Program Schemes

Reunión MOISES, 3 Septiembre 2004

Argimiro Arratia

Universidad de Valladolid

arratia@mac.uva.es

A **program scheme** $\rho \in \text{ NPS}(\tau)$ involves a finite set of variables and is over some vocabulary τ. It consists of a finite sequence of instructions:

First is $\text{INPUT}(x_1, \ldots, x_m)$ and last is $\text{OUTPUT}(x_1, \ldots, x_m)$.
The others are

(i) *assignment instruction* of the form:
$var := atom$, where an *atom* is a variable or constant symbol of $\tau \cup \{0, max\}$ (0, max two special constants not in τ), or
GUESS *var*

(ii) *test instruction* : WHILE t DO $\alpha_1; \ldots; \alpha_q$ OD
where each $\alpha_1; \ldots; \alpha_q$ is an instruction (assignment or test) and t is a conjunction of *simple tests*, or their negations, of the form: $atom = atom$ and $R(atom, \ldots, atom)$,
where R is a relation symbol of τ.
(note we can have nested while instructions)

NPS$= \{\rho \in \mathrm{NPS}(\tau) \,|\, \tau$ is some vocabulary$\}$

(NPS $+ \leq$) means we have a successor operation $y := x + 1$ built-in into programs

A program scheme $\rho \in \mathrm{NPS}(\tau)$ takes as input a finite τ-structure $\mathcal{A}$, by interpreting each constant and relation of τ as constants and relations of $\mathcal{A}$; the 0 and max as two different elements of $\mathcal{A}$, and initially setting input/output variables to 0

$\mathcal{A}$ **is accepted by** ρ ($\mathcal{A} \models \rho$) if for some sequence of guesses (applied wherever the inst. GUESS var appears), that nondeterministically assign values from $\mathcal{A}$ to the variables of ρ, causes the program to halt with all its input/output variables set to max

DPS is NPS without GUESS

Program Schemes Background

1970s : Program Schemata (Constable & Gries, 1972). No attention paid to resources

1980s : Datalog, Dynamic Logics, Logics of Programs (Harel & Peleg, 1984; Neil Jones, 1977-90s; Tiuryn, 1988). Relates programming with computational complexity

1990s : Program Schemes developed mainly by Stewart, close to Dynamic Logics or Datalog, but tied up with Finite Model Theory (e.g. inputs are not strings from binary alphabets but finite structures)

Example

Let $\rho \in$ NPS(σ), $\sigma = \{E, C, D\}$, be

```
INPUT(x, y)
x := C
WHILE x ≠ D DO
    GUESS y
    WHILE ¬E(x, y) DO
        GUESS y OD
    x := y OD
x := max
y := max
OUTPUT(x, y)
```

Then $\mathcal{A} \models \rho \iff \mathcal{A} \in$ TC

An advantage of the Prog. Scheme model: we can incorporate operations as STACKS and ARRAYS

Programs with Stacks

NPSS is NPS plus the instructions:

$$var := \text{POP}$$

$$\text{PUSH } var$$

Programs in NPSS come with a stack. PUSH *var* places the value of *var* on top of the stack. *var* := POP removes the value of the top of the stack and *var* assumes this value. All programs begin computing with empty stack. A program accepts its input as before but on termination the stack must be empty.

Programs with Arrays

NPSA is NPS plus the instructions:

(assignment) $A[atom, \ldots, atom] := atom$

(test) $atom = A[atom, \ldots, atom]$

A is an array symbol.

Programs in NPSA may have more than one array, and prior to computation all arrays variables are set to 0

DPSA is NPSA without GUESS.
(NPSA + $\leq$) has a built-in successor operation

NOTE: IF THEN ELSE is definable from WHILE

Example : Let $\rho \in$ (NPSA$+ \leq$)(σ), $\sigma = \{E\}$, be

INPUT(x)
$A[0] := max$
WHILE $x \neq 0$ DO
  GUESS y
  IF $x \neq y \wedge E(x,y) \wedge A[y] = 0$ THEN
    $A[y] := max$
    $x := y$ FI OD
$(x, z) := (0, 0)$
WHILE $x \neq max \wedge z = 0$ DO
IF $A[x] = max$ THEN
  $x := x + 1$
ELSE $z := max$ FI
OD
IF $A[max] = max \wedge z = 0$
  THEN $x := max$
ELSE $x := 0$ FI
OUTPUT(x)

Then $\mathcal{A} \models \rho \iff \mathcal{A} \in$ HAMCircuit

NPSB is NPSA but assignment to array elements are force to be only 0 or max, i.e., only allow $A[atom, \ldots, atom] := max$. Initially arrays are set to 0 and once they are set to max they keep that value to the end.

Example: the program scheme in previous slide is in NPSB

Programming vs Complexity

(Stewart 93) (DPS $+ \leq$) $=$ **L**

(Ste 93)(Arratia, Chauhan, Stewart 99) (NPS $+ \leq$) $=$ **NL**

(A, C, S 99) (NPSS $+ \leq$) $=$ **P**

(Stewart 00) (NPSA $+ \leq$) $=$ (DPSA $+ \leq$) $=$ **PSPACE**

(Stewart 02) (NPSB $+ \leq$) $=$ **NP**

Program Schemes w/o Order

Define

NPS(1) = NPS = prog. schm. where test in WHILE-loops is quantifier free FO-formula

NPS(2) = $\{\forall x_1 \ldots \forall x_p \, \rho : \rho \in \mathrm{NPS}(1)\}$

NPS(3) = $\{\rho :$ test in WHILE is $\rho' \in \mathrm{NPS}(2)\}$

and so on ...

A similar hierarchy can be defined from NPSS

Let $\sigma = \{E, U\}$, E binary, U unary. View σ-structures as digraphs with specified set of vertices or *roots*, U. Let $\rho' \in \mathrm{NPS}(3)$ be

INPUT(x)
GUESS x
WHILE $\neg U(x)$ DO
    GUESS x OD
IF $\forall y\, \rho(x, y)$ THEN
    $x := max$ ELSE
    $x := 0$ FI
OUTPUT(x)

where $\rho \in \mathrm{NPS}(1)$ is

INPUT(z, w)
$z = x$
WHILE $z \neq y$ DO
    GUESS w
    IF $E(z, w)$ THEN
    $z := w$ FI OD
$(z, w) := (max, max)$
OUTPUT(z, w)

$\mathcal{A} \models \rho' \iff \mathcal{A}$ is a rooted digraph where at least one root have paths to every other vertice

Theorem [A,C,S 99]: Over arbitrary finite structures

1. $\mathrm{NPS}(1) \subset \ldots \subset \mathrm{NPS}(m) \subset \mathrm{NPS}(m+1) \subset \ldots$
2. $\mathrm{NPSS}(1) \subset \ldots \subset \mathrm{NPSS}(m) \subset \mathrm{NPSS}(m+1) \subset \ldots$
3. $(\pm\mathrm{TC})^*[\mathrm{FO}] = \bigcup_{m\geq 1} \mathrm{NPS}(m) \subset \bigcup_{m\geq 1} \mathrm{NPSS}(m) = (\pm\mathrm{PS})^*[\mathrm{FO}]$

But, in the presence of a built-in successor relation

1. $\mathrm{NPS}(1) = \bigcup_{m\geq 1} \mathrm{NPS}(m) = (\pm\mathrm{TC})^*[\mathrm{FO}] = \mathbf{NL}$
2. $\mathrm{NPSS}(1) = \bigcup_{m\geq 1} \mathrm{NPSS}(m) = (\pm\mathrm{PS})^*[\mathrm{FO}] = \mathbf{P}$

The tools have an Ehrenfeucht-Fraissé flavour

Theorem[Hierarchy Theorem for NPS]: Let $\rho \in \mathrm{NPS}(\tau)$. If there exists families of τ-structures, $\{\mathcal{A}_k\}_{k\geq 0}$ and $\{\mathcal{B}_k\}_{k\geq 0}$, such that:

(i) for each $k \geq 0$, $\mathcal{A}_k \subseteq \mathcal{B}_k$ and for all sentence of the form $\psi := \exists \overline{x} \forall \overline{y} \phi(\overline{x}, \overline{y})$, with ϕ a quantifier-free first-order formula and $|\overline{y}| + |\overline{x}| \leq k$, we have

$$\mathcal{A}_k \models \psi \ \text{ iff } \ \mathcal{B}_k \models \psi;$$

(ii) for some $\beta \in \mathrm{NPS}(1)$ and all $k \geq 0$,

$$\mathcal{A}_k \models \beta \text{ and } \mathcal{B}_k \not\models \beta.$$

Then, for all $m \geq 0$, $\quad \mathrm{NPS}(m) \subset \mathrm{NPS}(m+1)$.

Computational complexity open problems equivalent to programming queries

Theorem [Ste 02]: The following are equivalent:

1. **NP** = **PSPACE**
2. NPSB = NPSA, over arbitrary finite structures.

(This is like Abiteboul & Vianu logical characterisation of the **P** = **PSPACE** question)

Variations of the program scheme model:

an alternative way of doing iteration

Program Schemes based on FOR-loops

A program scheme $\rho \in$ RFDPS involves a finite set of variables, a finite set of array symbols and is over some vocabulary σ. It consists of a finite sequence of instructions:

First is INPUT$(x_1, \ldots, x_m)$ and last is OUTPUT$(x_1, \ldots, x_m)$, and

(i) *assignment instruction* of the form: $\tau := atom$,
where τ is variable or array term, and *atom* is variable, array term or constant symbol of $\tau \cup \{0, max\}$

(ii) *if-then-fi-block*: IF φ THEN $\alpha_1, \ldots, \alpha_l$ FI
where φ is quantifier-free FO-formula in $\sigma \cup \{0, max\}$

(iii) *repeat-do-od-block* : REPEAT DO $\alpha_1; \ldots; \alpha_l$ OD

(iv) *forall-do-od-block* : FORALL x WITH A^j DO $\alpha_1; \ldots; \alpha_l$ OD
FORALL x DO $\alpha_1; \ldots; \alpha_l$ OD
where each $\alpha_1; \ldots; \alpha_q$ is an instruction

How computation goes

Given an input model $\mathcal{A}$ of size n:

- REPEAT DO $\alpha_1; \ldots; \alpha_l$ OD :
  iteratively executes the block of instructions $\alpha_1; \ldots; \alpha_l$ n times
- FORALL x WITH A^j DO $\alpha_1; \ldots; \alpha_l$ OD :
  n child processes are set off in parallel, one for each value $u \in |\mathcal{A}|$, which is taken by the "control" variable x and the jth entry of array A. When all child processes terminates, i.e., reaches the *forall-od* inst., if the values of local variables are all max, then the value of x is set at max, otherwise is set at 0.

$\mathcal{A}$ is accepted by $\rho \iff$ exist distinct values for symbols 0 and max for which the computation of ρ on input $\mathcal{A}$ reaches OUTPUT with all variables set at max

Theorem [Gault & Stewart 04]: There is a program scheme of RFDPS accepting any first order definable problem

However, RFDPS can compute problems not first order definable. Let ρ:

```
INPUT(x)
REPEAT DO
    IF x = 0 THEN
        x := max
    ELSE
        x := 0 FI
OD
OUTPUT(x)
```

Here acceptance and rejection is independent of the distinct chosen values for 0 and max. For any vocabulary σ, a σ-structure is accepted by ρ iff it has odd size.

Moreover

Theorem [G & S, 04]: There is a program scheme of RFDPS accepting any problem definable in IFP logic.

However

RFDPS does not captures **P**

Conclusions

With Program Schemes:

- we can explore computational complexity through a programming style, which should be more appealing than the Turing machine to the programmer;
- we can design new logics (in the broad sense set forth by Gurevich, as language with recursive syntax and semantics) by adding programming constructs (e.g. arrays, stacks, and different forms of recursion) which are hard to conceive within the rigid framework of Logic;
- take advantage of its programming nature to visualise problems as programs, and its link with Finite Model Theory for techniques for proving power of computation