

# 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  $\tau$ . It consists of a finite sequence of instructions:

First is  $INPUT(x_1, \ldots, x_m)$  and last is  $OUTPUT(x_1, \ldots, x_m)$ . 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  $\tau$ ), 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  $\tau$ . (note we can have nested while instructions)

# $NPS = \{ \rho \in NPS(\tau) \mid \tau \text{ is some vocabulary} \}$

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

4

A program scheme  $\rho \in \text{NPS}(\tau)$  takes as input a finite  $\tau$ -structure  $\mathcal{A}$ , by interpreting each constant and relation of  $\tau$  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  $\rho$  ( $\mathcal{A} \models \rho$ ) if for some sequence of guesses (applied wherever the inst. GUESS var appears), that nondeterministically assign values from  $\mathcal{A}$  to the variables of  $\rho$ , 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 
$$\rho \in \operatorname{NPS}(\sigma), \sigma = \{E, C, D\}$$
, be  
1 INPUT $(x, y)$   
2  $x := C$   
3 WHILE  $x \neq D$  DO  
4 GUESS  $y$   
5 WHILE  $\neg E(x, y)$  DO  
6 GUESS  $y$  OD  
7  $x := y$  OD  
8  $x := max$   
9  $y := max$   
10 OUTPUT $(x, y)$ 

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

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 := POP

#### 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, ..., atom] := atom

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

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

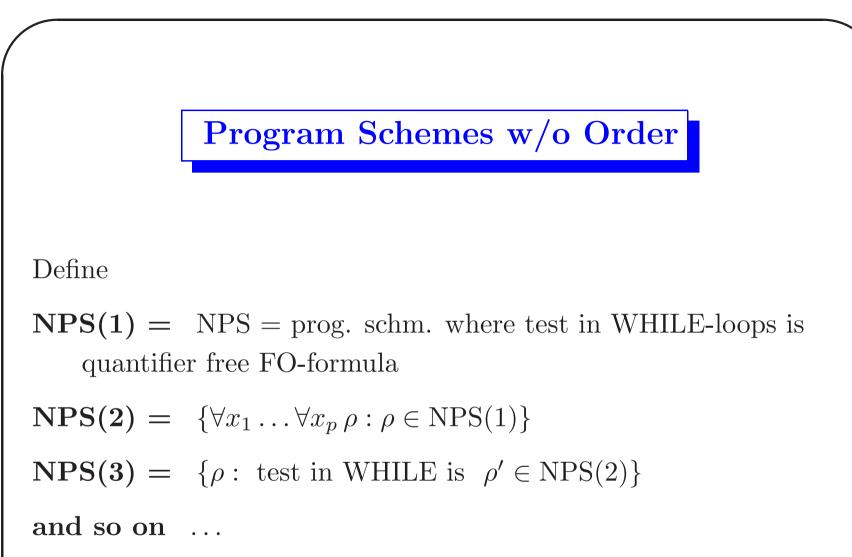
NOTE: IF THEN ELSE is definable from WHILE

**Example :** Let 
$$\rho \in (NPSA + \leq)(\sigma), \sigma = \{E\}$$
, be  
1 INPUT(x)  
2  $A[0] := max$   
3 WHILE  $x \neq 0$  DO  
4 GUESS y  
5 IF  $x \neq y \land E(x, y) \land A[y] = 0$  THEN  
6  $A[y] := max$   
7  $x := y$  FI OD  
8  $(x, z) := (0, 0)$   
9 WHILE  $x \neq max \land z = 0$  DO  
10 IF  $A[x] = max$  THEN  
14 IF  $A[max] = max \land z = 0$   
11  $x := x + 1$   
15 THEN  $x := max$   
12 ELSE  $z := max$  FI  
16 ELSE  $x := 0$  FI  
13 OD  
17 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



A similar hierarchy can be defined from NPSS

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

- 1 INPUT(x) where  $\rho \in NPS(1)$  is
- 2 GUESS x 1 INPUT(z, w)
- 3 WHILE  $\neg U(x)$  DO 2 z
- 4 GUESS x OD 3 WHIL
  - 5 IF  $\forall y \rho(x, y)$  THEN

$$6 x := max \text{ ELSE}$$

 $7 x := 0 ext{FI}$ 

8 OUTPUT(x)

 $2 \qquad z = x$ 

- 3 WHILE  $z \neq y$  DO
- 4 GUESS w
- 5 IF E(z, w) THEN

$$6 z := w \text{ FI OD}$$

$$(z,w) := (max, max)$$

8 OUTPUT(z, w)

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

7

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

1.  $NPS(1) \subset \ldots \subset NPS(m) \subset NPS(m+1) \subset \ldots$ 

2. 
$$\operatorname{NPSS}(1) \subset \ldots \subset \operatorname{NPSS}(m) \subset \operatorname{NPSS}(m+1) \subset \ldots$$

3. 
$$(\pm \text{TC})^*[\text{FO}] = \bigcup_{m \ge 1} \text{NPS}(m) \subset \bigcup_{m \ge 1} \text{NPSS}(m) = (\pm \text{PS})^*[\text{FO}]$$

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

1. NPS(1) = 
$$\bigcup_{m \ge 1}$$
 NPS(m) =  $(\pm TC)^*$ [FO] = **NL**

2. 
$$\operatorname{NPSS}(1) = \bigcup_{m \ge 1} \operatorname{NPSS}(m) = (\pm \operatorname{PS})^*[\operatorname{FO}] = \mathbf{P}$$

#### The tools have an Ehrenfeucht-Fraissé flavour

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

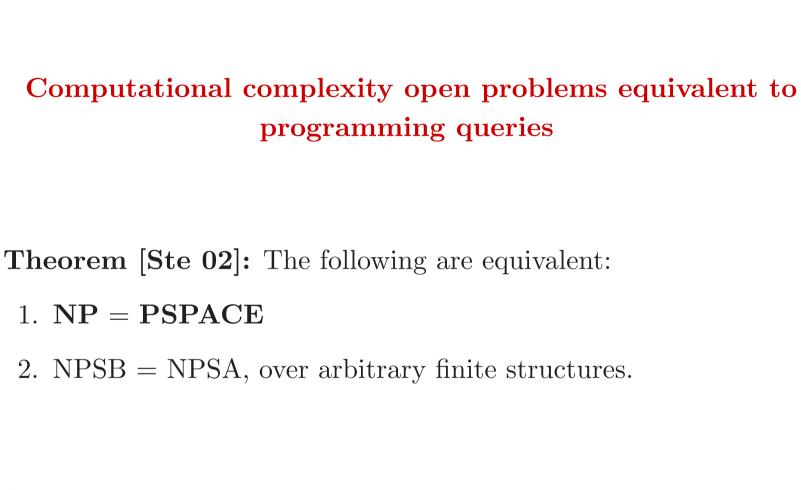
(i) for each  $k \ge 0$ ,  $\mathcal{A}_k \subseteq \mathcal{B}_k$  and for all sentence of the form  $\psi := \exists \overline{x} \forall \overline{y} \phi(\overline{x}, \overline{y})$ , with  $\phi$  a quantifier-free first-order formula and  $|\overline{y}| + |\overline{x}| \le k$ , we have

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

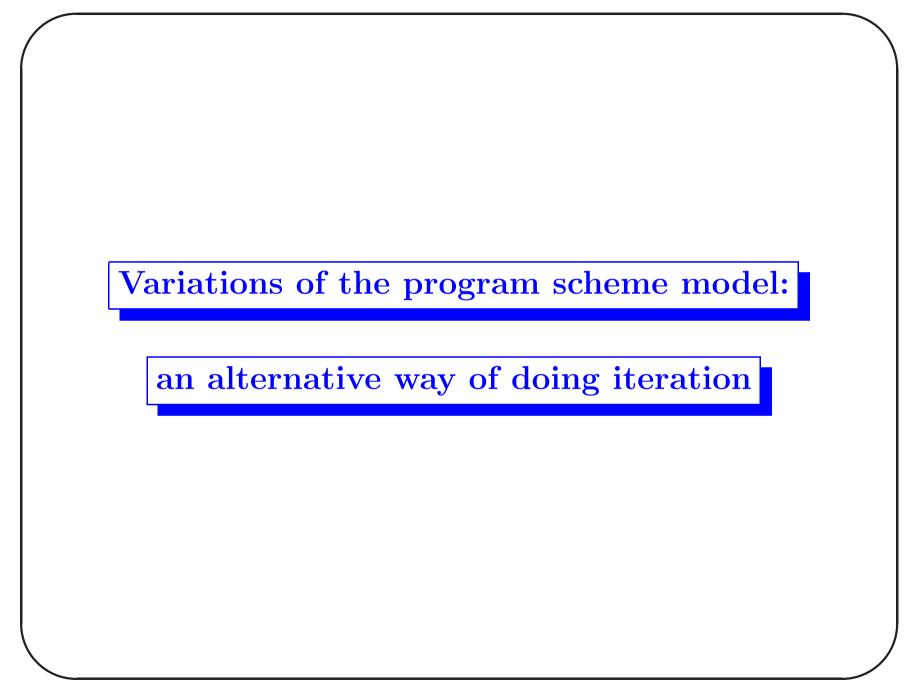
(ii) for some  $\beta \in NPS(1)$  and all  $k \ge 0$ ,

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

Then, for all  $m \ge 0$ ,  $NPS(m) \subset NPS(m+1)$ .



(This is like Abiteboul & Vianu logical characterisation of the  $\mathbf{P} = \mathbf{PSPACE}$  question)



#### **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  $\sigma$ . 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  $\tau$  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  $\varphi$  THEN  $\alpha_1, \ldots, \alpha_l$  FI where  $\varphi$  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  $\rho$  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  $\rho {:}$ 

- 1 INPUT(x)
- 2 REPEAT DO
- 3 IF x = 0 THEN
- 4 x := max
- 5 ELSE
- $6 x := 0 ext{ FI}$
- 7 OD
- 8 OUTPUT(x)

Here acceptance and rejection is independent of the distinct chosen values for 0 and max. For any vocabulary  $\sigma$ , a  $\sigma$ -structure is accepted by  $\rho$  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 \mathbf{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