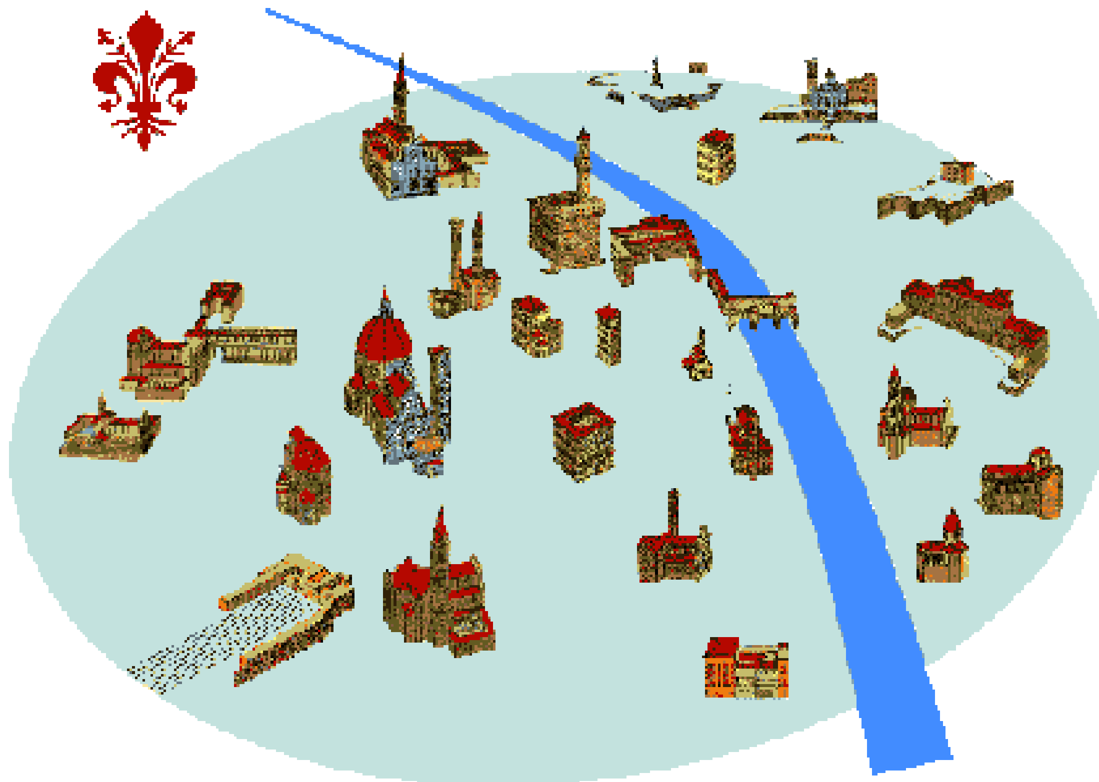
Decidability Results for Dynamic Installation of Compensation Handlers

Ivan Lanese
Computer Science Department
University of Bologna/INRIA
Italy

Joint work with Gianluigi Zavattaro

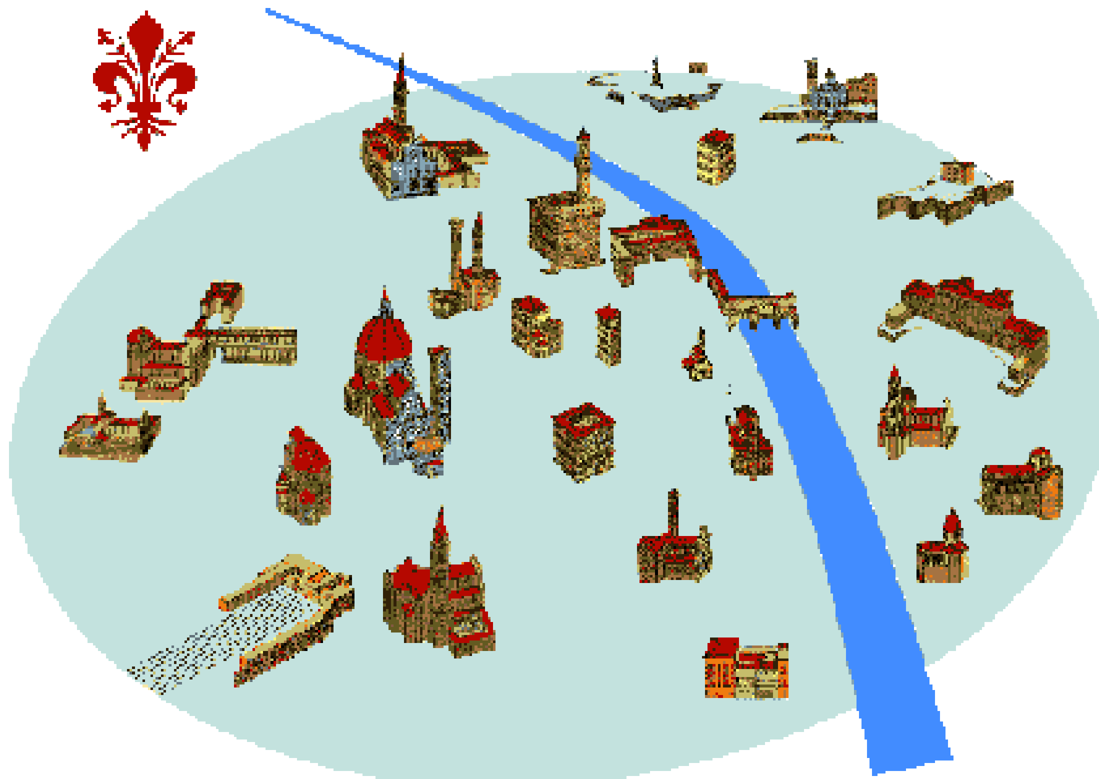
Map of the talk

- Long-running transactions
- Compensation installation
- Gap in the expressive power
- Conclusions



Map of the talk

- Long-running transactions
- Compensation installation
- Gap in the expressive power
- Conclusions



Handling unexpected events

- Current applications run in environments such as the Internet or smartphones
- Possible sources of errors
  - Communication partners may disconnect
  - Message loss
  - Received data may not have the expected format
  - Changes in the environment
  - ...
- Unexpected events should be managed so to ensure correct behavior even in unreliable environments

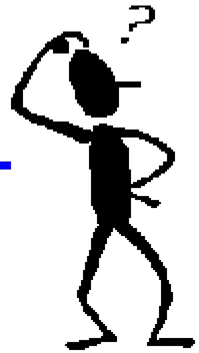
Compensation handling

- In service-oriented computing the concept of a long running transaction has been proposed
  - Computation that either succeeds or it aborts and is compensated
- The compensation needs to take back the system to a correct state
  - Undoing cannot always be perfect
  - Approximate rollback
- Programming compensations is a delicate task

Different primitives in the literature

- Long-running transactions used in practice
  - WS-BPEL, Jolie
- A flurry of proposals in the literature
  - Sagas, StAC, cjoin, SOCK, dcπ, webπ, …
- Are the proposed primitives equivalent?
- Which are the best ones?

A difficult problem



- Approaches to compensation handling can differ according to many features
  - Flat vs nested transactions
  - Automatic vs programmed abort of subtransactions
  - Static vs dynamic definition of compensations
- Approaches applied to different underlying languages
  - Differences between the languages may hide differences between the primitives

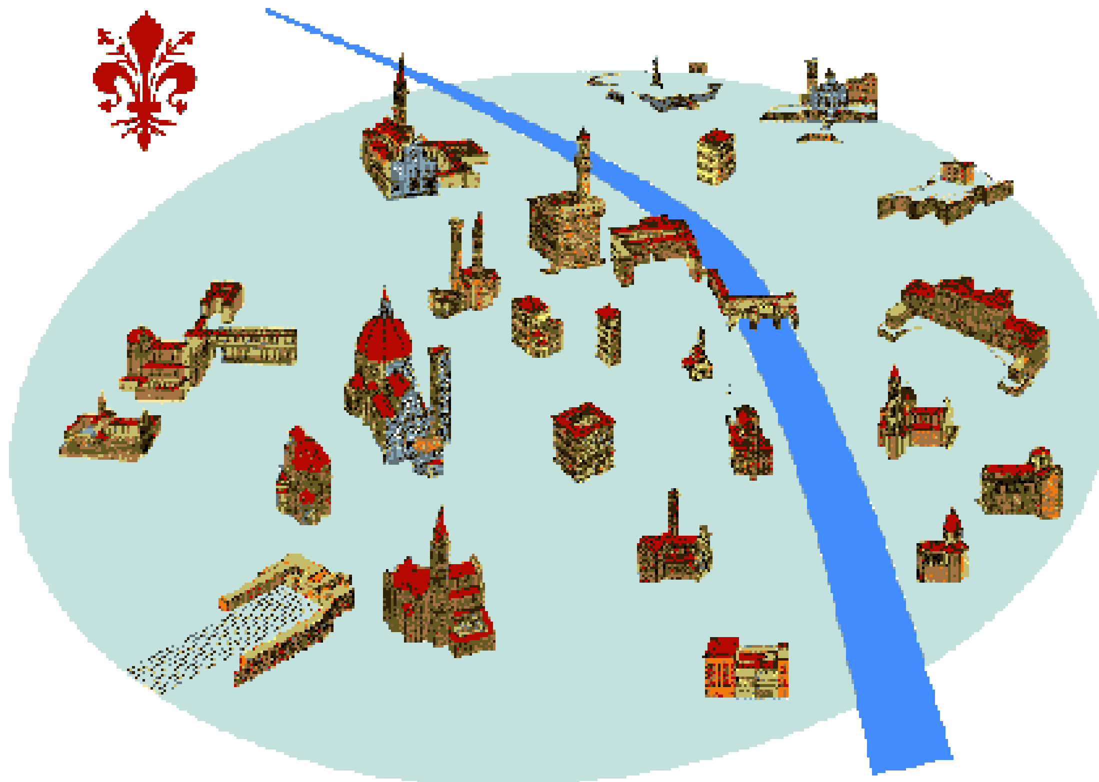
Our approach

- Taking the simplest possible calculus (π-calculus)
- Adding different primitives to it
- Comparing their expressive power

- Too many possible differences
- We concentrate on static vs dynamic definition of compensations
- Decidability of termination (all computations terminate) allows to discriminate them
  - In a π-calculus without restriction

Map of the talk

- Long-running transactions
- Compensation installation
- Gap in the expressive power
- Conclusions



Static compensations

- The compensation code is fixed
  - Java try P catch e Q
  - Q is the compensation for the already executed part of P
  - Q does not depend on when P has been interrupted
- First approach that has been proposed
- Still the most used in practice (WS-BPEL)
- Not flexible enough

Dynamic compensations

- The compensation can be updated during the computation
  - To take into account the changes in what has been done
- A primitive to define a new compensation is needed
  - The new compensation may possibly extend the old one

Syntax of the calculus

P	$::=$	0	inaction
		$\sum_i \pi_i . P_i$	guarded choice
		$!\,\pi . P$	guarded replication
		$P \mid Q$	parallel composition
		$t[P, Q]$	transaction
		$\langle P \rangle$	protected block
		$inst\lfloor \lambda X. Q \rfloor . P$	compensation update
		X	process variable

$$\pi \ ::= a(x) \quad \mid \quad \bar{a}\langle v \rangle$$

Simple examples

- Transactions can compute

$$\bar{a}\langle b\rangle | t[a(x).\, x.\, 0, Q] \rightarrow 0 | t[b.\, 0, Q]$$

- Transactions can be aborted

$$\bar{t} | t[a.\, 0, Q] \rightarrow \langle Q\rangle$$

- Transactions can commit suicide

$$t[\bar{t}.\, 0 | a.\, 0, Q] \rightarrow \langle Q\rangle$$

- Protected code is protected

$$t[\bar{t}.\, 0 | \langle a.\, 0\rangle, Q] \rightarrow \langle a.\, 0\rangle | \langle Q\rangle$$

Simple examples: compensation update

- Parallel update

$$t[inst\lfloor \lambda X.\, P|X \rfloor.\, a.\, 0, Q] \rightarrow t[a.\, 0, P|Q]$$

- Nested update (reverse order)

$$t[inst\lfloor \lambda X.\, b.\, X \rfloor.\, a.\, 0, Q] \rightarrow t[a.\, 0, b.\, Q]$$

- Replacing update

$$t[inst\lfloor \lambda X.\, 0 \rfloor.\, a.\, 0, Q] \rightarrow t[a.\, 0, 0]$$

Classes of calculi

- Dynamic compensations
- Nested compensations
- Parallel compensations
- Replacing compensations
- Static compensations

Classes of calculi

- Dynamic compensations
  - $\lambda X.\,Q$ is arbitrary
- Nested compensations
- Parallel compensations
- Replacing compensations
- Static compensations

Classes of calculi

- Dynamic compensations
- Nested compensations
  - Old compensation is preserved, inside a new context
  - $\lambda X.Q$ is linear
- Parallel compensations
- Replacing compensations
- Static compensations

Classes of calculi

- Dynamic compensations
- Nested compensations
- Parallel compensations
  - New compensation items can be added in parallel
  - $Q = Q'|X$ and Q' does not contain X
- Replacing compensations
- Static compensations

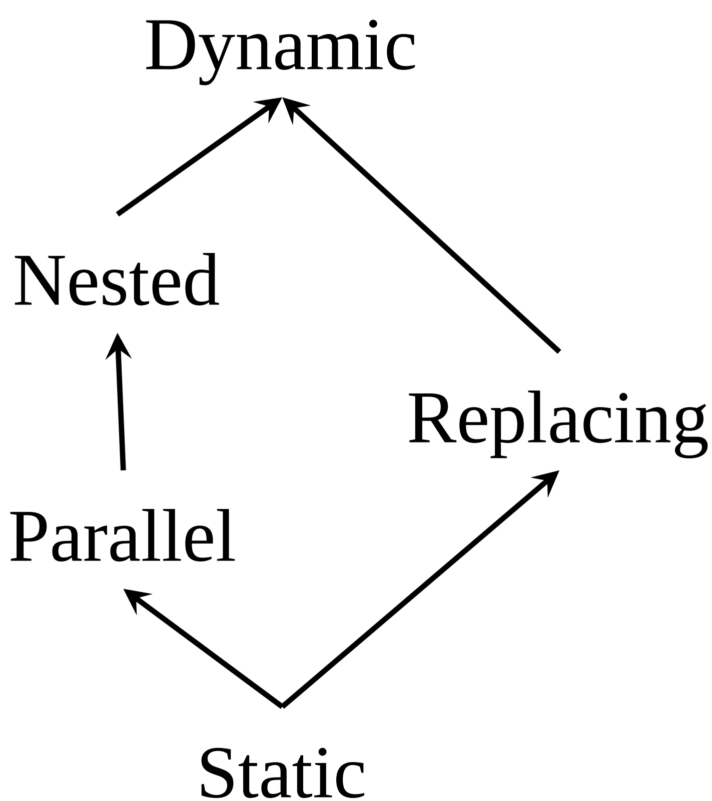
Classes of calculi

- Dynamic compensations
- Nested compensations
- Parallel compensations
- Replacing compensations
  - Old compensation is discarded
  - Q does not contain X
- Static compensations

Classes of calculi

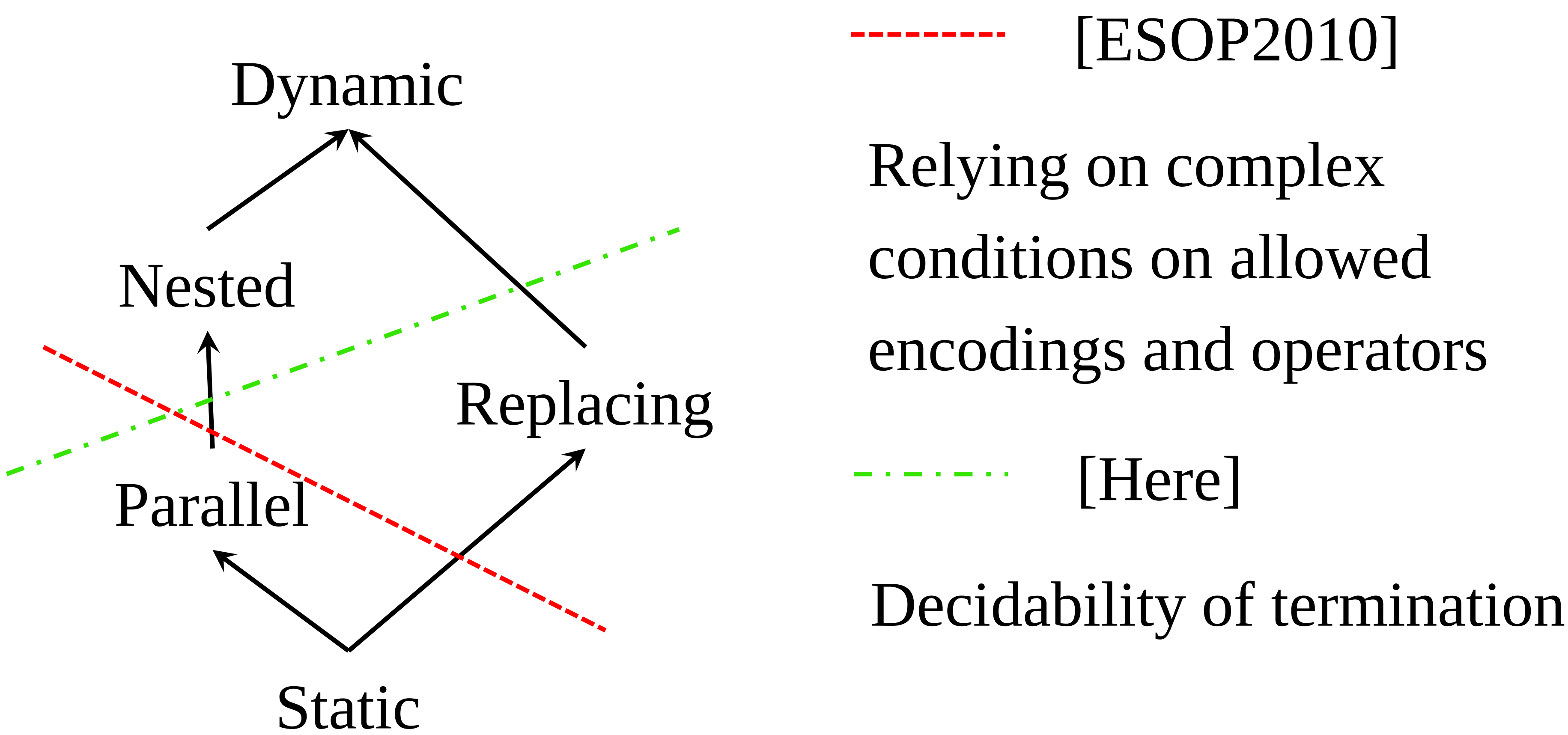
- Dynamic compensations
- Nested compensations
- Parallel compensations
- Replacing compensations
- Static compensations
  - Compensation updates are never used

A partial order



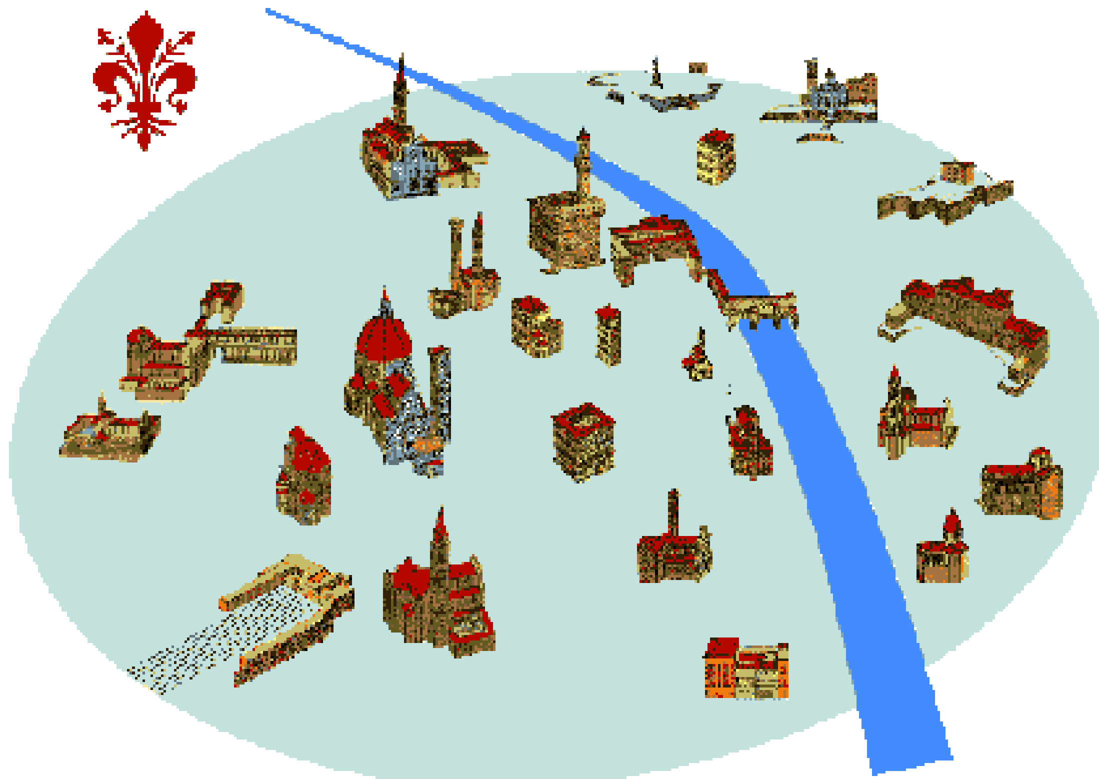
Are the inclusions strict?

A partial order



Map of the talk

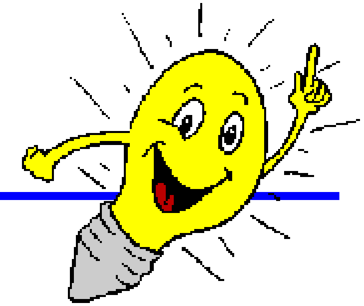
- Long-running transactions
- Compensation installation
- Gap in the expressive power
- Conclusions



Undecidability for nested compensations

- We prove that they can code RAMs
- RAMs are a Turing powerful model
  - Termination is undecidable
- A RAM includes
  - A set of registers containing non negative integers
  - A set of indexed instructions
- Two possible instructions
  - Inc(r_j): increment r_j and go to next instruction
  - DecJump(r_j,s): if r_j is 0 go to instruction s, otherwise decrement r_j and go to next instruction,
- A RAM terminates if an undefined instruction is reached

Encoding idea



- Instructions are replicated processes $!p_i.do\ i$
  - Can be triggered by an output on their name $\overline{p_i}$
- A register is a transaction of the form $r_j[Q, \bar{u}^n.\bar{z}]$
- Q contains the code for managing the register
- The increment instruction asks to increment the register using the compensation update $\lambda X.\bar{u}.X$
- The decrement instruction aborts the register
  - If a $\bar{z}$ becomes enabled, it recreates the register and jumps
  - Otherwise it recreates the register with one less $\bar{u}$ and goes to next instruction
- The encoding preserves termination

Decidability for parallel/replacing compensations

- We exploit the theory of Well-Structured Transition Systems (WSTS)
- Termination is known to be decidable for WSTS
- We just have to prove that for each process P its derivatives form a WSTS

Well Quasi Ordering (wqo)

- A reflexive and transitive relation $(S,\leq)$ is a wqo if given an infinite sequence $s_1,s_2,\ldots$ of elements in S, there exist $i<j$ such that $s_i \leq s_j$

Well-Structured Transition System

- $(S,\rightarrow,\leq)$ is a WSTS if
  – $(S,\rightarrow)$ is a finitely branching transition system
  – $(S,\leq)$ is a wqo
  – Compatibility: for every $s_1 \rightarrow s_2$ and $s_1 \leq t_1$ there exists $t_1 \rightarrow t_2$ such that $s_2 \leq t_2$

$$\begin{tikzcd}
s_1 \arrow[r, dash, bend left, "\leq"] \arrow[d] & t_1 \arrow[d, dashed] \\
s_2 \arrow[r, dash, dashed, bend left, "\leq"] & t_2
\end{tikzcd}$$

Idea of the proof

- Given a process P with parallel or replacing compensations in its derivatives
  - No new names are generated
  - The set of sequential subprocesses never increases
- This is not the case for nested compensations, since they allow to create infinitely many sequential processes
- The order in the next slide is a wqo thanks to Higman's lemma
- Compatibility holds
- Decidability follows from the theory of WSTS

Wqo on processes

$$P \equiv S | \prod_{i=1}^{n} t_i[P_i, Q_i]| \prod_{j=1}^{m} \langle R_j \rangle$$

$$\leq$$

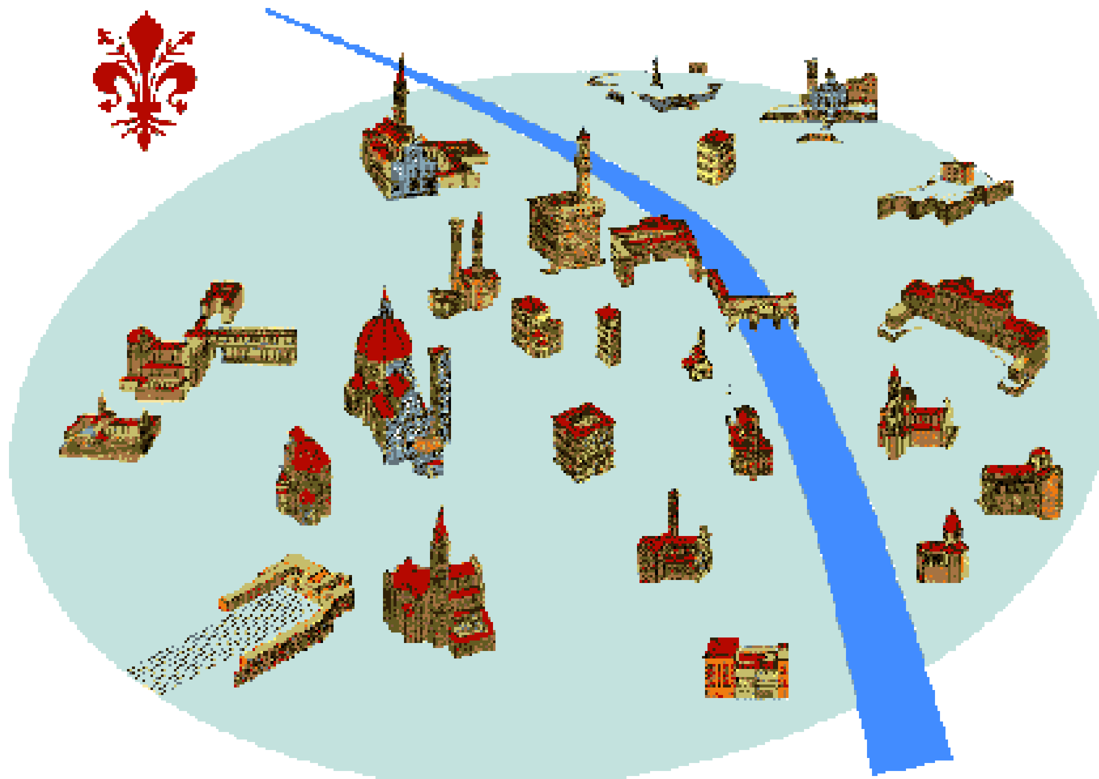
$$P' \equiv S|Q| \prod_{i=1}^{n} t_i[P'_i, Q'_i]| \prod_{j=1}^{m} \langle R'_j \rangle$$

if

$P_i \leq P'_i$ *and* $Q_i \leq Q'_i$ and $R_j \leq R'_j$

Map of the talk

- Long-running transactions
- Compensation installation
- Gap in the expressive power
- Conclusions



Summary

- We distinguished different forms of compensation installation
- We showed that decidability of termination allows to highlight a gap between
  - Dynamic and nested compensations on one side
  - Static, parallel and replacing compensations on the other side
- The result is robust
  - Different ways of managing subtransactions
  - The same holds for CCS with similar primitives
- Absence of restriction is fundamental

Future work

- Can we give termination preserving encodings of
  - Dynamic into nested compensations?
  - Parallel/replacing into static compensations?
- The full picture of the expressive power of primitives for long running transactions is still far
  - Other dimensions
  - Which is the impact of the underlying calculus?

End of talk

Thanks!

Questions?