Verified Templates for design of Combinatorial Algorithms

N.V. Shilov

A.P. Ershov Institute of Informatics Systems,

(Novosibirsk, Russia)

SCAN 2012:

September 28, 2012

Novosibirsk, Russia

A sad split

There exists a split between reliable computing and program verification communities:

- sometimes it seems that computing people assume that program code that “implements” a reliable method can be justified by extensive testing,
- while verification people think that reliability of any specified computational program can be formally verified in automatic mode from scratch.

Looking for a compromise

We try to find a compromise both extreme viewpoints by suggesting, formalizing and verifying (manually but formally) templates for design of algorithms for combinatorial optimization.

Algorithm Design Patterns

Three algorithmic design patterns are core in the combinatorial optimization, namely:

- Dynamic Programming (DyP),
- Backtracking (BTR) and
- Branch-and-Bound (B&B).

Algorithm Design Patterns

They can be

- formalized as design templates,
- specified by correctness conditions,
- and formally verified in Floyd-Hoare style.

Relevance to SCAN

- Most global optimization methods using interval techniques employ a branch-and-bound strategy.
- These algorithms decompose the search domain into a collection of boxes, arrange them into a tree-structure (according to inclusion), and compute the lower bound on the objective function by an interval technique.

Verified Templates for B&B and Backtracking

BTR and B&B templates have been considered

- in brief: Shilov N.V. *Verified Template for Branch-and-Bound*. (Proceeding of Constraint Programming and Decision Making Workshop CoProD-2012),
- in full details: Shilov N.V. *Verification of Backtracking and branch-and-bound Design Templates*. (Modeling and analysis of information systems, 18(4), 2011, p.168-180, in Russian),
- English translation to appear in Automatic Control and Computer Sciences, 2012, 46(7) (distributed by Springer).

Towards DyP Template

- DyP formalization, specification and verification are topics of the talk.
- A methodological novelty consists in interpretation of DyP as the set-theoretic least fix-point (lfp) according to Knaster-Tarski theorem.

The first face of DyP

Dynamic programming was introduced by Richard Bellman in early 1950’s as a recursive method for solving optimization problems presented by appropriate Bellman equation:

- R. Bellman *The theory of dynamic programming.* Bulletin of the AMS, 60, 1954, p.503-516.

.

Canonical form for Bellman Equation

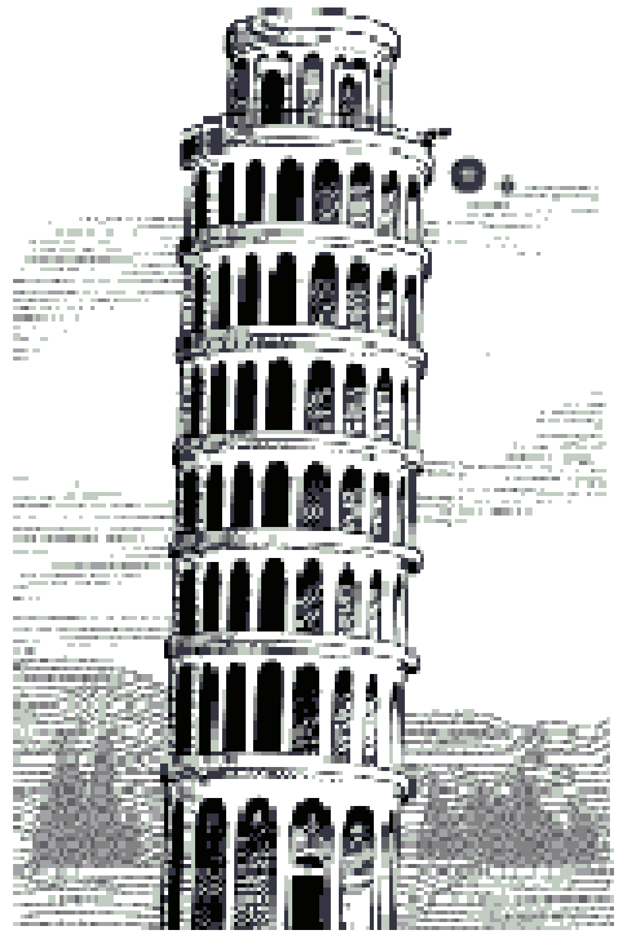
$G(x) = \text{if } p(x) \text{ then } f(x) \text{ else } g(x, (G(t_i(x)), i\in[1..n]))$

where

- $G:X\to Y$ is the objective function,
- $p\subseteq X$ is a known predicate,
- $f:X\to Y$ is a known function,
- $g:X^*\to X$ is a known function with a variable arity,
- all $t_i:X\to X$, $i\in [1..n]$ are known functions also.

Example: Dropping Bricks Puzzle

- Mechanical stability of a brick is the height (in meters) that is safe for the brick to fall down, while height (h+1) meters is unsafe.
- You have to define stability of bricks of a particular kind by dropping them from different levels of a tower of H meters.



Example: Dropping Bricks Puzzle

- How many times do you need to drop bricks, if you have 2 bricks in the stock?
- What is the optimal number (of droppings) in this case?

Example: Dropping Bricks Puzzle

Bellman Equation for Dropping Bricks Puzzle

G_H = if $H=0$ then 0

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

In particular, $G_{100}=14$.

Towards Iterative DyP

Definition 1:

Let $G:X \to Y$ be defined by Bellman equation.

- For every $v \in X$, such that $p(v)$ does not hold, let $bas(v) = \{t_i(v) : i \in [1..n]\}$.
- For every $v \in X$ let support be the set $spp(v)$ of all argument values that occur in the computation of $G(v)$.

Towards Iterative DyP

Proposition 1: For every $v \in X$, if the objective function G is defined for v, then spp(v) is finite, and can be pre-computed according to the following recursive algorithm:

$spp(x) = \text{if } p(x) \text{ then } \{x\} \text{ else } \{x\} \cup (\bigcup_{y \in bas(x)} spp(y))$.

Towards Iterative DyP

Denition 2: Let $G:X \to Y$ be defined by Bellman equation. Let us say that a function $SPP:X \to 2^X$ is an upper support approximation, if for every argument value v, the following conditions hold:

- $v \in SPP(v)$,
- $spp(u) \subseteq SPP(v)$ for every $u \in SPP(v)$,
- if spp(v) is finite then SPP(v) is finite.

Precondition for Iterative DyP

- $D \neq \emptyset$,
- S and P are *trivial* and *target* subsets in D,
- $F: 2^D \to 2^D$ is a call-by-value total *monotone* function,
- $R: 2^D \times 2^D \to Bool$ is a call-by-value total function *monotone* on the second argument.

Template and Postcondition for Iterative DyP

```
\\template:
var Z:= S, Z1 : subsets of D;
repeat Z1:=Z; Z:=F(Z)
until (R(P,Z) or Z=Z1)
\\Postcondition:
```

$R(P,Z) \Leftrightarrow R(P, \mathrm{lfp}\ \lambda Q{:}(S \cup F(Q)))$

Correctness of Iterative DyP

Proposition. 2:

- DyP template is partially correct, i.e. for any input data that meets the precondition, the algorithm instantiated from the template either loops or halts in such a way that the postcondition holds upon the termination.
- Assuming that for some input data the precondition of DyP template is valid, and the domain D is finite, then the algorithm instantiated from the template terminates after at most |D| iterations of the loop repeat-until.

Correctness of Iterative DyP

The proposition can be proved by Knaster-Tarski fixpoint theorem, since template described finite computations of the least fixpoint according to procedure suggested by Knaster and Tarski.

Why it is DyP?

If to adopt

- the graph of G on SPP(v) as D,
- a set $\{(u, f(u)) : p(u) \ \& \ u \in SPP(v)\}$ as S,
- a singleton $\{(v,G(v))\}$ as P,
- a mapping $\lambda Q. \{(u,w) \in D :$

$$\exists w_1 \ldots \exists w_n : (t_1(u),w_1),\ldots(t_n(u),w_n) \in Q \ \& \ w = g(u,w_1,\ldots w_n)\}$$

as $F:2^D \to 2^D$,

- and $\exists w: (v,w) \in (X \cap Y)$ as $R(X,Y):2^D \times 2^D \to Bool$,

Why it is DyP?

then the instantiated algorithm computes G(v) in the following sense:

- it terminates after iterating repeat-until loop |SPP(v)| times at most,
- upon the termination $(v,G(v))\in Z$,
- and there is no any $w\neq G(v)$ such that $(v,w)\in Z$.

Why it is DyP?

I would not like to forth everyone to think about Dynamic Programming in terms of fixpoint computations, but I do believe that the verified iterative DyP template will help to teach and automatize reliable algorithm design.

Why it is DyP?

The template-based approach to teaching Dynamic Programming is in use in Master program at Information Technology Department of Novosibirsk State University since 2003.

Check full details in: N. Shilov *Inverting Dynamic Programming.* Proc. of 3rd Int. Valentin Turchin Workshop on Metacomputation, Pereslavl-Zalessky, Russia, July 5-9, 2012, p.216-227.

Why it is DyP?

A possible application of the unified template is data-flow parallel implementation of the Dynamic Programming, but this topic need more research.

Questions?

Thanks!