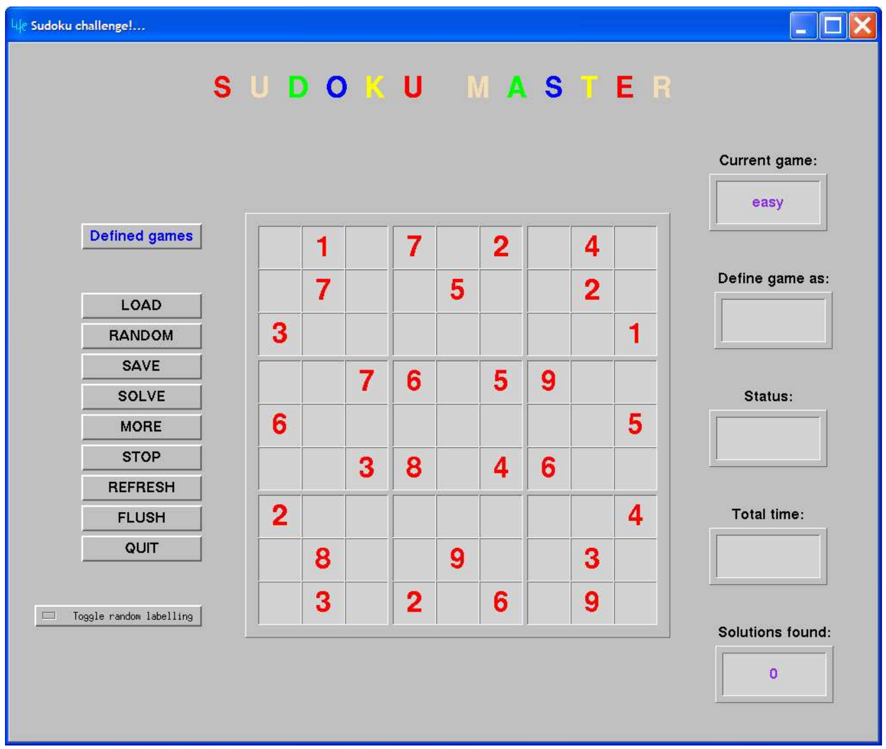
LIFE **bonus: a declarative Su Doku GUI**

ILOG Products and Solutions
IBM

LIFE bonus: a declarative *Su Doku* GUI

Life is just a mirror, and what you see out there,
you must first see inside of you.

WALLY 'FAMOUS' AMOS



A *LIFE* *Su Doku* game GUI display

Epilogue

In life, the earlier one fails, the earlier one eventually succeeds!

Altaïr El-Ghoul

ILOG Products and Solutions

Thank You For Your Attention !

ILOG Products and Solutions