Etude on Recursion Elimination…

Nikolay Shilov
Innopolis University (Russia)
(a talk at workshop on Verification &Program Transformation VPT-2017)

Part o

INTRODUCTION AND MOTIVATION

Puzzles named after Great People

- Recursive function M_{91} has the following definition:

$$M_{91}(n)=\begin{cases} n-10, \text{ if } n>100, \\ M_{91}(M_{91}(n+11)) \text{ otherwise.} \end{cases}$$

- It was introduce by John McCarthy, studied by him with Zohar Manna and Amir Pnueli, and by Donald Knuth.

On M_{91} function

- But this function can be computed using an auxiliary function as $M_{91}(n)=M_{aux}(n,1)$ where

$$
M_{aux}(n, m) = \begin{cases} n, \text{ if } m=0; \\ M_{aux}(n-10, m-1), \text{ if } n>100, m>0; \\ M_{aux}(n+11, m+1), \text{ if } n\leq 100, m>0. \end{cases}
$$

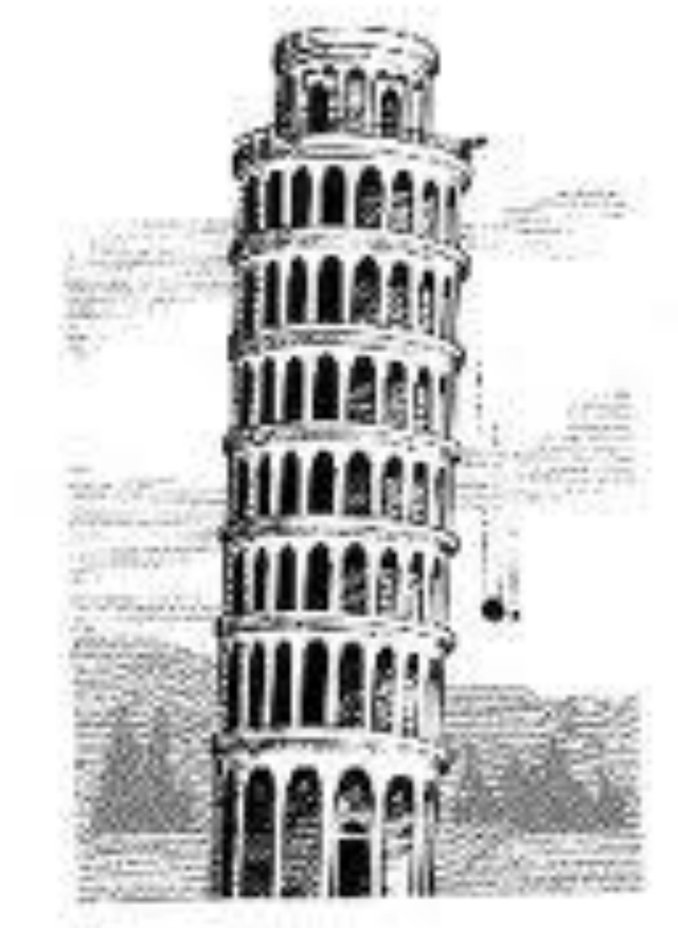
More on M_{91} function

- and even directly

$$M_{91}(n)=\begin{cases} n-10, \text{ if } n>101, \\ 91 \text{ otherwise.} \end{cases}$$

Dropping Bricks Problem

You have to define stability of bricks by dropping them from a tower of H meters. How many times do you need to drop bricks, if you have just 2 bricks?



Part I

BELLMAN EQUATION FROM PROGRAM SCHEMATA PERSPECTIVE

Descending (top-down) Dynamic Programming

- General pattern of Bellman equation may be formalised by the following *scheme of recursive descending Dynamic Programming*:

$$G(x) = \text{if } p(x) \text{ then } f(x)$$

$$\text{else } \underbrace{g(x, \{h_i(x, G(t_i(x)), i \in [1..n]\})}_{\substack{\text{the term is \textit{linear in each branch}} \\ \text{w.r.t. the objective function } G}};$$

Descending (top-down) Dynamic Programming (cont.)

- In this scheme
  - $G:X \rightarrow Y$ is a symbol for the objective function,
  - $p:X \rightarrow Bool$ is a symbol for a known predicate,
  - $f:X \rightarrow Y$ is a symbol for a known function,

Descending (top-down) Dynamic Programming (cont.)

- $g: X \times Z^* \to Y$ is a symbol for a known function with a variable (but finite) number of arguments,
- all $h_i: X \times Y \to Z$, $i \in [1..n]$ are symbols for known functions,
- all $t_i: X \to X$, $i \in [1..n]$ are symbols for known functions too.

Example 1: Discrete Knapsack Problem

- Bellman equation specifies *the maximal gross price* that is possible to collect:
- MaxP(W, n) = if n=0 then 0

  else if W_n> W then MaxP(W, n-1)

  else max{MaxP(W, n-1),

  MaxP(W-W_n, n-1)+P_n}

Example 1: Discrete Knapsack Problem

- It does not make sense to convert this functional program into imperative ascending dynamic programming form because a complexity to compute the support spp (the set of all values used in recursion) has the same complexity as computation of MaxP itself.

Example 2:
Integer Knapsack Problem

- But if it is known that knapsack capacity W and weights of all goods *are integers* (natural numbers) then it makes sense to use a trivial <u>upper approximation</u> for the support

$$SPP(W, N) = [0..W]\times[0..N].$$

Example 3: Dropping Bricks Problem

- In particular, in Dropping Bricks Problem:

$$G(H) = \text{if } \underbrace{H=0}_{\text{It is } p(x).} \text{ then } \underbrace{0}_{\text{It is } f(x).}$$

$$\text{else } \underbrace{1+\min_{1\leq h\leq H}\overbrace{\max\{(h-1),G(\underbrace{H-h}_{\text{It is } t_i(x).})\}}^{\text{It is } h_i(x).}}_{\text{It is } g(x,\ \{h_i(x,\ G(t_i(x)),\ i\in[1..n])\}).}.$$

More Examples: Factorial and Fibonacci Numbers

- Fac(n) = if n = 0 then 1else n * Fac(n -1);
- Fib(n) = if n = 0 or n = 1 then 1

  else Fib(n-2) + Fib(n-1).

Observations

- Discrete Knapsack needs stack or queue in dynamic memory,
- Integer Knapsack Problem needs array in dynamic memory to be allocated just once,
- Factorial and Fibonacci Numbers just need static memory of fixed size.

More Observations

- Surprisingly, but DBP also needs just static memory of fix-size, since

$$G(H) = \min \{n \in N : n \times (n + 1)/2 \geq H\}.$$

Part II

DYNAMIC, STATIC AND FIX-SIZE MEMORY

Problem

- When
  - stack/queue/associative array,
  - one-time allocated array,
  - fix-size static memory

  Is needed/suffice to implement Bellman equation?

A Need of Dynamic Memory

- It follows from

  Paterson M.S., Hewitt C.T. Comparative Schematology. Proc. of the ACM Conf. on Concurrent Systems and Parallel Computation, 1970, p.119-127.

  that *static memory* is *not sufficient* for general case of Bellman equation.

A Need of Dynamic Memory

- The following program scheme

  F(x) = if p(x) then x else f(F(g(x)), F(h(x)))

  is not equivalent to any standard program scheme :

  for every n>0

  there exists an interpretation T_n

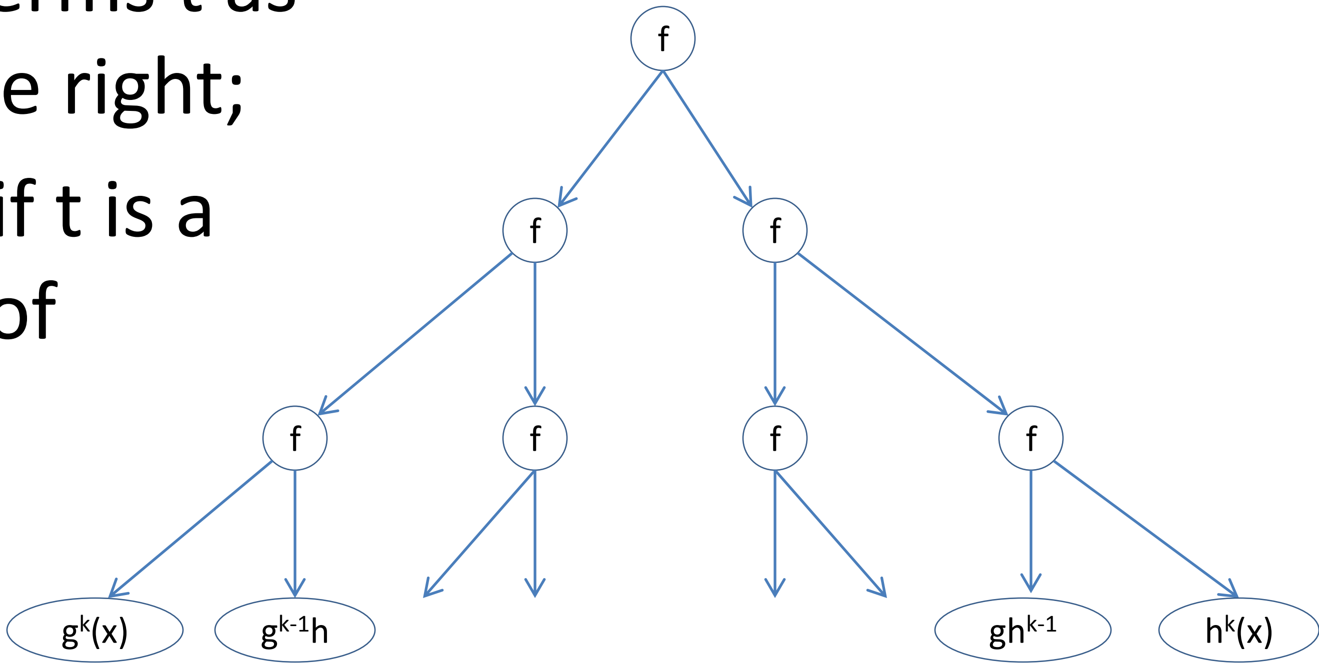
  where any standard program scheme

  needs n variables to compute F.

A Need of Dynamic Memory (proof)

Sample T_n:

- values are terms t as shown to the right;
- p(t) is true, if t is a binary tree of height n.



Support of the Objective Function

- If G(v) is defined for some argument value v, then it is possible to pre-compute the *support* spp(v) , the set of all argument values that occur in the computation of G(v)):

  spp(x) = if p(x) then {x}

  else $\{x\} \cup (\cup_{y \in bas(x)} spp(y))$.
- Remark, that for every argument value v, if G(v) is defined, then spp(v) is finite.

When one-time allocated array suffice

- One-time allocated array suffice for computing

  $G(x) = \text{if } p(x) \text{ then } f(x)$

  $\text{else } g(x, \{h_i(x, G(t_i(x)), i \in [1..n])\});$

  if all t_1, t_2, … t_n are interpreted by commutative functions.

When one-time allocated array suffice…

- It makes sense when

  $D_1(v)\times D_2(v)\times\ldots D_n(v)\times <$ complexity of spp(v)

  where $D_i(v) = \min \{ k : p(t^k(v)) \}$.
- Example: Integer Knapsack Problem, DBP.
- Counter-example: general case Discrete Knapsack Problem (since weights may be non-commensurable).

When fix-size static memory suffice

- Fix-size static memory suffice for computing

  $G(x)$ = if $p(x)$ then $f(x)$

  else $g(x, \{h_i(x, G(t_i(x)), i\in[1..n])\})$;

  if n=const and all $t_1, t_2, \ldots t_n$ are interpreted so that $t_i=t_1^{\,i}$ for all $i\in[1..n]$.
- Examples: Factorial and Fibonacci Numbers.
- Counter-example: Paterson-Hewitt scheme.

Part III

ANALYSIS OF DROPPING BRICKS PUZZLE

Question

- How to transform recursive program for DBP

  G(H) = if H=0 then 0

  else $1+\min_{1\le h\le H}\max\{(h-1),G(H-h)\}$

  into iterative one?

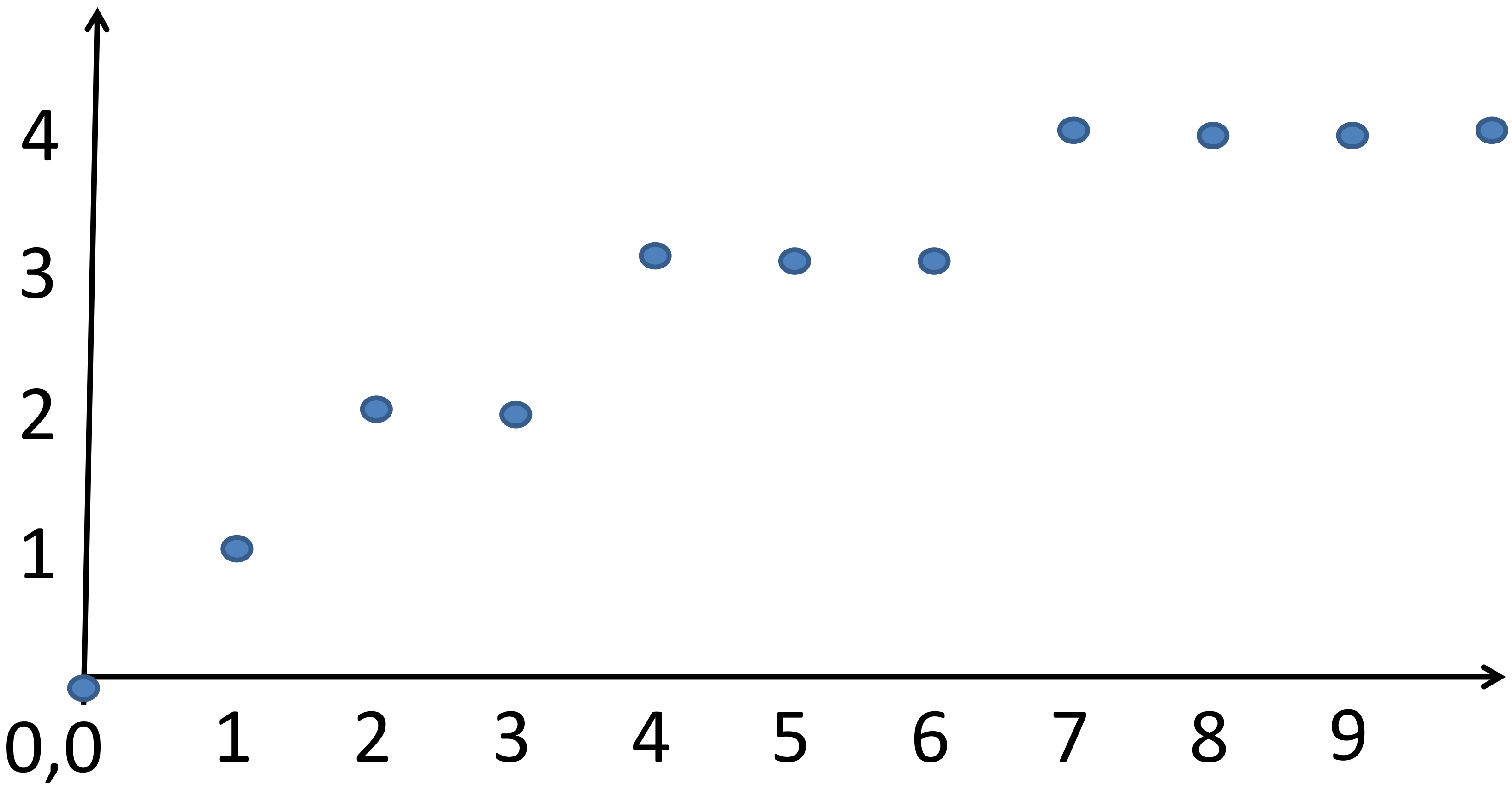
Towards a solution…

- $G(H)$ = if $H=0$ then 0

  else $1+\min_{1\leq h\leq H}\max\{(h-1),G(H-h)\}$.

  Is a monotone function without jumps >1 (see next slide). (But why?)

Towards a solution…



Towards a solution…

$G(x) = 1+$

$+ \min \{ \max\{(1-1), G(x-1)\},$

……………………………………

max(y-1), G(x-y)}

……………………………………

$\max\{(x-1), G(x-x)\} \}$

where element in bold is the last such that

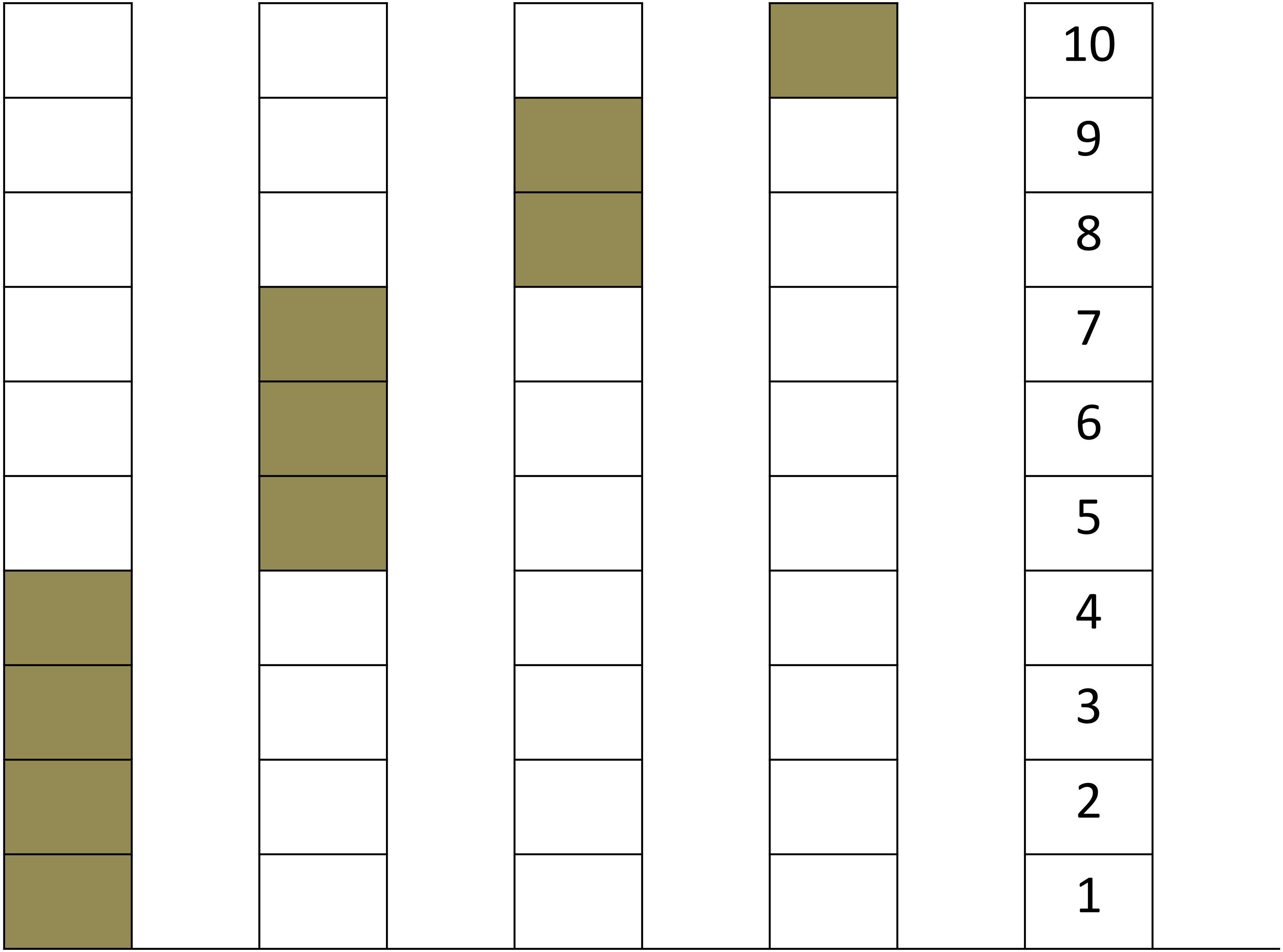
$(y-1) \leq G(x-y)$.

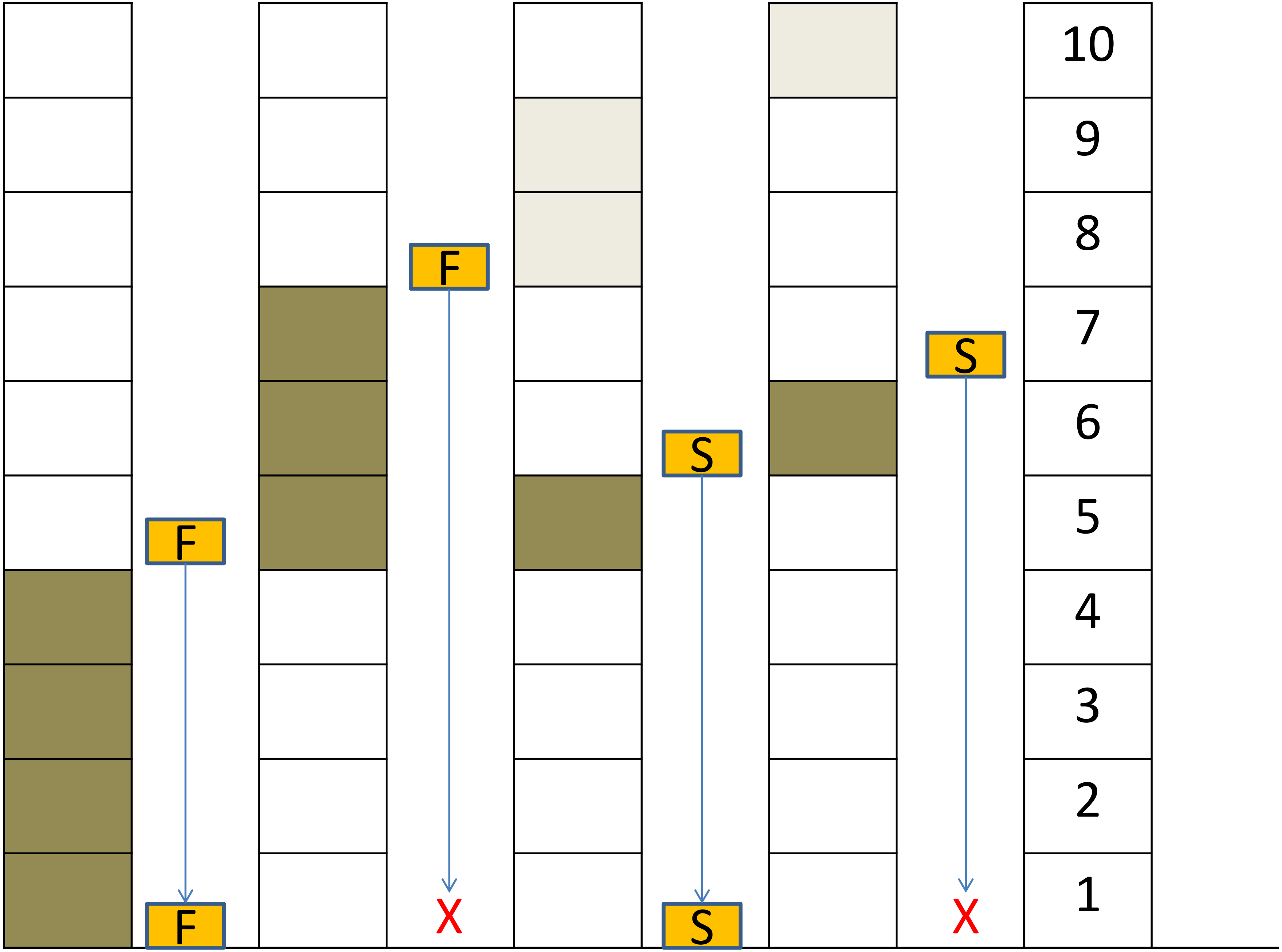
Towards a solution…

- Two possibilities for this y:
  - G(x-y)=y,
  - G(x-y)=(y-1).
- Let us consider the second option only. (Again, why?).
- If to adopt a= x-y, and b= y-1 then we have
  - G(a) = b,
  - G(a + b + 1) = b + 1.

Finally…

- $G(0) = 0$,
- $G(1) = G(0+1) = 0+1 = 1$,
- $G(3)= G(1+(1+1)) = (1+1) = 2$,
- $G(6) = G(3+(2+1)) = (2+1) = 3$,
- ………………………………………….
- $G((n+1)+n+\ldots+1) = G((n+\ldots+1) + (n+1)) = (n=1)$;
- thus $G(H) = \min \{n\in N : n\times(n + 1)/2 \geq H\}$.


10
9
8
7
6
5
4
3
2
1



10
9
8
7
6
5
4
3
2
1
F
F
F
X
S
S
S
X

Further Questions

- How to
  - make this program transformation formal?
  - generalize this transformation technique?

(Friendly) Questions and Critics Welcome!

- Questions?
- Comments?
- Suggestions?
- Refutations?