Reactive Synthesis for Declare Process Specifications

Luca Geatti[1] Marco Montali[2] Andrey Rivkin[2,3]

[1] University of Udine [2]Free University of Bozen-Bolzano [3]Danish Technical University

1. Declare process specifications

Declarative process specifications based on temporal constraints.

Conforming execution traces implicitly defined as those that satisfy all constraints.

- Pre-defined catalog of constraint templates defined on task placeholders;
  (cf. temporal patterns in software engineering, close similarity with trajectory constraints in planning)
- Every template described as an LTL_f formula over task placeholders;
- constraint grounds a template on specific tasks.

Table with Declare templates and their LTL_f formalization

Template	LTL_f formalization	Pastification (past(·))
existence(p)	$\mathsf{F}(p)$	$\mathsf{O}(p)$
absence2(p)	$\neg\mathsf{F}(p \wedge \mathsf{XF}(p))$	$\mathsf{H}(p \to \mathsf{ZH}(\neg p))$
choice(p,q)	$\mathsf{F}(p) \vee \mathsf{F}(q)$	$\mathsf{O}(p \vee q)$
exc-choice(p,q)	$(\mathsf{F}(p) \vee \mathsf{F}(q)) \wedge$ $\neg(\mathsf{F}(p) \wedge \mathsf{F}(q))$	$\mathsf{O}(p \vee q) \wedge$ $(\mathsf{H}(\neg p) \vee \mathsf{H}(\neg q))$
resp-existence(p,q)	$\mathsf{F}(p) \to \mathsf{F}(q)$	$\mathsf{H}(\neg p) \vee \mathsf{O}(q)$
coexistence(p,q)	$\mathsf{F}(p) \leftrightarrow \mathsf{F}(q)$	$(\mathsf{H}(\neg p) \vee \mathsf{O}(q)) \wedge$ $(\mathsf{H}(\neg q) \vee \mathsf{O}(p))$
response(p,q)	$\mathsf{G}(p \to \mathsf{F}(q))$	$q \,\mathsf{T}\, (\neg p \vee q)$
precedence(p,q)	$(\neg q)\, \mathsf{W}\, (p)$	$\mathsf{H}(q \to \mathsf{O}(p))$
succession(p,q)	$\mathsf{G}(p \to \mathsf{F}(q)) \wedge$ $(\neg q)\, \mathsf{W}\, (p)$	$p \,\mathsf{T}\, (\neg p \vee q) \wedge$ $\mathsf{H}(q \to \mathsf{O}(p))$
alt-response(p,q)	$\mathsf{G}(p \to \mathsf{X}((\neg p)\, \mathsf{U}\, q))$	$(p \vee q)\, \mathsf{T}\, (\neg p) \wedge \mathsf{H}(q \to \mathsf{Z}(q \,\mathsf{T}\, ((p \wedge \neg q) \to \mathsf{Z}(q \,\mathsf{T}\, \neg p))))$
alt-precedence(p,q)	$((\neg q)\, \mathsf{W}\, p) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}((\neg q)\, \mathsf{W}\, p))$	$\mathsf{H}(q \to \mathsf{O}(p)) \wedge$ $\mathsf{H}((q \wedge \neg p) \to$ $\mathsf{Z}(p \,\mathsf{T}\, (q \to (p \,\mathsf{T}\, (\neg p)))))$
alt-succession(p,q)	$\mathsf{G}(p \to \mathsf{X}((\neg p)\, \mathsf{U}\, q)) \wedge$ $((\neg q)\, \mathsf{W}\, p) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}((\neg q)\, \mathsf{W}\, p))$	past(alt-response(p,q)) $\wedge$ past(alt-precedence(p,q))
chain-response(p,q)	$\mathsf{G}(p \to \mathsf{X}(q))$	$\neg p \wedge \mathsf{H}(\mathsf{Y}(p) \to q)$
chain-precedence(p,q)	$\mathsf{G}(\mathsf{X}(q) \to p)$	$\mathsf{H}(q \to \mathsf{Z}p)$
chain-succession(p,q)	$\mathsf{G}(p \leftrightarrow \mathsf{X}(q))$	past(chain-response(p,q)) $\wedge$ past(chain-precedence(p,q))
not-coexistence(p,q)	$\neg(\mathsf{F}(p) \wedge \mathsf{F}(q))$	$\mathsf{H}(\neg p) \vee \mathsf{H}(\neg q)$
neg-succession(p,q)	$\mathsf{G}(p \to \neg\mathsf{F}(q))$	$\mathsf{H}(\neg p) \vee (\neg q)\, \mathsf{S}\, (p \wedge \neg q \wedge \mathsf{ZH}(\neg p))$
neg-chain-succession(p,q)	$\mathsf{G}(p \to \widetilde{\mathsf{X}}(\neg q)) \wedge$ $\mathsf{G}(q \to \widetilde{\mathsf{X}}(\neg p))$	$\mathsf{H}(\mathsf{Y}(p) \to \neg q) \wedge$ $\mathsf{H}(\mathsf{Y}(q) \to \neg p)$

Pastification: encoding of the template formula into a **pure-past** formula

2. Orchestration of Declare specifications

Standard assumption (<u>unrealistic</u>): all tasks under control of the orchestrator

- specification formula is the conjunction of all constraint formulae;
- specification formula encoded as a deterministic finite-state automaton (DFA);
- the DFA describes all and only the conforming traces;

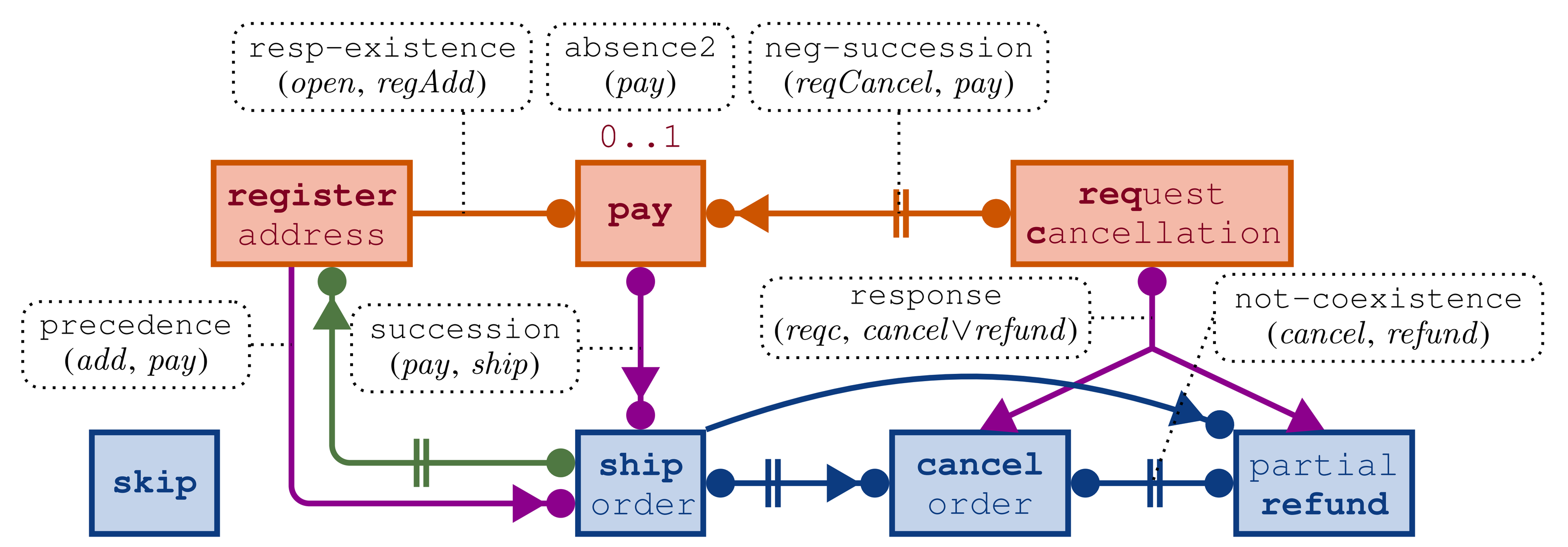
$\Longrightarrow$ Orchestration using the **DFA as execution engine**

In reality: some tasks under the responsibility of external, **uncontrollable actors**

- Tasks partitioned into Controller tasks and Environment tasks;
- still, environment is constrained by rules on its own tasks;
- so **two specifications**:
  - Assumption specification capturing what the environment can/cannot do;
  - Guarantee specification describing what the orchestration should satisfy.

$\Longrightarrow$ Orchestration as **assume-guarantee reactive synthesis**

3. An intuitive example



- Customer tasks vs Seller tasks
- Assumption specification: customer constraints and seller-customer constraints
- Guarantee specification: seller constraints and customer-seller constraints

5. Assume-guarantee synthesis for Declare

- Tasks Σ partitioned into **controllable actions** $\mathcal{C}$ and **uncontrollable actions** $\mathcal{U}$.
- Controller and Environment generate **simple finite traces**, alternating one action from $\mathcal{U}$ (chosen by Enrivonment) and one action from $\mathcal{C}$ (chosen by Controller).
- A **simple strategy** for Controller is a function $s : (\mathcal{U})^+ \to \mathcal{C}$ that, for every finite sequence of elements in $\mathcal{U}$, determines an element in the set $\mathcal{C}$.

Realizability of LTL_f formula ϕ over simple traces: for every infinite alternation of moves, there exists an index k such that the simple trace from 0 to k satisfies ϕ.

Given:

- Declare specification ϕ_E for Environment,
- Declare specification ϕ_C for Controller,

Realizability of (ϕ_E, ϕ_C): is $\phi_E \to \phi_C$ is *realizable* over simple finite traces?

Reactive synthesis of (ϕ_E, ϕ_C): if (ϕ_E, ϕ_C) realizable, compute a simple strategy.

6. Naive procedure (2-EXPTIME)

Define two special LTL_f formulae to enforce simple traces:

- $simple_{\mathrm{Con}}(\mathcal{C})$ - Controller plays at odd time points;
- $simple_{\mathrm{Env}}(\mathcal{U})$ - Environment plays at even time points.

Then (ϕ_E, ϕ_C) is realizable if and only if the LTL_f formula

$$simple_{\mathrm{Con}}(\mathcal{C}) \wedge ((simple_{\mathrm{Env}}(\mathcal{U}) \wedge \phi_E) \to \phi_C)$$

is realizable in the classical sense.

Naive procedure:

1. Build formula $simple_{\mathrm{Con}}(\mathcal{C}) \wedge ((simple_{\mathrm{Env}}(\mathcal{U}) \wedge \phi_E) \to \phi_C)$;
2. feed this formula into a classical algorithm for LTL_f realizability/synthesis.

$\Longrightarrow$ Doubly exponential time procedure.

6. Optimized procedure using pastification (EXPTIME)

- Premise: realizability of pure-past LTL_f formula is in (single!) EXPTIME.
  (Since "the past already happened", pure-past formulae can be encoded into DFAs with a singly exponential blow-up)
- Pastification past(ϕ) of LTL_f formula ϕ: encoding into pure-past LTL_f. (always doable, exponential blow-up)
- **Key result**:
  Every Declare template formula is pastifiable in linear time!
  (see Table on the top-left)
- Also:
  $simple_{\mathrm{Con}}(\mathcal{C})$ and $simple_{\mathrm{Env}}(\mathcal{U})$ are pastifiable in linear time.

$\Longrightarrow$ Singly exponential time procedure
$\downarrow$

Algorithm 1: *synth*-DECLARE

```
input : Σ = C ∪ U
input : environment specification φ_E
input : controller specification φ_C
ψ_SimpleCon ⟵ past(simple_Con(C));
ψ_SimpleEnv ⟵ past(simple_Env(U));
ψ_E ⟵ past(φ_E);
ψ_C ⟵ past(φ_C);
A ⟵ build-DFA(ψ_SimpleCon ∧ ((ψ_SimpleEnv ∧ ψ_E) → ψ_C));
s ⟵ realize-and-extract(A);
if s has been found then
    return s;
else
    return unrealizable.
```

7. Symbolic approach

We provide a novel algorithm to encode a pure-past LTL_f formula ϕ into a *linear-size, fully symbolic DFA* $\mathcal{S}(\phi)$.

We exploit this to obtain a symbolic version of Algorithm 1:

1. apply pastification as in Algorithm 1 obtaining the overall formula Φ;
2. encode Φ into its symbolic DFA $\mathcal{S}(\phi)$.
3. decide realizability by solving a *symbolic reachability game* over $\mathcal{S}(\phi)$.

Differently from classical reachability games, this symbolic version can be solved efficiently through manipulation of the Boolean formulas that define $\mathcal{S}(\phi)$, by computing the *strong preimage* of the transition formula until a fixpoint is reached.