In Search of Semantic Models for Reconciling Futures and Transactional Memory

Jingna Zeng, Paolo Romano, Luís Rodrigues, Seif Haridi, João Barreto
TM and intra-transaction parallelism

• TM has greatly matured over last decade:
  – hundreds of papers from academy and industry
  – hardware support in mainstream processors
  – integration in standard compilers

• Most literature assumes sequential execution of operations within transactions

• Can TM be used to exploit parallelism within transactions?
TM and parallel nesting

• Existing TMs that support intra-transaction parallelism offer *parallel nesting* abstraction:
  – fork-join semantics:
    • forking thread blocks till completion of nested txs

• To the best of our knowledge, no TM provides support for an alternative, more powerful abstraction:
  – the *future* abstraction
The Future abstraction

```java
Future<T> f = submit(task); // submit an asynchronous task
... // do something else
T x = f.eval(); // pick up task's result
```

Diagram:
- `future`
- `continuation`
- `f=submit(task)`
- `x=f.eval()`
How to support Futures in TM?

• Basic idea – *Transactional Future*:
  – allow transactions to submit/evaluate futures
  – futures run as transactions that:
    • can access shared variables
    • can return some result value
  – a future and its continuation appear as atomic units

• 2 key issues:
  – which serialization orders should be allowed between futures and continuations?
  – how to define the boundaries of a continuation?
Transaction Futures Semantics: a basic example

- Intuitively we want to guarantee atomicity between $T_F$ and its continuation...
Transaction Futures Semantics: a basic example

- ...but what are the expected serialization orders between $T_F$ and its continuation?
Transactional Futures Semantics: a basic example

- ...but what are the expected serialization orders between \( T_F \) and its continuation?
  - before \( T_F \)'s continuation: strongly ordered
Transactional Futures Semantics: a basic example

- ...but what are the expected serialization orders between $T_F$ and its continuation?
  - before $T_F$’s continuation: strongly ordered
  - either before or after $T_F$’s continuation: weakly ordered
How to support Futures in TM?

• Basic idea – *Transactional Future*:
  – allow transactions to submit/evaluate futures
  – futures run as transactions that:
    • can access shared variables
    • can return some result value
  – a future and its continuation appear as atomic units

• 2 key issues:
  – which serialization orders should be allowed between futures and continuations?
  – how to define the boundaries of a continuation?
How to define continuations?

• The Future abstraction enables parallel computations with complex dependency graphs, e.g.:
  – submitting futures from within continuations
  – escaping transactional futures
    • within the same top-level transaction, or
    • submitted and evaluated in different top-level transact.

• **Pro:** great flexibility for expert programmers

• **Con:** non-trivial to define continuations
Submission of a future by a continuation
Escaping transactional future

Here $T_{F1}$ returns the reference of $T_{F2}$ to $T0$, in order to allow $T0$ to evaluate $T_{F2}$.
Escaping transactional future

Logic underlying definition of $T_{F2}$ continuation:
Sequence of causally-related operations that leads from $T_{F2}$’s submission to its evaluation

• Continuation of $T_{F2}$ spans two transactional futures!
• $T_{F2}$ should observe both writes on $x$ and $y$ or none!
Transactional future escaping from its top-level transaction

$T_F$ is used as a communication means between T1 and T2.

T1 writes $T_F$’s reference in variable x and commits. This allows a different top-level transaction, e.g. T2, to evaluate $T_F$. 
Transactional future escaping from its top-level transaction

Logic underlying definition of \( T_F \) continuation:
Sequence of causally-related operations that leads from \( T_F \)’s submission to its evaluation

- Using the above rationale, a continuation can span two or more top-level transactions \( \Rightarrow \) *strongly atomic continuation*
- Constrain \( T_F \)’s continuation within the top-level tx that submitted \( T_F \) \( \Rightarrow \) *weakly atomic continuation*
How to formalize these concepts?

• Via the Future Serialization Graph:
  – similar in spirit to transaction serialization graph
  – but aimed to:
    1. allow for rigorous definition of futures and their continuations
    2. capture ordering relations between futures and continuations
Future Serialization Graph

Vertexes:

$V_{T^B}$: all ops since tx begin to first \{commit, abort, submit, eval\}

$V_{T^C}$: all ops since subm. of a future to first \{commit, abort, submit, eval\}

$V_{T^E}$: all ops since evaluation of a future to first \{commit, abort, submit, eval\}
Future Serialization Graph

Edges:

\( V_1 \rightarrow V_2 \), for each vertex \( V_1, V_2 \) in FSG s.t.:

- \( V_1 \) and \( V_2 \) are executed by the same thread \( t \) and \( t \) executes \( V_1 \) before \( V_2 \)
Future Serialization Graph

Edges:
For each transactional future $T$:
- $V_T^S \rightarrow V_T^B$: submission of a future precedes its execution
  (where $V_T^S$ is the vertex in FSG containing $T$’s submission)
- $V_T^B \rightarrow V_T^E$: evaluation of a future follows its execution
Future Serialization Graph

Edges:
For each strongly ordered transactional future $T$:
- $V_T^B \rightarrow V_T^C$: future precedes its continuation
FSG+

Extension of FSG to include read-after-write dependencies:

• captures causal relations among transactions that communicate futures’ references via shared variables.
Strongly Atomic Continuations

- Strongly atomic continuation of a tx future $T_F$:
  - set of vertexes that connect $V_{TF}^C$ to $V_{TF}^E$ in FSG+

![Diagram showing the strongly atomic continuation of a tx future $T_F$.](image-url)
Weakly Atomic Continuations

• Weakly atomic continuation of a tx future $T_F$:
  – set of vertexes in FSG+ that connect $V_{T_F}^C$ to $V_{T_F}^E$
  – constraint to the top-level tx that submitted $T_F$
Using the FSG+

- The FSG+ defines the transactional futures semantics by restricting the admissible serialization orders of transactions
- ...but can also be used by a graph-based concurrency control algorithm to ensure the desired semantics
Using the FSG+

• Intuition:
  – Enforce aciclicity of the FSG+ extended with the conflicts developed among transaction

• One important subtlety:
  – FSG+ can associate multiple vertexes to transactions and to futures/continuations:
    • if a conflict is developed from/to futures/transactions/continuations that include multiple vertexes in the FSG+
      ➔ add edges from/to all of the vertexes that they include
Summary

• Futures represent a powerful abstraction to:
  – exploit intra-transaction parallelism
  – enable new synchronization and communication patterns for transactional programming

• First attempt to define semantics of futures in a transactional context:
  – graph-based specification of alternative properties for:
    • serialization orders between futures and continuations
    • definition of continuations
Open research questions

• Is the current formalization complete?
• Can alternative formalisms be used?
• Which are the theoretical costs/complexity of the various semantics?
• Can these semantics be implemented efficiently in a practical system?
Thank you.
Questions?
Backup Slides
Inclusion of an operation in a transaction

• An operation \( op \) is \textit{included} in a transaction \( T \) if there is a path in the FSG:
  – from the vertex associated with the begin of \( T \)
  – to the vertex associated with the commit/abort of \( T \)
  – that passes via the vertex associated with \( op \)

• A transaction \( T \) includes:
  – all and only the operations issued by \( T \),
  – by any \textit{non-escaping} future submitted by \( T \),
  – and, recursively, by \( T \) ’s futures
Atomicity between top level transactions

- Let \( op \) be an operation included in a top-level transaction \( T \)
- Assume \( op \) develops a data dependency to/from an operation \( op' \) included in a different top-level transaction \( T' \)
- Add a data dependency edge from all the vertexes included in \( T \) to/from all the vertexes included in \( T' \)
Atomicity between futures and continuations

• Let \( op \) be an operation included in a transactional future \( T \)
• Assume \( op \) develops a data dependency to/from an operation \( op' \) included in the continuation of \( T \)
• Add a data dependency edge from all the vertexes included in \( T \) to/from all the vertexes included in the continuation of \( T \)