Compositional Static Analysis for Implicit Join Synchronization in a Transactional Setting

Mai Thuong Tran, Martin Steffen, and Hoang Truong

University of Oslo, Norway
Vietnam National University, Việt Nam

SEFM 2013,

Madrid, Spain, 27. 09. 2013

Motivation

- software transactions: modern concurrency control mechanism
- proposed/being developed for a number of PLs
- enhanced performance + programmability
- price to pay: memory resource consumption

Resource consumption & SW transactions

- optimistic concurrency control: not “prevent” potential interference at the entry of a CR, but check and potentially repair/compensate/undo (potential) conflicts at the end
- conflict management (conflict detection + potential roll-back) $\Rightarrow$ info to reconstruct the original state needs to be stored.

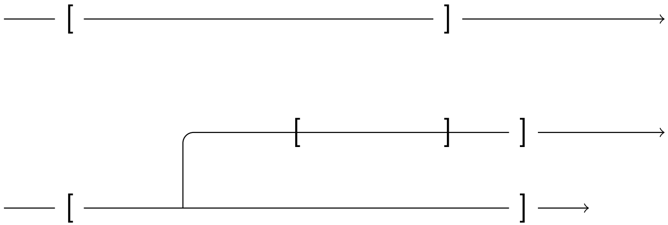
Model: Transactional Featherweight Java

- TFJ: formal proposal for Java + transactions [Jagannathan et al., 2005]
- transactions model:
  - nested
  - multi-threaded
  - non-lexical scope
- "inheritance" of the resource consumption of parent thread
- child threads: joining commit ⇒ implicit synchronization ⇒ main complication

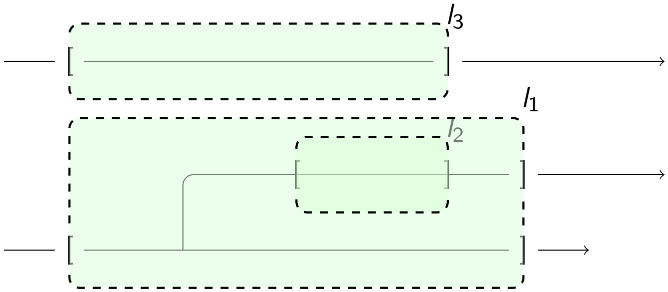
TFJ syntax

$$
\begin{array}{rcll}
P & ::= & \mathbf{0} \mid P \parallel P \mid p\langle e\rangle & \text{processes/thread} \\
L & ::= & \text{class } C\{\vec{f}{:}\vec{T}; K; \vec{M}\} & \text{class definitions} \\
K & ::= & C(\vec{f} : \vec{T})\{\text{this}.\vec{f} := \vec{f}\} & \text{constructors} \\
M & ::= & m(\vec{x}{:}\vec{T})\{e\} : T & \text{methods} \\
e & ::= & v \mid v.f \mid v.f := v \mid \texttt{if } v \texttt{ then } e \texttt{ else } e & \\
 & \mid & \texttt{let } x{:}T = e \texttt{ in } e \mid v.m(\vec{v}) & \text{expressions} \\
 & \mid & \texttt{new } C(\vec{v}) \mid \texttt{spawn } e \mid \texttt{onacid} \mid \texttt{commit} & \\
v & ::= & r \mid x \mid \texttt{null} & \text{values}
\end{array}
$$

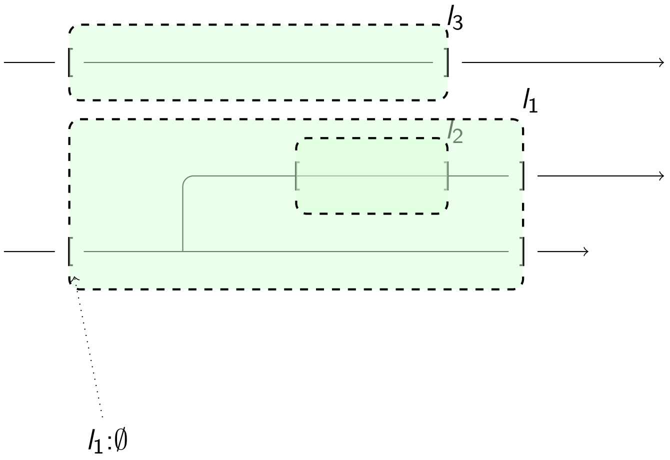
Nested and multi-threaded transactions



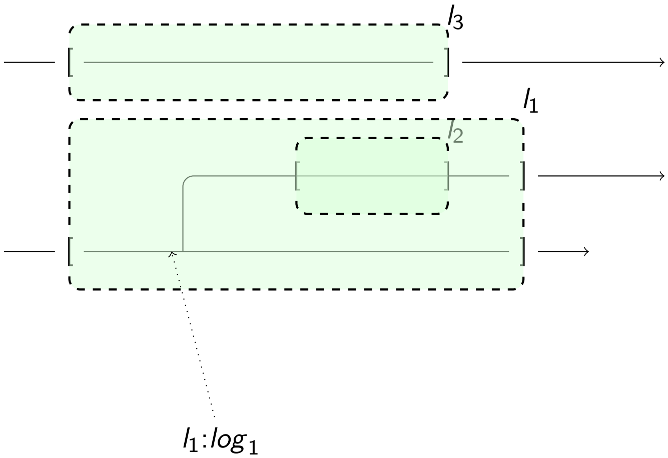
Nested and multi-threaded transactions



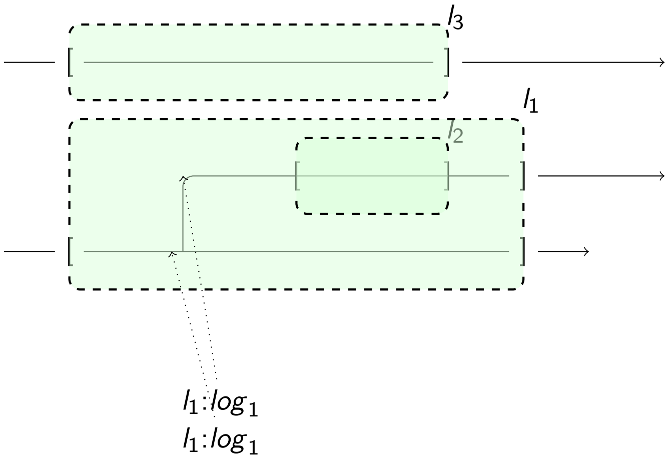
Nested and multi-threaded transactions



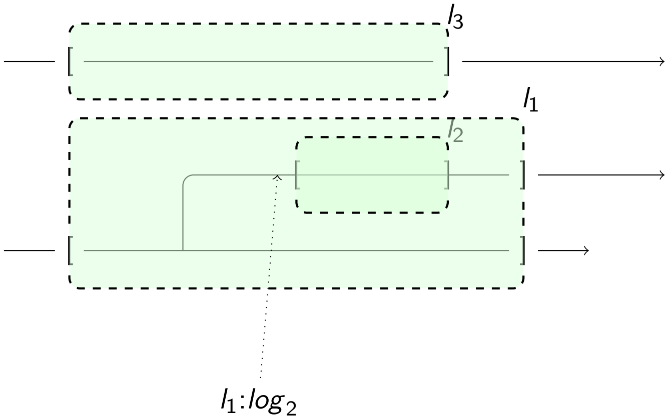
Nested and multi-threaded transactions



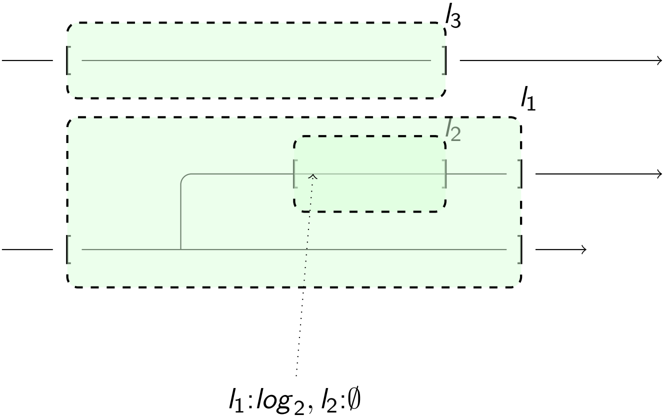
Nested and multi-threaded transactions



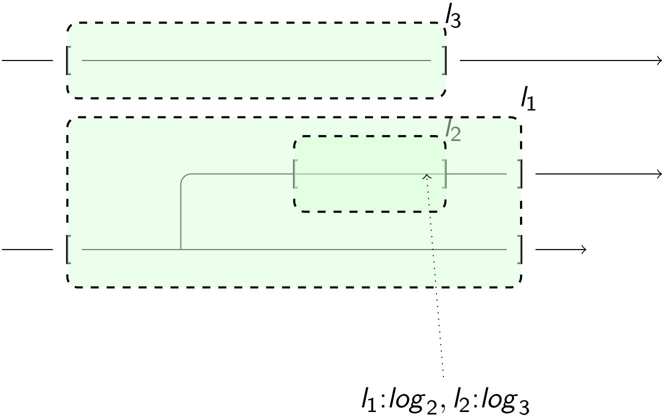
Nested and multi-threaded transactions



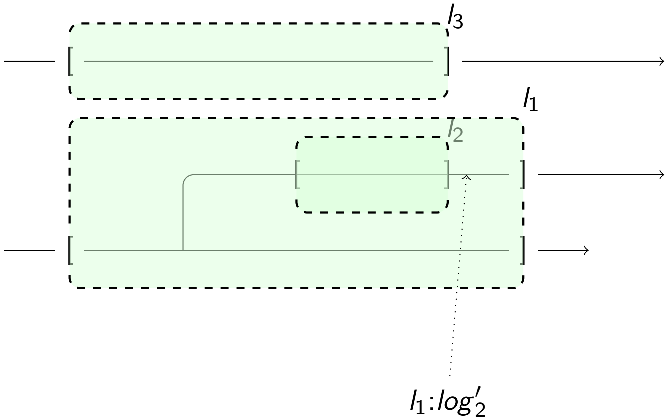
Nested and multi-threaded transactions



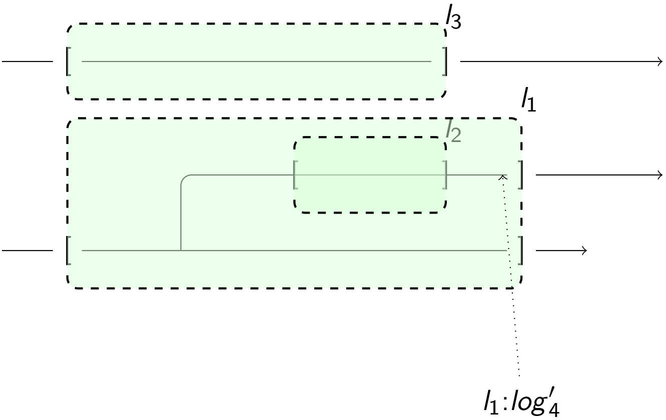
Nested and multi-threaded transactions



Nested and multi-threaded transactions



Nested and multi-threaded transactions



Goal & complications

Goal

Static estimation on upper bounds of resource consumption

- memory consumption = number of transactions potentially running at in parallel $\times$ local resource consumption
- challenges:
  - "concurrent" analysis ($\neq$ safe-commits ... iFM'10, FSEN'10 [Mai Thuong Tran and Steffen, 2010, Johnsen et al., 2012])
  - implicit join-synchronization via commits ($\neq$ "Resource bounds for components" (ICTAC'05, FMOODS'05 [Truong, 2005, Truong and Bezem, 2005] ...))
  - multithreading and nested transactions $\Rightarrow$ parent-child relationship between threads relevant

Challenges

- compositional, syntax directed analysis

⇒ "interface information"

- e.g., nesting depth (cf. "safe commit"):
  - "single threaded": pre and post are enough

$$n \vdash \texttt{commit} :: n-1$$

$$\frac{n_1 \vdash e_1 :: n_2 \qquad n_2 \vdash e_2 :: n_3}{n_1 \vdash e_1; e_2 :: n_3}$$

- parallel execution

Challenges

- compositional, syntax directed analysis

⇒ "interface information"

- e.g., nesting depth (cf. "safe commit"):
- parallel execution
  - $\parallel$ without synchronization

$$\frac{\vdash P_1 :: t_1 \qquad \vdash P_2 :: t_2}{\vdash P_1 \parallel P_2 : t_1 + t_2}$$

Challenges

- compositional, syntax directed analysis

$\Rightarrow$ "interface information"

- e.g., nesting depth (cf. "safe commit"):
- parallel execution
  - $\parallel$ without synchronization

$$\frac{\vdash P_1 :: t_1 \qquad \vdash P_2 :: t_2}{\vdash P_1 \parallel P_2 : t_1 + t_2}$$

  - ; *explicit* sequentialization/join

$$\frac{\vdash P_1 :: t_1 \qquad \vdash P_2 :: t_2}{\vdash P_1 ; P_2 : t_1 \vee t_2}$$

Challenges

- compositional, syntax directed analysis

⇒ “interface information”

- e.g., nesting depth (cf. “safe commit”):
- parallel execution

here:

- neither independent parallelism nor full sequentialization
- implicit join synchronization via commits

$$(\texttt{spawn}\ e_1); e_2$$

Seq. composition & Joining commit

```
onacid;                                        // thread 0 (main t
  onacid;
    spawn (e1;commit;commit);                  // thread 1
    onacid;
      spawn (e2;commit;commit;commit);         // thread 2
    commit;
    e3
  commit;
  e4;
```

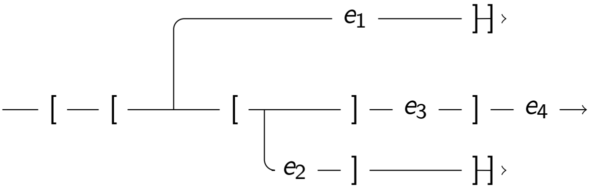
in the following:

`onacid` $\Rightarrow$ [

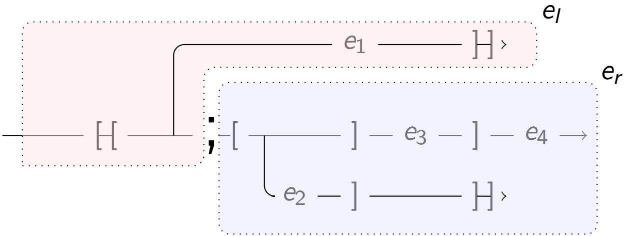
`commit` $\Rightarrow$]

$$
\begin{aligned}
e_1 &= [;[;[;\ldots;];];] = [^3;\ldots;]^3 \\
e_2 &= [^4;\ldots;]^4 \\
e_3 &= [^5;\ldots;]^5 \\
e_4 &= [^6;\ldots;]^6
\end{aligned}
$$

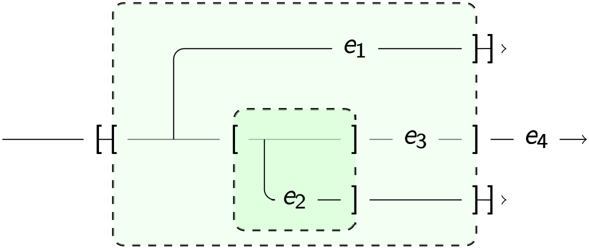
Seq. composition & Joining commit



Seq. composition & Joining commit



Seq. composition & Joining commit



Judgment & interface information

Judgment

$$n_1 \vdash e :: n_2, h, l, \vec{t}, S$$

- current thread
  - n_1 and n_2: balance, pre- and post-condition
  - h, l: maximum/minimum *during* execution
- not (only) current thread

compositionality

for **;** : S: contribution of *spawned* threads after execution of e

for $\|$: $\vec{t}$: sequence of *total* weights of current + spawned threads during e, separated by joining commits

Judgment & interface information

Judgment

$$n_1 \vdash e :: n_2, h, l, \vec{t}, S$$

- current thread
  - n_1 and n_2: balance, pre- and post-condition
  - h, l: maximum/minimum *during* execution
- not (only) current thread

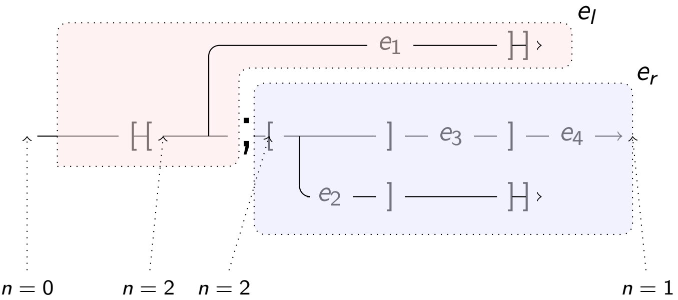
compositionality

for **;** : S: contribution of *spawned* threads after execution of e

for $\|$: $\vec{t}$: sequence of *total* weights of current + spawned threads during e, separated by joining commits

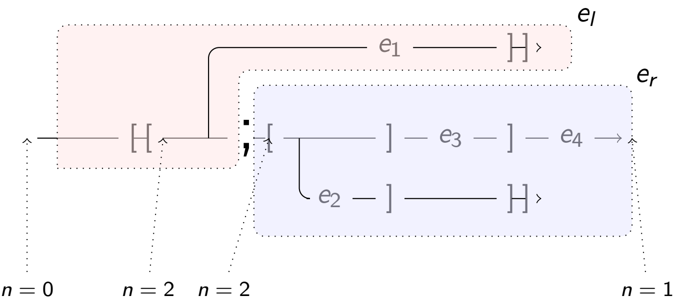
Sample derivation: pre- and post

$$\dfrac{\dfrac{\vdots}{0 \vdash [\ \ [\ ;\mathtt{spawn}\ (e_1\ ]\ \ ]\ )\ ::\ 2} \qquad \dfrac{\vdots}{2 \vdash [\ ;\mathtt{spawn}\ (e_2\ ]\ \ ]\ \ ]\ );\ ]\ ;e_3\ ]\ ;e_4\ ::\ 1}}{0 \vdash [\ \ [\ ;\mathtt{spawn}\ (e_1;\ ]\ \ ]\ )\ ;\ [\ ;\mathtt{spawn}\ (e_2;\ ]\ \ ]\ \ ]\ );\ ]\ ;e_3\ ]\ ;e_4\ ::\ 1}$$



Sample derivation (high and low)

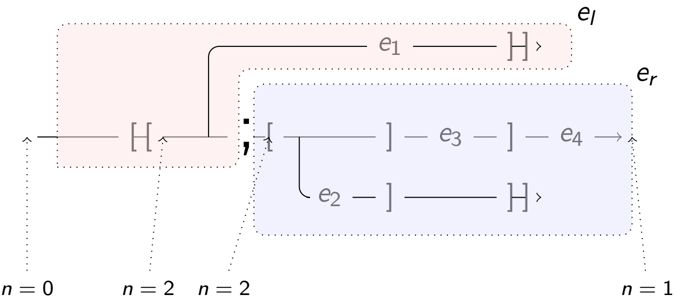
$$\dfrac{\dfrac{\vdots}{0 \vdash [\ \ [\ ; \texttt{spawn}\ (e_1\ ]\ \ ]\ )\ ::\ 2,0} \qquad \dfrac{\vdots}{2 \vdash [\ ; \texttt{spawn}\ (e_2\ ]\ \ ]\ \ ]\ );\ ]\ ; e_3\ ]\ ; e_4\ ::\ 7,1}}{0 \vdash [\ \ [\ ; \texttt{spawn}\ (e_1;\ ]\ \ ]\ )\ ;\ [\ ; \texttt{spawn}\ (e_2;\ ]\ \ ]\ \ ]\ );\ ]\ ; e_3\ ]\ ; e_4\ ::\ 7,0}$$



Sample derivation (par. contribution and synchronization)

$$\dfrac{\dfrac{\vdots}{0 \vdash [\ [\,;\texttt{spawn}\ (e_1\ ]\ \ ]\ ) :: [7], \{(2,3)\}} \qquad \dfrac{\vdots}{2 \vdash [\,;\texttt{spawn}\ (e_2\ ]\ \ ]\ \ ]\ );\ ]\,; e_3\ ]\,; e_4 :: [10,8], \{(1,0)\}}}{0 \vdash [\ [\,;\texttt{spawn}\ (e_1;\ ]\ \ ]\ )\ ;\ [\,;\texttt{spawn}\ (e_2;\ ]\ \ ]\ \ ]\ );\ ]\,; e_3\ ]\,; e_4 :: t, \{(1,0),(1,0)\}}$$

$$t = 7 \quad \vee \quad (10 + |\{(2,3)\}|) \quad \vee \quad (8 + |\{(1,0)\}|)$$



Sample derivation: different split

$$\dfrac{\dfrac{\vdots}{0 \vdash [\ ^2; \texttt{spawn}\ e_1;\ [\ ; (\texttt{spawn}\ e_2);\ ]\ :: [15], \{(2,3),(0,2)\}} \qquad \dfrac{\vdots}{2 \vdash e_3;\ ]\ ; e_4 :: [7,7], \{\}}}{0 \vdash [\ ^2; \texttt{spawn}\ e_1;\ [\ ; (\texttt{spawn}\ e_2);\ ]\ ; e_3\ ]\ ; e_4 :: 1, 7, 0, t, \{(1,0),(1,0)\}}$$

Sequential composition

$$\frac{\begin{array}{c} n_1 \vdash e_1 :: n_2, h_1, l_1, \vec{s}, S_1 \qquad n_2 \vdash e_2 :: n_3, h_2, l_2, \vec{t}, S_2 \\ h = h_1 \vee h_2 \quad l = l_1 \wedge l_2 \qquad p = n_2 - l_1 \qquad S = S_1 \downarrow_{l_2} \cup S_2 \quad \vec{u} = \vec{s} \oplus_p (S_1 \otimes_{n_2} \vec{t}) \end{array}}{n_1 \vdash \texttt{let } x{:}T = e_1 \texttt{ in } e_2 :: n_3, h, l, \vec{u}, S} \text{ T-LET}$$

Sequential composition

$$\frac{\begin{array}{c} n_1 \vdash e_1 :: n_2, h_1, l_1, \vec{s}, S_1 \qquad n_2 \vdash e_2 :: n_3, h_2, l_2, \vec{t}, S_2 \\ h = h_1 \vee h_1 \qquad l = l_1 \wedge l_2 \\ \vec{s} = s_1, \ldots, s_k \qquad \vec{t} = t_1, \ldots, t_m \qquad k, m \geq 1 \qquad p = n_2 - l_1 \\ t'_1 = t_1 + |S_1| \qquad t'_2 = t_2 + |S_1 \downarrow_{n_2-1}| \qquad t'_3 = t_3 + |S_1 \downarrow_{n_2-2}| \qquad \ldots \\ S = S_1 \downarrow_{l_2} \cup S_2 \\ \vec{u} = s_1, \ldots, s_{k-1}, s_k \vee t'_1 \vee \ldots \vee t'_p, t'_{p+1}, \ldots, t'_m \end{array}}{n_1 \vdash e_1; e_2 :: n_3, h, l, \vec{u}, S} \; \text{T-LET}$$

Parallel composition

- similarly complex
- merging trees / forests using join-commits-labels
- using *tree* representation of future joining commit behavior

Parallel composition

- similarly complex (“hidden” in def. of $\otimes$)
- merging trees / forests using join-commits-labels
- using *tree* representation of future joining commit behavior t_1 and t_2

$$\frac{\Gamma_1 \vdash P_1 : t_1 \qquad \Gamma_2 \vdash P_2 : t_2}{\Gamma_1, \Gamma_2 \vdash P_1 \parallel P_2 : t_1 \otimes t_2} \text{ T-PAR}$$

Results and future work

Soundness

Soundness of the analysis: "subject reduction"

- higher-order functions
- type inference
- machine checked proof of SR (Coq/OTT)
- different synchronization model

References I

[Jagannathan et al., 2005] Jagannathan, S., Vitek, J., Welc, A., and Hosking, A. (2005).
A transactional object calculus.
Science of Computer Programming, 57(2):164–186.

[Johnsen et al., 2012] Johnsen, E. B., Mai Thuong Tran, T., Owe, O., and Steffen, M. (2012).
Safe locking for multi-threaded Java with exceptions.
Journal of Logic and Algebraic Programming, special issue of selected contributions to NWPT'10.
available online 3. March 2012.

[Mai Thuong Tran and Steffen, 2010] Mai Thuong Tran, T. and Steffen, M. (2010).
Safe commits for Transactional Featherweight Java.
In Méry, D. and Merz, S., editors, *Proc. of the 8th Intl. Conf. on Integrated Formal Methods (iFM 2010)*, Lecture Notes in Computer Science, pages 290–304. Springer-Verlag.
An earlier and longer version has appeared as UiO, Dept. of Informatics Technical Report 392, Oct. 2009.

[Mai Thuong Tran et al., 2011] Mai Thuong Tran, T., Steffen, M., and Truong, H. (2011).
Estimating resource bounds for software transactions.
Technical report 414, University of Oslo, Dept. of Informatics.

[Truong, 2005] Truong, H. (2005).
Guaranteeing resource bounds for component software.
In Steffen, M. and Zavattaro, G., editors, *FMOODS '05*, volume 3535 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag.

[Truong and Bezem, 2005] Truong, H. and Bezem, M. (2005).
Finding resource bounds in the presence of explicit deallocation.
In *ICTAC'05*, volume 3722 of *Lecture Notes in Computer Science*, pages 227–241. Springer-Verlag.