A simulation tool for *tccp* programs

Leticia Lavado

joint work with

Laura Panizo María del Mar Gallardo

Dept. Lenguajes y Ciencias de la Computación

Andalucía Tech, University of Málaga

September 13, 2016

Motivation

Simulation and Verification of tccp programs

- Reactive and concurrent systems
- Application domains: automotive, trains, medical applications...
- Complex to model and analyse
  - Concurrency features
  - Synchronization
  - Critical properties
- Complex systems ➜ critical applications ➜ need to guarantee software safety and reliability
- *tccp* declarative language ➜ lack of simulation and analysis tools

Our approach

A simulation tool

- Design and implementation a tool for simulating *tccp* programs
  - Modular design and architecture
  - Based on an mechanism similar to abstract machines
  - **Simulation**: to define a *tccp* interpreter based on *tccp* operational semantics

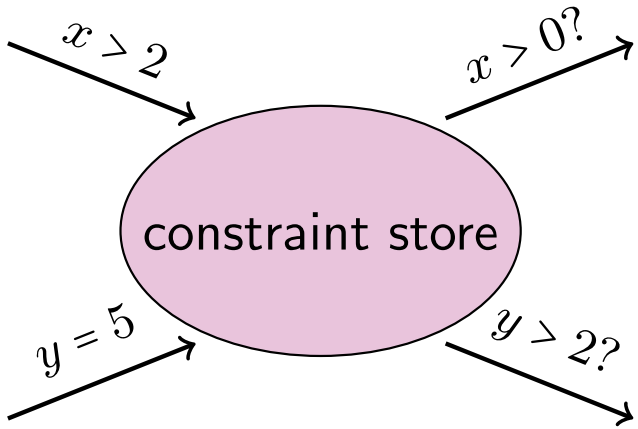
Outline

- Background: The *tccp* language
- Architecture of the proposal
- Implementation issues
- Evaluation
- Related Work
- Conclusions and Future Work

Background

The tccp language (de Boer et al. 2000)

- *tccp* is a timed extension of *ccp* (Saraswat 1993)
- parametric w.r.t. an underlying constraint system
- store-as-constraint paradigm
- computation = parallel execution of agents that add or ask information to/for a monotonic global constraint store



- time = global discrete clock

Background

The tccp language (de Boer et al. 2000)

- **tccp program** $P ::= D.A$ where
  - $D ::= \bigcup\{p(\vec{x}) :- A\}$
  - $A ::= \mathsf{stop} \mid \mathsf{tell}(c) \mid \sum_{i=1}^{n} \mathsf{ask}(c_i) \rightarrow A_i \mid A_1 \parallel A_2 \mid \exists x\, A \mid p(\vec{x}) \mid$
    $\mathsf{now}\ c\ \mathsf{then}\ A_1\ \mathsf{else}\ A_2$
- **small-step operational behaviour:**
  - $\langle A_1, c_1\rangle \rightarrow \langle A_2, c_2\rangle \rightarrow \langle A_3, c_3\rangle \rightarrow \langle A_4, c_4\rangle \ldots$
  - $\rightarrow$ is the transition relation (one-time unit)

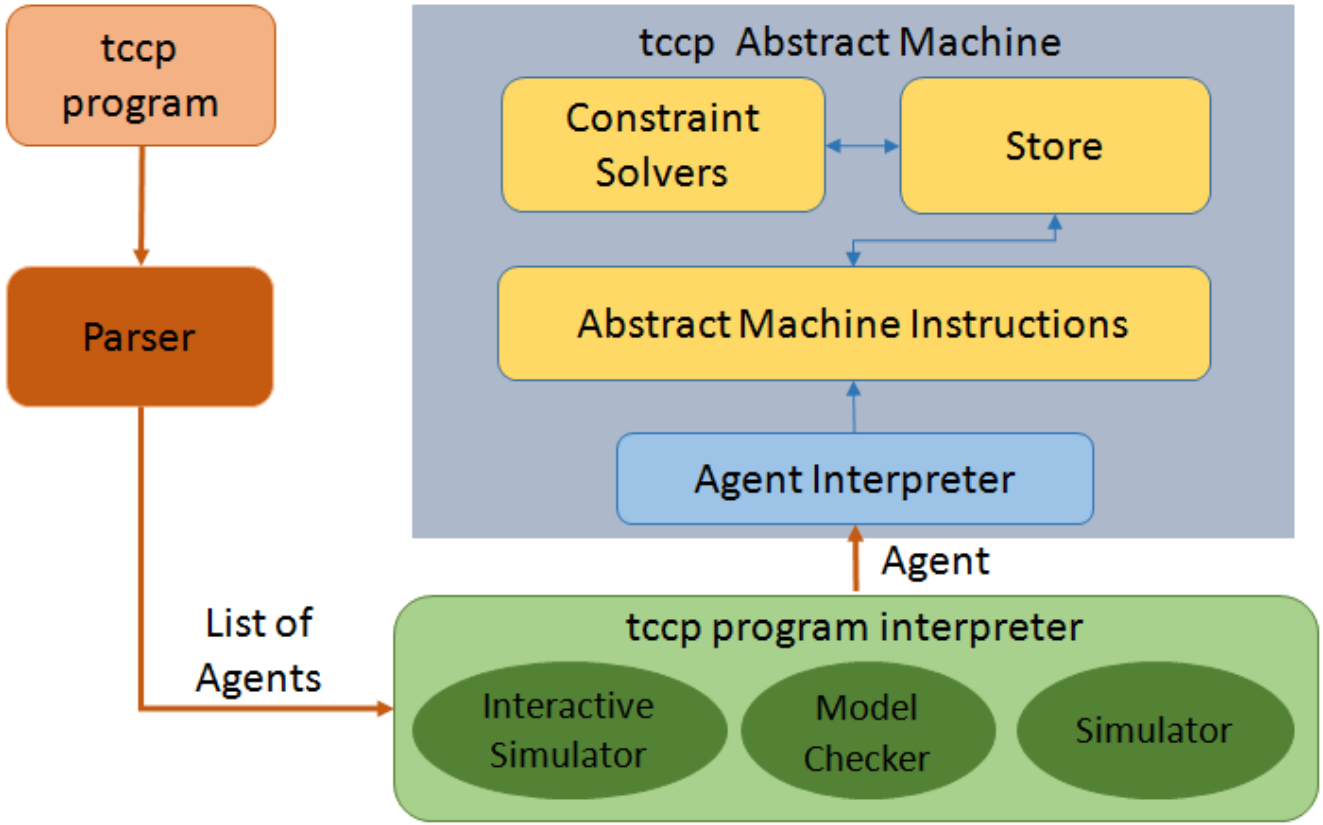
Example: modelling a photocopier

```
initialize(MIdle):- ∃ E,C,A,T(tell(A=[free|_])|| tell(T=[MIdle|_])
                    || tell(E=[off|_]) || system(MIdle,E,C,A,T)).

system(MIdle,E,C,A,T):- ∃ E',C',A',T'(tell(E=[_|E'])|| tell(A=[_|A'])
                    || tell(C=[_|C']) || tell(T=[_|T']) || user(C,A)
                    || ask(true) → photocopier(C,A',MIdle,T,E')
                    || ask(A'=[free|_])→ system(MIdle,E',C',A',T'))).

user(C,A):- ask(A'=[free|_])→tell(C=[on|_])
          || ask(A'=[free|_])→ tell(C=[off|_])
          || ask(A'=[free|_])→tell(C=[c|_])
          || ask(A'=[free|_])→ tell(true).
```

Example: modelling a photocopier

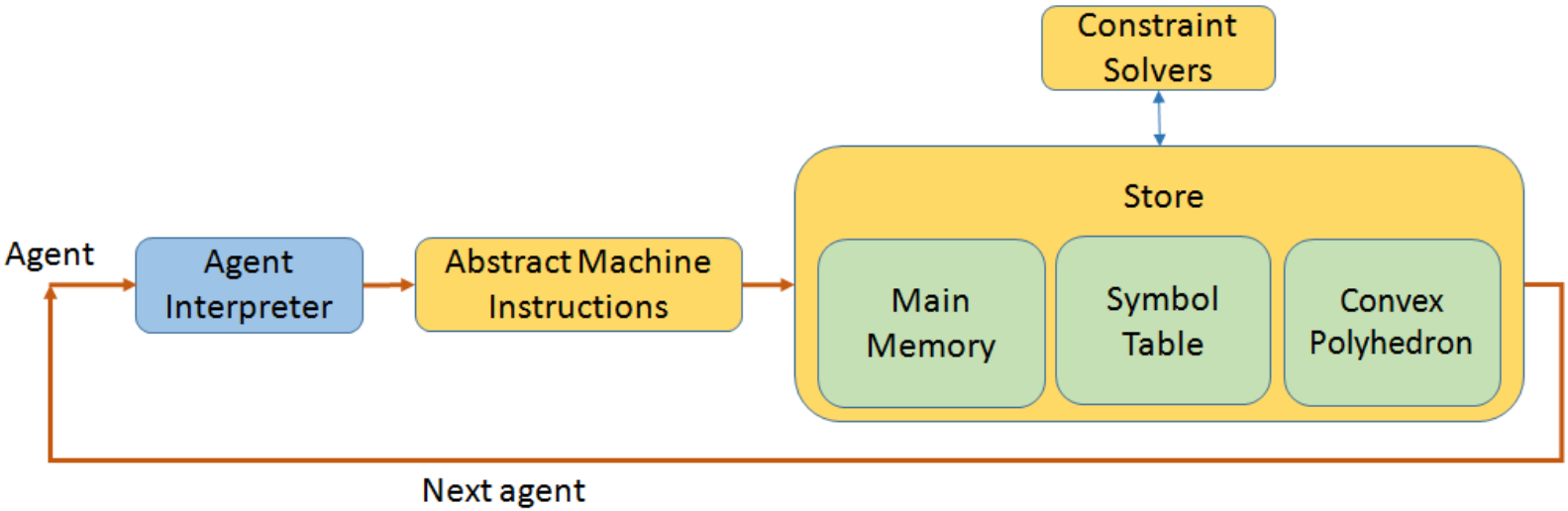
```
photocopier(C,A,MIdle,E,T):- ∃ Aux,Aux',T'(tell(T=[Aux|T'])
     || ask(true) →
          now(Aux>0) then
               now(C=[on|_]) then
                    tell(E=[going|_])
                    || tell(T'=[MIdle|_])|| tell(A=[free|_])
               else now(C=[off|_]) then
                         tell(E=[stop|_])
                         || tell(T'=[MIdle|_])|| tell(A=[free|_])
                    else now(C=[c|_]) then
                              tell(E=[going|_])
                              || tell(T'=[MIdle|_]) || tell(A=[free|_])
                         else tell(Aux'=Aux-1)
                              || tell(T'=[Aux'|_]) || tell(A=[free|_])
          else tell(E=[stop|_]) || tell(A=[free|_]).
```

Architecture of the proposal



Architecture of the proposal

Executing scheme



Architecture of the proposal

Abstract Machine instructions

Given $x \in Var$, and A be a *tccp* agent:

$A.x$ ➜ x in the scope of A

$p.\vec{x}$ ➜ formal parameters $\vec{x}$ of procedure p

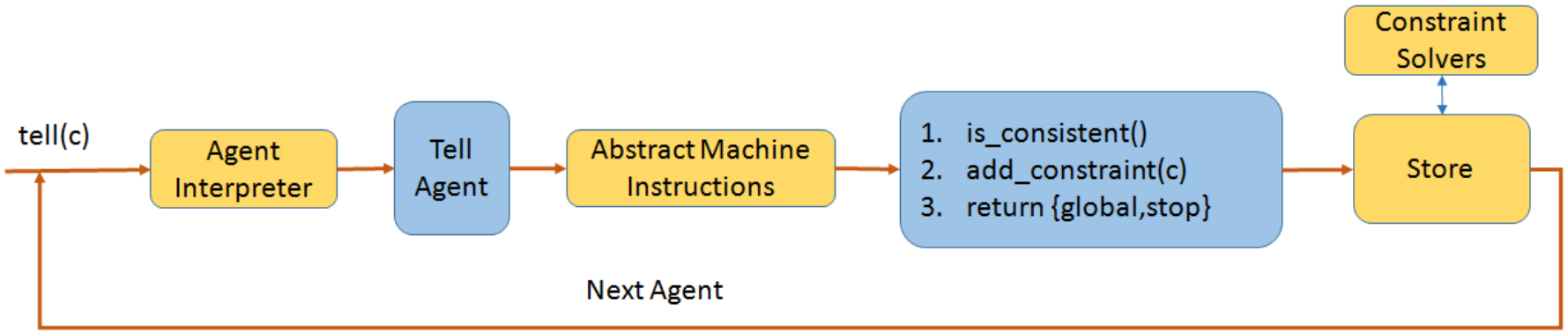
- $is\_consistent()$
- $add\_variable(A.x)$
- $add\_parameter(p.\vec{x}, \vec{x'})$
- $add\_constraint(A.c)$
- $entails(c)$
- $merge(local_1, local_2)$

Architecture of the proposal

Agent execution

Modifying main memory

- **tell:** $tell(c)$
  1. $is\_consistent()$
  2. $add\_constraint(c)$
  3. Return $\langle global, \text{stop} \rangle$



Architecture of the proposal

Agent execution

Modifying symbol table

- **hidding:** $\exists x\, A$
  1. $is\_consistent()$
  2. $add\_variable(A.x)$
  3. Return $execute(A)$
- **procedure call:** $p(\vec{x}) :- A$
  1. $is\_consistent()$
  2. $add\_parameter(p.\vec{x}, p.\vec{x'})$
  3. Return $\langle global, A \rangle$

Architecture of the proposal

Agent execution

No modifying the store

- **choice:** $\sum_{i=1}^{n} \text{ask}(c_i) \rightarrow A_i$
  1. $is\_consistent()$
  2. If $\neg entails(c_i)$ for $i = 1, \ldots, n$, $\rightarrow$ step 4, else select randomly $\text{ask}(c_i) \rightarrow A_i$ such that $entails(c_i) \rightarrow$ step 3
  3. Return $\langle global, A_i \rangle$
  4. Return $\langle global, \sum_{i=1}^{n} \text{ask}(c_i) \rightarrow A_i \rangle$
- **now:** now c then A else B
  1. $is\_consistent()$
  2. if $entails(c) \rightarrow execute(A)$, else $\rightarrow execute(B)$
- **parallel:** $A_1 || A_2$
  1. $is\_consistent()$
  2. $execute(A_1) = \langle local_1, nA_1 \rangle$
  3. $execute(A_2) = \langle local_2, nA_2 \rangle$
  4. Return $\langle merge(local_1, local_2), nA_1 || nA_2 \rangle$

Implementation issues

Main elements of the implementation

Tools and architecture entities

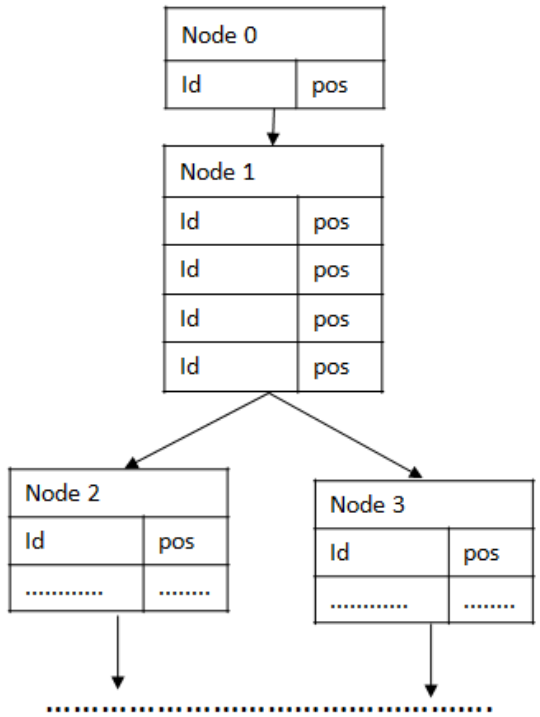
- Parsers (ANTLR)
- Interpreters *tccp* and agent interpreter (Java)
- Constraint Solvers (PPL for numeric constraint solver)
- Store ➜ symbol table and main memory (Java)

Implementation issues

Store implementation

Symbol table = scope of agents

- Tree structure
- Each node stores the list of variables belong to its scope
- Contains identifiers and references to main memory



Implementation issues

Store implementation

Main memory = information of all variables

- Available types: constant, discrete variable, expression, functor and reference
- Data field: depends on the type of element
- PPL convex polyhedron that keeps the numeric linear constraints

0	type	value
1	type	value
2	type	value
3	type	value
4	type	value
5	type	value
6	type	value
7	type	value
8	.....	.....
9		
10		

disc_poly

true

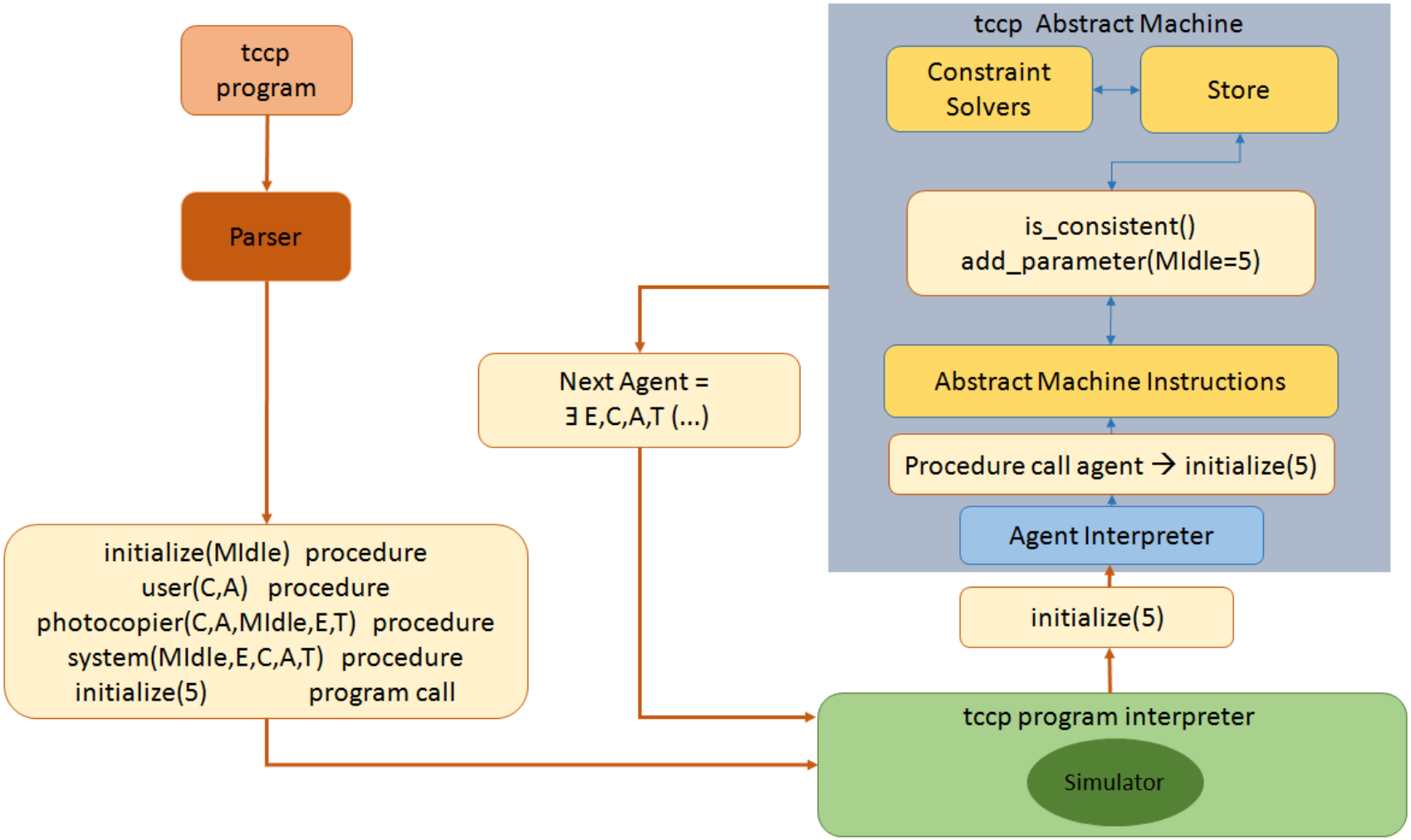
Implementation issues

Problems faced

- Logic, concurrent and synchronous nature of *tccp*
  - Store consistency - Constraint solving (logic and numeric constraints)
  - Dynamic generation of fine-grained procedures
  - Dynamic generation of local variables
  - Concurrency in the store - Parallel execution of agents

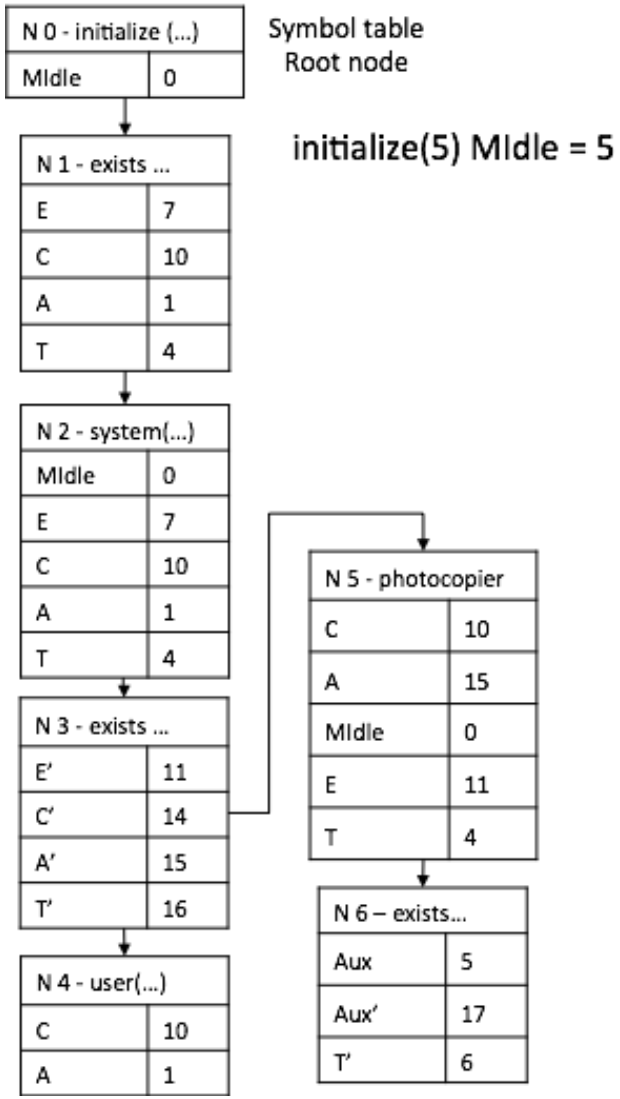
Evaluation

Running the photocopier program



Evaluation

Photocopier Example: Store state after 7 steps



Main Memory

0	DISC_VAR	D_0
1	FUNCTOR	2
2	CTE	free
3	REF	15
4	FUNCTOR	5
5	REF	0
6	REF	16
7	FUNCTOR	8
8	CTE	off
9	REF	11
10	FUNCTOR	12
11	UNDEF	-1
12	UNDEF	-1
13	REF	14
14	UNDEF	-1
15	FUNCTOR	20
16	FUNCTOR	18
17	DISC_VAR	D_1
18	REF	17
19	UNDEF	-1
20	CTE	free
21	UNDEF	-1

disc_poly

$D\_0 = 5 \wedge D\_1 = 4$

Evaluation

Performance statistics

	30 steps	**100 steps**	**500 steps**
Symbol Table (nodes)	26	85	417
Global Memory (regiters)	78	239	1169
disc_poly (dimensions)	6	6	6
Heap used (MB)	4	4.1	7
Heap allocated (MB)	16.3	16.3	16.3
Parser (ms)	192	196	197
Simulation (ms)	87	192	2,170

Related Work

Concurrent, declarative and synchronous languages

- declarative and synchronous character
  - Lustre (SCADE Suite)
  - SIGNAL (POLYCHRONY)
- concurrent logic programming
  - PARLOG
  - KL1
- tccp tools
  - Mozart-Oz
  - tccp Interpreter

Conclusions and Future Work

Contributions

- An abstract machine for *tccp*
- An implementation of a *tccp* simulator
- Available tool ➜ http://morse.uma.es/tools/tccp

Future work = extend to Hy-*tccp*

- Extend the abstract machine to Hy-*tccp*
- Extend *tccp* model checking algorithms to Hy-*tccp*
- Analysing reachability, safety and other properties
- Abstract interpretation based tools for diagnosis and analysis

Thanks for your attention!
Questions?