Static and dynamic typing for the termination of mobile processes

R. Demangeon, D. Hirschkoff, D. Sangiorgi

Plan

1. Insuring termination for π-calculus processes
2. Refining an existing type system for termination
3. A mixed type system for termination
4. Conclusion

Insuring termination of concurrent processes

Termination

- Termination is a desired property. Useful to prove soundness of programs.
- Several techniques to prove termination for sequential systems.

Termination and concurrency

- Work exists with a fixed number of threads
  - The Terminator Project : tool proving thread termination (used for Vista).
- Systems whose topology changes dynamically
  Termination of $\pi-$calculus processes.

The model we work with: the π-calculus

- $!p(x).P \mid \overline{p}\langle a\rangle.Q$
  - $!p(x).P$: server
  - $\overline{p}\langle a\rangle.Q$: client $\quad \overline{p}\langle a\rangle$ is the request
- Reduction: $!p(x).P \mid \overline{p}\langle a\rangle.Q \ \rightarrow\ !p(x).P \mid P_{[a/x]} \mid Q$
- Several requests: $!p(x).P \mid \overline{p}\langle a\rangle.Q \mid \overline{p}\langle b\rangle.R$

  more generally: many servers, many requests

The model we work with: the π-calculus

- $!p(x).P \mid \overline{p}\langle a\rangle.Q$
  - $!p(x).P$: server
  - $\overline{p}\langle a\rangle.Q$: client $\qquad \overline{p}\langle a\rangle$ is the request
- Reduction: $!p(x).P \mid \overline{p}\langle a\rangle.Q \;\rightarrow\; !p(x).P \mid P_{[a/x]} \mid Q$
- Several requests: $\qquad !p(x).P \mid \overline{p}\langle a\rangle.Q \mid \overline{p}\langle b\rangle.R$

  more generally: many servers, many requests
- Replication: source of divergence $\qquad \overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle$

  (we will sometimes use simply CCS processes $\qquad \overline{a} \mid !a.\overline{a}$)

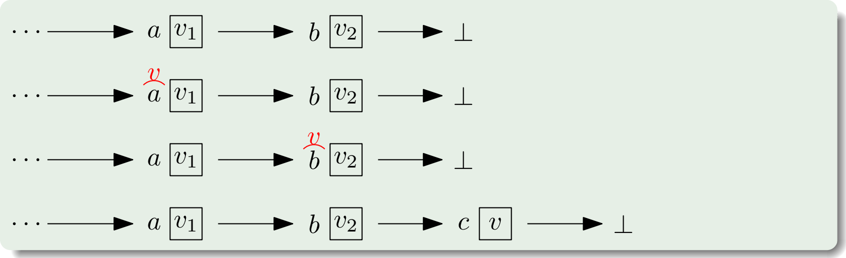
The model we work with: the π-calculus

- $!p(x).P \mid \overline{p}\langle a\rangle.Q$
  - $!p(x).P$: server
  - $\overline{p}\langle a\rangle.Q$: client $\quad \overline{p}\langle a\rangle$ is the request
- Reduction: $!p(x).P \mid \overline{p}\langle a\rangle.Q \rightarrow !p(x).P \mid P_{[a/x]} \mid Q$
- Several requests: $!p(x).P \mid \overline{p}\langle a\rangle.Q \mid \overline{p}\langle b\rangle.R$

  more generally: many servers, many requests
- Replication: source of divergence $\quad \overline{a}\langle b\rangle \mid !a(x).\overline{a}\langle x\rangle$

  (we will sometimes use simply CCS processes $\quad \overline{a} \mid !a.\overline{a}$)

A concurrent list structure: the symbol table



- Concurrent access.
- Dynamically evolving structure

A list structure: the symbol table

The symbol table in π

- the encoding of the symbol table in the π-calculus contains terms that look like

$$!p(a,b).a.(\overline{b} \mid \overline{p}\langle a,b\rangle)$$

- $!p(a,b)\ldots$ is a list cell constructor
- a and b are names of the same kind
- Firing the replication (i.e., visiting the next cell): we trade $\overline{a}$ for $\overline{b}$.

Type Systems for termination

Original systems

- [DengSangiorgi06] : 4 type systems.
- Weight based paradigm.

Typing $!p(x).P$

- In a context with other replications.
- First idea:
  - Assign levels to names
  - Free (not under a replication) outputs of the continuation have smaller levels.

    example: $!a(x).(\overline{d}\langle x\rangle \mid \overline{e}\langle x\rangle)$

Deng's System 4

- Constructed to handle structures like the symbol table.

  $$!p(a,b).a.(\overline{b} \mid \overline{p}\langle a,b\rangle)$$

- When a typable replication is fired:
  - Either the weight decreases,
  - Or the weight stays the same, but an ordering between names exchanged decreases.

Typing Rule for Replication

$$\frac{\Gamma \vdash P \qquad \textbf{either } (wt(\kappa) > wt(os(P)) \textbf{ or } (wt(\kappa) = wt(os(P)) \wedge \kappa\ \mathcal{R}_{mul}\ os(P))}{\Gamma \vdash !a_1(\widetilde{x_1}).\ldots.a_k(\widetilde{x_k}).P}$$

- $os(P)$: outputs in P not occurring under a replication.
- $wt(\kappa)$: weight of the input prefix.
- $\mathcal{R}_{mul}$: multiset extension of the ordering $\mathcal{R}$.

Deng's System 4

- Constructed to handle structures like the symbol table.

$$!p(a,b).a.(\overline{b} \mid \overline{p}\langle a,b\rangle)$$

- When a typable replication is fired:
  - Either the weight decreases,
  - Or the weight stays the same, but an ordering between names exchanged decreases.

Typing Rule for Replication

$$\frac{\Gamma \vdash P \qquad \textbf{either } (wt(\kappa) > wt(os(P)) \textbf{ or } (wt(\kappa) = wt(os(P)) \wedge \kappa\ \mathcal{R}_{mul}\, os(P))}{\Gamma \vdash !a_1(\widetilde{x_1}).\ldots.a_k(\widetilde{x_k}).P}$$

- $os(P)$: outputs in P not occurring under a replication.
- $wt(\kappa)$: weight of the input prefix.
- $\mathcal{R}_{mul}$: multiset extension of the ordering $\mathcal{R}$.

Typing the example above

- $P = !p(a,b).a.(\overline{b} \mid \overline{p}\langle a,b\rangle)$
  - With a and b of the same type (can be enforced).
  - Typable by setting $lvl(a) = lvl(b) = 2, lvl(p) = 1$ and by stating that $a > b$
- in presence of $\overline{p}\langle u,v\rangle \mid \overline{u}$, u is traded for v.
  The output on p forces $u > v$.

The limitations of this system

Handling tree structures

- $!p(a,l,r).a.(\overline{p}\langle a,l,r\rangle \mid \overline{l} \mid \overline{r})$
  - Natural extension of the previous structure.
  - Cannot be typed, as the global weight increases.

Remote allocation

- Remote allocation: new nodes are created somewhere, the structure is constructed elsewhere.
- $!p(a).p(b).!a.\overline{b} \quad | \quad (\nu u)(\overline{p}\langle u\rangle.P) \mid (\nu v)(\overline{p}\langle v\rangle.Q)$
- No way to enforce $u > v$ or $v > u$.
- Not typable

The limitations of this system

Handling tree structures

- $!p(a,l,r).a.(\overline{p}\langle a,l,r\rangle \mid \overline{l} \mid \overline{r})$
  - Natural extension of the previous structure.
  - Cannot be typed, as the global weight increases.

Remote allocation

- Remote allocation: new nodes are created somewhere, the structure is constructed elsewhere.
- $!p(a).p(b).!a.\overline{b} \quad | \quad (\nu u)(\overline{p}\langle u\rangle.P) \mid (\nu v)(\overline{p}\langle v\rangle.Q)$
- No way to enforce $u > v$ or $v > u$.
- Not typable

Contributions of this work

- We improve the *expressiveness* of type systems for π-calculus processes.
- A new, more refined static type system
  - handles tree-like structures
- a mixed system: combination of static and run-time analysis to avoid divergences
  - typing forms of remote allocation

Plan

Typing the terminating tree structure

Relevant parts of the code of a tree structure

$$
\begin{aligned}
T_0 \quad \stackrel{\text{def}}{=} \quad & !node(a,l,r,s,e).a(mode,v,ans).\\
& \text{if } mode = \text{search then}\\
& \quad \text{if } v = s \text{ then} \quad \overline{ans}\langle e\rangle \mid \overline{node}\langle a,l,r,s,e\rangle\\
& \quad \text{else} \quad \overline{l}\langle mode,v,ans\rangle \mid \overline{r}\langle mode,v,ans\rangle \mid \overline{node}\langle a,l,r,s,e\rangle\\
& \text{else } \ldots
\end{aligned}
$$

boils down to typing

$$P_1 = !p(a,l,r).a.(\overline{p}\langle a,l,r\rangle \mid \overline{l} \mid \overline{r})$$

A new type system

$$P_1 = !p(a, l, r).a.(\overline{p}\langle a, l, r\rangle \mid \bar{l} \mid \bar{r})$$

- System 4 in Deng's work: priority is given to the weight
  1/ weight of the process   2/ partial order between names
  when the weight remains the same
- hence P_1 cannot be typed

- We want to allow the weight to increase in some cases
- Main difficulty: weight increasings and decreasings should not compensate eachother.
  - $P_2 = !u.v.\bar{t}$. Seems typable if u, v, t have the same weight.
  - However $P_1 \mid P_2 \mid \overline{p}\langle t, u, v\rangle$ diverges.

Typing rules for replication

$$[\text{Rep1}]\ \frac{\mathcal{R} \vdash \kappa.P \qquad \exists l > 0 \text{ s.t. } \begin{cases} (i) & \forall j > l, M_\kappa|_j = os(P)|_j \\ (ii) & \forall j \geq l, rs(P)|_j = \emptyset \\ (iii) & os(P)|_l \subsetneq M_\kappa|_l \end{cases}}{\mathcal{R} \vdash !\kappa.P}$$

$$[\text{Rep2}]\ \frac{\mathcal{R} \vdash \kappa.P \qquad \exists l > 0 \text{ s.t. } \begin{cases} (i) & \forall j > l, M_\kappa|_j = os(P)|_j \\ (ii) & \forall j \geq l, rs(P)|_j = \emptyset \\ (iii) & \mathrm{card}(M_\kappa|_l) \leq \mathrm{card}(os(P)|_l) \\ (iv) & M_\kappa|_l\,(\mathcal{R}_\kappa)_{\mathrm{mul}}\, os(P)|_l \end{cases}}{\mathcal{R} \vdash !\kappa.P}$$

Explanations

- Two rules, when the weight decreases (as before) and when the weight increases.
- [Rep1] the weight decreases (more strict condition than in Deng's S4)
- [Rep2] the weight increases, the ordering decreases.

Handling name allocation

- Operator ν: name creation.
- Usages of ν have to be controlled.
- Danger: infinite decreasing chains in the partial order.

A diverging process

- $T_2 \stackrel{\text{def}}{=} \; !p(a,l,r).a.(\nu\ l_1,r_1)(\bar{l} \mid \bar{r} \mid \overline{p}\langle l,l_1,r_1\rangle)$
- We rule out such situations (clause (ii) in the above rules).

Typing the tree structure

$$
\begin{array}{ll}
T_0 \stackrel{\text{def}}{=} & !node(a, l, r, s, e).a(mode, v, ans). \\
& \text{if } mode = \text{search then} \\
& \quad \text{if } v = s \text{ then } \quad \overline{ans}\langle e\rangle \mid \overline{node}\langle a, l, r, s, e\rangle \\
& \quad \text{else } \quad \bar{l}\langle mode, v, ans\rangle \mid \bar{r}\langle mode, v, ans\rangle \mid \overline{node}\langle a, l, r, s, e\rangle \\
& \text{else } \ldots
\end{array}
$$

Typing the structure

T_0 can be type-checked with:

- $T_{a,l,r} = \sharp^3(\mathsf{M}, \mathsf{S}, T_{ans})$,
- $T_{ans} = \sharp^2(\mathsf{K})$,
- $T_{node} = \sharp^1_{\{(1,2),(1,3)\}}(T_a, T_a, T_a, \mathsf{S}, \mathsf{K})$,

Soundness of the type system

Soundness

If $\mathcal{R} \vdash P$ then P terminates.

Ideas of the proof

- Consider an infinite derivation from a typed process.
- Prove that such derivation must contain infinitely many communications involving replicated processes.
- By typability, a measure on processes necessarily decreases at each such step: contradiction.

Plan

1. Insuring termination for π-calculus processes
2. Refining an existing type system for termination
3. A mixed type system for termination
4. Conclusion

Handling remote allocation

An example of remote allocation

$!p(a).p(b).!a.b \quad | \quad (\nu u)(\overline{p}\langle u\rangle.P) \mid (\nu v)(\overline{p}\langle v\rangle.Q)$

- static type systems lead to technically complex solutions (e.g. dependent types)
- Solution: a mixed analysis
  - A static part to recognize some subprocesses as converging.
  - A dynamic part to monitor the execution of processes. The execution is aborted when a (potentially) dangerous loop arises.

The Dynamic system (I)

Assertions

- We add a new construct to the grammar of processes:

$$[a > b]P$$

- "a dominates b" according to a (global) partial order.
- the assertion guards the execution of P: the partial order is updated before P can proceed

Controlling loops

- Purpose: preventing a loop from arising.
- $!a.[a > b]P \mid !b.[b > a]Q \mid \overline{a}$.
  The two replications cannot be fired in a same reduction sequence.

The Dynamic system (II)

Semantics

- States of our system:
  - pairs $(P, \mathcal{R})$ with
    - P annotated process
    - $\mathcal{R}$ ordering, maintained along the execution.
  - $\bot$: aborted execution.
- Two new rules for assertions:
  - $([a > b]P, \mathcal{R}) \to (P, \mathcal{R} \cup (a, b))$ when $\mathcal{R} \cup (a, b)$ is an ordering
  - $([a > b]P, \mathcal{R}) \to \bot$ otherwise

The static part

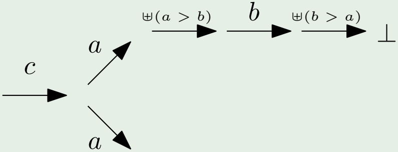
Principles

- modified type inference for a type system using only levels
  [DemangeonHirschkoffKobayashiSangiorgi07]
- if some replicated sub-process $!a.P$ cannot be typed (the weight does not decrease), add assertions ("the process terminates provided $a > b$")
  - by affecting the same level to every name, every check is done at run-time.
  - but having more annotations leads to more overhead at run-time
- we execute $(\llbracket P \rrbracket, \emptyset)$, where $\llbracket P \rrbracket$ is the annotated process

An ad-hoc example

$$S = !a.\overline{b} \mid !b.\overline{a} \mid !c.\overline{a} \mid a.\overline{f} \mid \overline{c}$$

- Static analysis:
  - we set $lvl(c) = 2$, $lvl(a) = lvl(b) = lvl(f) = 1$.
  - $!c.\overline{a}$ typable without assertions
  - $!a.\overline{b} \mid !b.\overline{a}$ is not
- Annotating S
  $[\![S]\!] = !a.[a > b]\overline{b} \mid !b.[b > a]\overline{a} \mid !c.\overline{a} \mid a.\overline{f} \mid \overline{c}$
- $([\![S]\!], \emptyset)$ may exhibit the following reduction sequences:



Soundness of the mixed type system

Soundness

If P is a π-calculus process, then $(\llbracket P \rrbracket, \emptyset)$ exhibits no divergence.

- Our analysis accepts all processes, including diverging ones. Executing the translation of the latter will yield $\bot$.
- Another theorem states that P and $\llbracket P \rrbracket$ have the same reductions unless the translated process reaches $\bot$.
- Our system is, of course, not complete. The translation of some terminating processes can reach $\bot$.

  e.g. $\llbracket !a.b.\overline{a} \mid \overline{a} \mid \overline{b} \rrbracket = !a.b.[a > a]\overline{a} \mid \overline{a} \mid \overline{b}$

Benefits of the mixed approach

Avoiding divergences

We can handle processes that contain diverging branches

- Non-determinism
- Dead code

Merging trees

In the paper, we describe the type-checking of a forest data-structure where trees that have been allocated at different sites can be merged.

Plan

Conclusion

Two proposals for type-based analysis for termination of concurrent systems.

- A (rather subtle) static type system
- A mixed approach with dynamic checks
  - implementation by an undergraduate student
  - suggestions for further improvements of the static analysis part (towards less annotations)
  - study how the run-time analysis could be performed in a distributed setting