A characterization of Polynomial Space with Forks

Romain Péchoux
joint work with Emmanuel Hainry and Jean-Yves Marion

Lorraine University
Inria project Carte, Loria

30 mai 2012
Outline

Motivations

Semantics

Strong normalization, lock-freedom and confluence

Type system

Safe processes

Main characterization
ICC and polynomial Space

ICC related works on polynomial space:

- Function algebra with parameter substitution (Leivant-Marion 94)
- Function algebra with ramified recurrence (Leivant-Marion 97)
- Quasi-interpretation with LPO (Bonfante-Marion-Moyen 07)
- Lambda calculus with LLL based type system (Gaboardi-Ronchi Della Rocca-Marion 07)
- Higher-order types or life without cons (Jones 01)
- Matrix calculus (Niggl-Wunderlich, Jones-Kristiansen, Moyen)
Our approach

Take the methodology of the type system presented in LICS 2011 (Marion) that combines :

- data ramification principle (or tiering)
- with non-interference based type system
- on a simple imperative language

in order to characterize polynomial space on an imperative language with wait/fork mechanism.

Advantages of the presented methodology :

- a good expressivity
- very close to C-fork processes
Simple While Language with fork/wait

\[ E, E_1, \ldots, E_n \in \text{Exp} \ := \ X \mid \text{op}(E_1, \ldots, E_n) \]

\[ I \in \text{Inst} \ := \ \text{fork}() \mid \text{wait}(E) \]

\[ C, C' \in \text{Cmd} \ := \ X:=E \mid C \mid C' \mid \text{skip} \]
\[ \mid \text{while}(E)\{C\} \]
\[ X:=I \mid \text{if} \ E \ \text{then} \ C \ \text{else} \ C' \]

\[ P \in \text{Proc} \ := \ \text{return} \ X \mid C \mid P \]

\[ X \in \mathcal{X} \text{ and op} \in \mathcal{O} \]
Main results

A type system for imperative programs such that:

- Typable programs are computable in polynomial space under some restrictions:
  - termination
  - confluence
  - lock-freedom
  - the return type

- Each polynomial space problem can be computed by a typed program

- In a terminating program, all processes compute in polynomial time (polynomial number of steps)

A process being either the main program process or a subprocess created by a fork instruction.
Informal semantics: store, configuration and environment

Let $\mathcal{W}$ be the set of words over $\Sigma$. Sequential commands are evaluated as usual. A process $P$ is evaluated inside a configuration $c = (P, \mu)_{\rho}$:

- $P$ is the program counter
- A store $\mu : \mathcal{X} \rightarrow \mathcal{W}$ mapping each variable of $P$ to a value
- A set of ids $\rho$, the sons of $c$
- Each configuration has an id (an integer). The main process id is 1.

All configurations are stored in an environment $\mathcal{E}$, a partial function, mapping an id $\in \mathbb{N}$ to a configuration $c$. 
Informal semantics : fork

At the beginning, there is only one configuration (the main process) of id 1 and with $\rho = \emptyset$.

A fork instruction creates a new child:

- with a new id (set to the next available integer)
- that runs concurrently of its father
- with its own duplicated memory (the store and the program counter are duplicated)
- the child id is stored in the father $\rho$
Informal semantics: wait

The wait instruction provides a *one-way* communication and is the only way for a father to communicate with its child.

A $\text{wait}(E)$ instruction:

- evaluates the expression $E$ to a binary number $n$ encoding id $n$
- if $n \in \rho$ and the child is *returning* then:
  - the child return value is passed to the father
  - the child is erased
- otherwise the father waits for its children

Note that children of a killed father may still be alive
Example

P: \( X := \text{fork}(); \)
Q: \( \text{if } X > 0 \text{ then } \{
\)
R: \( \quad Y := \text{wait}(X);
\)
\( \quad Y := Y + 1
\)
\( \}\quad \text{else } \{
\)
\( \quad Y := 17
\)
\( \}
\)
S: \( \text{return } Y \)

Initial environment :
\( \mathcal{E}(1) = (P, \mu)_\emptyset \)
Fork evaluation :
\( \mathcal{E}(1) = (Q, \mu\{X := 2\})_{\{2\}} \)
\( \mathcal{E}(2) = (Q, \mu\{X := 0\})_\emptyset \)
After some steps :
\( \mathcal{E}(1) = (R, \mu\{X := 2\})_{\{2\}} \)
\( \mathcal{E}(2) = (S, \mu\{X := 0, Y := 17\})_\emptyset \)
Wait evaluation :
\( \mathcal{E}(1) = (S, \mu\{X := 2, Y := 18\})_{\{2\}} \)
\( \mathcal{E}(2) = \bot \)
Semantics of expressions and configurations

\[(X, \mu) \xrightarrow{\varepsilon} \mu(X)\]

\[(\text{op}(E_1, \ldots, E_n), \mu) \xrightarrow{\varepsilon} \llbracket \text{op} \rrbracket (d_1, \ldots, d_n)\]
\[
\text{if } \forall i, (E_i, \mu) \xrightarrow{\varepsilon} d_i
\]

\[(\text{skip}; P, \mu) \xrightarrow{\varsigma} (P, \mu)\]

\[(X := E; P, \mu) \xrightarrow{\varsigma} (P, \mu\{X \leftarrow d\})\]
\[
\text{if } (E, \mu) \xrightarrow{\varepsilon} d
\]

\[(\text{if } E \text{ then } C_{tt} \text{ else } C_{ff}; P, \mu) \xrightarrow{\varsigma} (C_w; P, \mu)\]
\[
\text{if } (E, \mu) \xrightarrow{\varepsilon} w \in \{tt, ff\}
\]

\[(\text{while}(E)\text{do}\{C\}; P, \mu) \xrightarrow{\varsigma} (P, \mu)\]
\[
\text{if } (E, \mu) \xrightarrow{\varepsilon} ff
\]

\[(\text{while}(E)\text{do}\{C\}; P, \mu) \xrightarrow{\varsigma} (C; \text{while}(E)\text{do}\{C\}; P, \mu)\]
\[
\text{if } (E, \mu) \xrightarrow{\varepsilon} tt
\]
Semantics of environments

Let $\mathcal{E}' = \mathcal{E}[i := c]$ be defined by:

- $\mathcal{E}'(j) = \mathcal{E}(j), \forall j \in \text{dom}(\mathcal{E}) - \{i\},$
- $\mathcal{E}'(i) = c$

The transition $\rightarrow$ for process evaluation is defined by:

$\mathcal{E}[i := c] \rightarrow \mathcal{E}[i := c']$
if $c \xrightarrow{c} c'$

$\mathcal{E}[i := (X := \text{fork}()); P, \mu)_\rho]$
$\rightarrow \mathcal{E}[i := (P, \mu\{X \leftarrow n\})_\rho \cup \{n\}, n := (P, \mu\{X \leftarrow 0\})_\emptyset]$
with $n = \#\mathcal{E} + 1$

$\mathcal{E}[i := (X := \text{wait}(E); P, \mu)_\rho]$
$\rightarrow \mathcal{E}[i := (P, \mu\{X \leftarrow \mu'(Y)\})_\rho, n := \bot]$
if $(E, \mu) \xrightarrow{e} n, n \in \rho$ and $\mathcal{E}_n = (\text{return } Y, \mu')$
Strong normalization, lock-freedom and confluence

Strong normalization
A process $P$ is strongly normalizing if $\forall \mu$ there is no infinite reduction starting from $(P, \mu)_\emptyset$ through the relation $\rightarrow$.

Lock-freedom
If $E \not\rightarrow$ and $E_1 = (X := \text{wait}(E); P, \mu)_\rho$ then $E$ is locked.
A process $P$ is lock-free if $\forall \mu$, there is no locked environment $E'$ s.t. $(P, \mu)_\emptyset \overset{*}{\rightarrow} E'$.

Confluence
A process $P$ is confluent if $\forall \mu$, $(P, \mu)_\emptyset \overset{*}{\rightarrow} E'$ and $(P, \mu)_\emptyset \overset{*}{\rightarrow} E''$, $\exists E^3$ s.t. $E' \overset{*}{\rightarrow} E^3$ and $E'' \overset{*}{\rightarrow} E^3$. 
Example of non-confluent process

\[
P: \quad X := \text{fork}(); \\
    Y := \text{fork}(); \\
    \text{return} \ Y
\]

The main process will return the process identifier of its second son. Depending on the order in which execution of the subprocesses occurs, this identifier can be either 3 or 4.
Tiers and typing environments

- **Tiers** are two elements 0, 1 underlying a boolean lattice \((\{0, 1\}, \preceq, 0, \lor, \land)\) such that \(0 \preceq 1\)
- Operator types \(\tau\) are defined by

\[
\tau := \alpha \mid \alpha \rightarrow \tau, \; \alpha \in \{0, 1\}
\]

- A *variable typing environment* \(\Gamma\) maps each variable in \(\mathcal{V}\) to a tier in \(\{0, 1\}\)
- An *operator typing environment* \(\Delta\) maps each operator \(\text{op}\) of arity \(n\) to a set \(\Delta(\text{op})\) of operator types of the shape \(\tau = \alpha_1 \rightarrow \ldots \alpha_n \rightarrow \alpha\)
\[
\frac{\Gamma(X) = \alpha}{\Gamma, \Delta \vdash X : \alpha} \quad (EV)
\]

\[
\frac{\Gamma, \Delta \vdash E_i : \alpha_i}{\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \alpha \in \Delta(\text{op})} \quad (EO)
\]

\[
\frac{\Gamma, \Delta \vdash \text{op}(E_1, \ldots, E_n) : \alpha}{\Gamma, \Delta \vdash X : 0} \quad (F)
\]

\[
\frac{\Gamma, \Delta \vdash \beta X := \text{fork}() : 0}{\Gamma, \Delta \vdash X : 0 \quad \Gamma, \Delta \vdash X : \alpha} \quad (W)
\]

\[
\frac{\Gamma, \Delta \vdash X : \alpha \quad \Gamma, \Delta \vdash E : \alpha' \quad E \in \text{Exp}}{\Gamma, \Delta \vdash \beta X := E : \alpha \quad \alpha \leq \beta \quad (CA)}
\]

\[
\frac{\Gamma, \Delta \vdash \beta C : \alpha \quad \Gamma, \Delta \vdash \beta C' : \alpha'}{\Gamma, \Delta \vdash \beta C; C' : \alpha \lor \alpha' \quad (CC)}
\]
\[ \Gamma, \Delta \vdash E : 1 \quad \Gamma, \Delta \vdash_{\beta} C : \alpha \]
\[ \Gamma, \Delta \vdash_{\beta} \text{while}(E)\{C\} : 1 \]  
(CW)

\[ \Gamma, \Delta \vdash E : \alpha \quad \Gamma, \Delta \vdash_{\beta} C : \alpha \quad \Gamma, \Delta \vdash_{\beta} C' : \alpha \]
\[ \Gamma, \Delta \vdash_{\beta} \text{if } E \text{ then } C \text{ else } C' : \alpha \]  
(CB)

\[ \Gamma, \Delta \vdash_{\beta} \text{skip} : \alpha \]  
(CS)

\[ \Gamma, \Delta \vdash_{\beta} C : 0 \]  
(CSub)

\[ \Gamma, \Delta \vdash_{\beta} C : 1 \]

\[ \Gamma, \Delta \vdash_{\beta} C : \alpha \quad \Gamma, \Delta \vdash X : \beta \]
\[ \Gamma, \Delta \vdash C; \text{return } X : \beta \]  
(P)
Some more intuitions

- The type discipline precludes values from flowing from tier 0 to 1 (but not command because of CS)
- Consequently, while loop guards are enforced to be of tier 1 (CW)
- In a (CB) rule the guard tier is equal to tier of both branches (could be weakened)
- However information may flow in the opposite direction (CA)
- The annotation $\beta$ keeps tract of the return type and is used by wait instructions (W)
Example:

found⁰ := ff⁰ : 0 ;
n¹ := length ( str¹ ) : 1 ;
l¹ := n/2¹ : 1 ;
x⁰ := fork ()⁰ : 0 ;
while l>0¹ do { 
  if x>0⁰ then 
    c⁰ := getchar ( str¹ , l¹ ) : 0 ;
  else 
    c⁰ := getchar ( str¹ , n−l¹ ) : 0 ;
  if c== '∗ '⁰ then 
    found⁰ := tt⁰ : 0 
  else skip : 0 ;
  l¹ := l−1¹ : 1
} : 1
if x>0⁰ then 
  sonf⁰ := wait ( x )⁰ : 0 ;
  found⁰ := or ( found , sonf )⁰ : 0
else skip : 0 ;
return found⁰
Neutral, positive and polynomial operators

An operator $\text{op}$ is:

1. **neutral** ($\text{op} \in Ntr$) if:
   1.1 either $\forall n \in \mathbb{N}^*, \lfloor \text{op} \rfloor(n) \subseteq W_{n-1}$.
   1.2 or there is a polynomial $P_{\text{op}}$ s.t. $\forall n \in \mathbb{N}^*$,
      $\exists V_{\text{op}} \subseteq W_n$,
      $\lfloor \text{op} \rfloor(n) \subseteq V_{\text{op}}$ and
      $\#V_{\text{op}} \leq P_{\text{op}}(n)$

2. **positive** ($\text{op} \in Pos$) if $\text{op} \notin Ntr$ and there is $c_{\text{op}} \in \mathbb{N}$
   s.t.:
   $$\forall \vec{d} \in W^m, \mid \lfloor \text{op} \rfloor(\vec{d}) \mid \leq \max_{i \in [1,m]} |d_i| + c_{\text{op}}$$

3. **polynomial** ($\text{op} \in Pol$) if $\text{op} \notin Ntr \cup Pos$ and there is a
   polynomial $Q_{\text{op}}$ s.t.:
   $$\forall \vec{d} \in W^m, \mid \lfloor \text{op} \rfloor(\vec{d}) \mid \leq Q_{\text{op}}\left( \max_{i \in [1,m]} |d_i| \right)$$
Safe operator typing environment

Definition

\( \Delta \) is safe if \( \forall \text{op} \in \text{dom}(\Delta) \) and 
\( \forall \alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \alpha \in \Delta(\text{op}) \) we have:

- if \( \text{op} \in \text{Ntr} \) then \( \alpha \leq \wedge_{i=1}^n \alpha_i \),
- if \( \text{op} \in \text{Pos} \) then \( \alpha = 0 \),
- if \( \text{op} \in \text{Pol} \) then \( \forall i \in [1, n], \alpha_i = 1 \) and \( \alpha = 0 \)

Intuitively:

- Neutral operators are iterable 
  (\( 1 \rightarrow 1 \) in a while loop guard).
- Positive operators are not iterable but composable
  (\( 0 \rightarrow 0 \) in a while-loop command).
- Polynomial operators are neither iterable nor composable 
  (\( 1 \rightarrow 0 \)).
Examples

\[(Ntr\ 1.1)\quad \lbrack \text{pred}\rbrack(u) = \varepsilon\quad \text{if}\ u = \varepsilon\]
\[ \quad = w\quad \text{if}\ u = a.w\]

\[(Ntr\ 1.2)\quad \lbrack ==\rbrack(u, w) = \top\top\quad \text{if}\ u = w\]
\[ \quad = \bot\bot\quad \text{otherwise.}\]

\[(Pos)\quad \lbrack \text{suc}\_d\rbrack(b) = d.b\]

\[(Pol)\quad \lbrack \text{calloc}\rbrack(u, w) = \underbrace{w \ldots w}_{|u|\ \times}\]

If $\Delta$ is a safe operator typing environment then :
\[\Delta(\text{pred}) \subseteq \{0 \rightarrow 0, 1 \rightarrow 1, 1 \rightarrow 0\},\]
\[\Delta(==) \subseteq \{1 \rightarrow 1 \rightarrow 1, \alpha \rightarrow \beta \rightarrow 0, \alpha, \beta \in \{0, 1\}\},\]
\[\Delta(\text{suc}\_d) \subseteq \{1 \rightarrow 0, 0 \rightarrow 0\},\]
\[\Delta(\text{calloc}) = \{1 \rightarrow 1 \rightarrow 0\}.\]
Safe process

Definition (Safe process)
Given $\Gamma$ a variable typing environment and $\Delta$ a operator typing environment, a process $P$ is a safe process if:

- $P$ is well-typed wrt $\Gamma$ and $\Delta$, i.e. $\Gamma, \Delta \vdash P : \beta$
- and $\Delta$ is safe

The search(str) program is safe wrt $\Gamma$, $\Delta$ provided.
Polynomial space

Theorem

The set of Pspace decision problems is exactly the set of problems decided by:

- safe,
- confluent,
- strongly normalizing,
- and lock-free processes \( P \)

Corollary

If \( P \) is a safe, confluent, strongly normalizing and lock-free processes \( P \) such that \( \Delta, \Gamma \vdash P : \textbf{1} \) then the function computed by \( P \) is in FPspace.
Intermediate Lemmata on tier 1 values

Lemma (Simple security)

*Given a safe process* \( P \) *wrt* \( \Gamma \) *and* \( \Delta \), *if* \( \Gamma, \Delta \vdash E : 1 \) *then* \( \forall X \in \mathcal{V}(E), \Gamma(X) = 1 \) *and all operators in* \( E \) *are neutral.*

Lemma (Bounded size)

*Given a safe process* \( P \) *wrt* \( \Gamma \) *and* \( \Delta \) *s.t.* \( \Gamma, \Delta \vdash E : 1 \), *for each store* \( \mu \), *if* \( \forall X \in \mathcal{V}(E), \mu(X) \in \mathbb{W}_n \) *and* \((E, \mu) \xrightarrow{\text{e}} d\) *then* \( d \in \mathbb{W}_n \).

Lemma (Bounded cardinality)

*Given a safe process* \( P \) *wrt* \( \Gamma \) *and* \( \Delta \) *and* \( \Gamma, \Delta \vdash E : 1 \), *the number of distinct values taken by* \( E \) *during the evaluation of* \((P, \mu)_\emptyset\) *is bounded polynomially in* \(|\mu|\).
Process tree

Definition (Process tree)
The process tree $T(\mathcal{E})$ of an environment $\mathcal{E}$ is defined by:

- the nodes are the configurations $\{\mathcal{E}_1, \ldots, \mathcal{E}_{\#\mathcal{E}}\}$
- the root is $\mathcal{E}_1$;
- for each $l \in [1, \#\mathcal{E}]$, there is an edge from $\mathcal{E}_l = (P, \mu)_\rho$ to $\mathcal{E}_k$, if $k \in \rho$.

Given a process tree $T$, its degree is denoted $d(T)$ and height $h(T)$.

$\rightsquigarrow$ pstree
Intermediate Lemmata on the process tree

**Lemma (Bounded degree)**

*Given a strongly normalizing and safe process* $P$, *there exists a polynomial* $Q$ *s.t.,* $\forall \mu$, *if* $(P, \mu) \not\to^* \mathcal{E}$ *then* $d(T(\mathcal{E})) \leq Q(|\mu|)$.

**Lemma (Bounded height)**

*Given a strongly normalizing and safe process* $P$, *there exists a polynomial* $Q$ *s.t.,* $\forall \mu$, *if* $(P, \mu) \not\to^* \mathcal{E}$ *then* $h(T(\mathcal{E})) \leq Q(|\mu|)$.

**Lemma (Subprocesses in polynomial time)**

*Given a strongly normalizing and safe process* $P$, *there is a polynomial* $Q$ *s.t.,* $\forall \mu$ *and* $\forall i \in \mathbb{N}$, *if* $(P, \mu) \not\to^i_k \mathcal{E}$ *then* $k \leq Q(|\mu|)$.

where $\not\to^i_k$ means that the $i$-th configuration has been evaluated $k$ times.
Pspace abiding strategy

Lemma (Bounded stores)

Given a strongly normalizing and safe decision process $P$, there exists a polynomial $Q$ such that, $\forall \mu$, if $(P, \mu)_{\emptyset} \rightarrow^* \mathcal{E}$ then $\forall i \leq |\mathcal{E}|$, if $\mathcal{E}_i = (P_i, \mu_i)_{\rho_i}$ then $|\mu_i| \leq Q(|\mu|)$.

Soundness.

We define a lazy Pspace abiding evaluation strategy that:

**Init** defines the current process to be the main process configuration

- executes the current process as long as possible
- on a wait instruction updates the current process to the waited process
- on a return instruction updates the current process to the father
Completeness

We write a safe process computing QBF:

- when an $\exists x \ (\forall)$ is encountered a fork instruction is called:
  - the son evaluates the remaining formula with $x$ set to $\top \top$
  - whereas the father evaluates the remaining formula with $x$ set to $\bot \bot$
  - at the end the father gets his son’s result and computes the disjunction with its own result (conjunction)

- A calloc operator is used in order to store the processes id.
Conclusions

- We have a characterization of Pspace combining non-interference and tiering methodologies
- The system is expressive (very close to C fork programs or Unix processes)
- It allows the programmer to simulate malloc/calloc operators (Polynomial operators)
- Possible extension on threads with creation