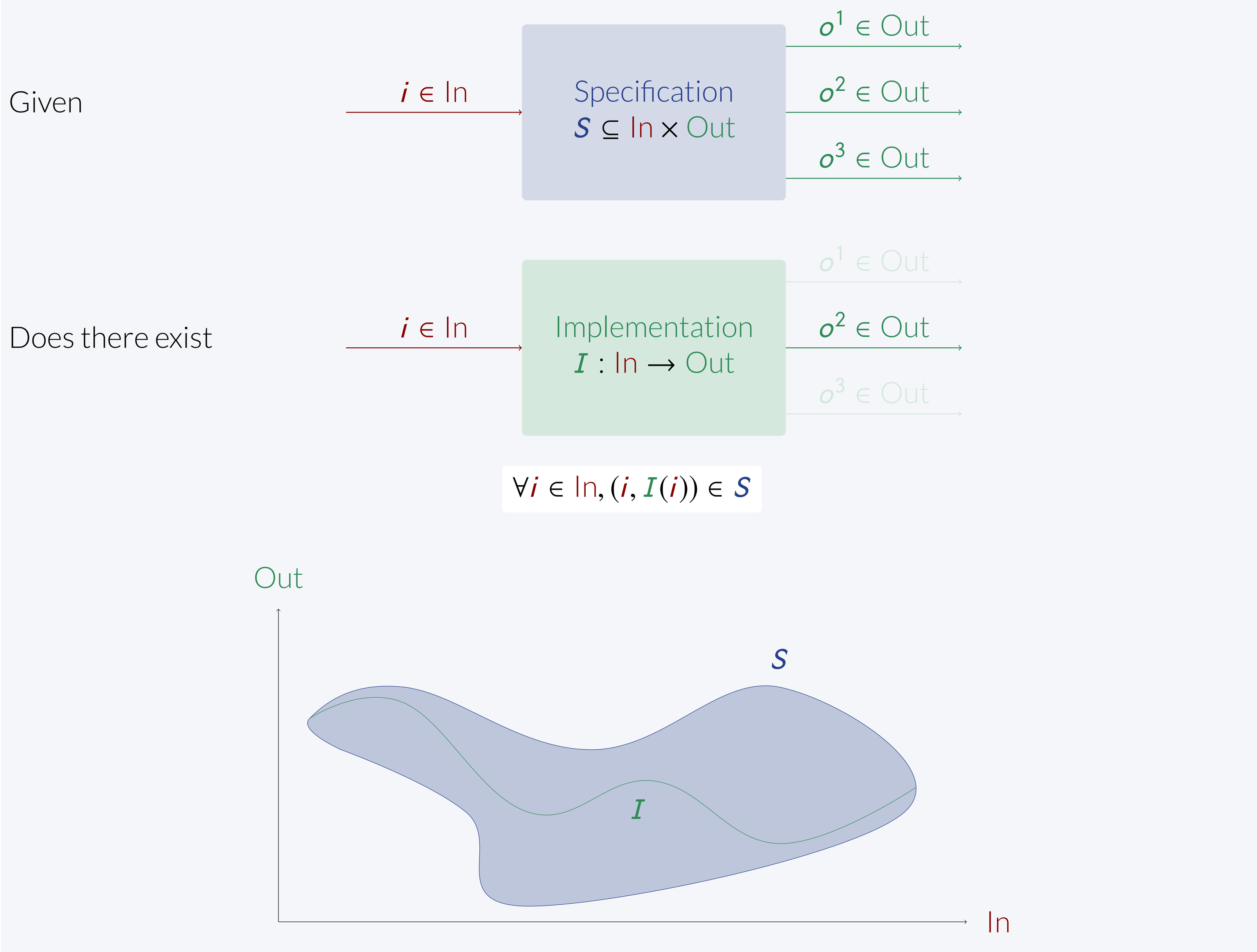
Reactive Synthesis of Systems over Data Words

Léo Exibard [1 2] Emmanuel Filiot [1] Pierre-Alain Reynier [2]

[1]Méthodes Formelles et Vérification
Université libre de Bruxelles

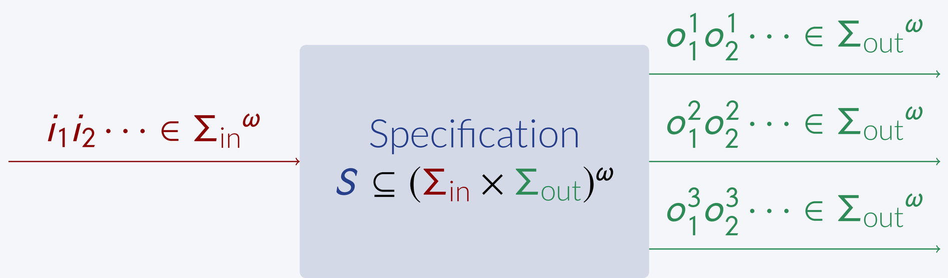
[2]Laboratoire d'Informatique et Systèmes
Aix-Marseille Université

Synthesis Problem



→ For instance, the specification $\text{Mod}_2 = \{(m, n) \mid m \equiv n[2]\}$ can be implemented by Parity : $m \mapsto \begin{cases} 0 \text{ if } m \text{ is even} \\ 1 \text{ if } m \text{ is odd} \end{cases}$

Reactive Synthesis



- Synchronous execution $i_1 o_1 i_2 o_2 \ldots$
- S is given as a **language**

Example

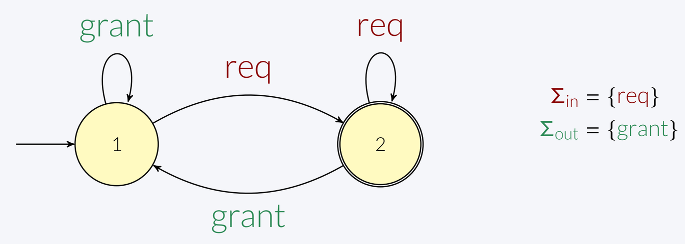


Figure 1. Universal co-Büchi automaton checking that every request is eventually granted.

$$\bigwedge_{c=1}^{n} \Box(\text{req}_c \rightarrow \Diamond \text{grant}_c)$$

$\Sigma_{in} = \{\text{req}_1, \ldots, \text{req}_n\}$
$\Sigma_{out} = \{\text{grant}_1, \ldots, \text{grant}_n\}$

$\Box$: G, "always"
$\Diamond$: F, "in the future"

Figure 2. LTL formula expressing that for each client $c \in \{1, \ldots, n\}$, each of its requests is eventually granted

Known Results

→ Target implementation I is a deterministic transducer

Specification	Complexity
Nondeterministic Büchi Automaton	ExpTime-complete (Büchi & Landweber, 1969)
Linear Temporal Logic formula	2-ExpTime-complete (Pnueli & Rosner, 1989)

Limitations

- Σ_{in} and Σ_{out} are assumed **finite**
- Impractical for *large* alphabets (e.g. Figure 2)
- Cannot handle *unbounded* number of possible inputs

$$\bigwedge_{c \in \mathbb{N}} \Box(\text{req}(c) \rightarrow \Diamond \text{grant}(c))$$

$\Sigma_{in} = \{\text{req}(i) \mid i \in \mathbb{N}\}$
$\Sigma_{out} = \{\text{grant}(i) \mid i \in \mathbb{N}\}$

Figure 3. **Unbounded** number of clients.

⇒ We assume the alphabet is **infinite**

How To Model Executions?

Data Words

Sequences of pairs $(a, d) \in \Sigma \times \mathcal{D}$

- Σ finite alphabet of *labels*
- $\mathcal{D}$ infinite set of *data*

1	2	2	3	1	3	1	
req	req	grt	req	grt	grt	req	…

Figure 4. A data word with labels $\Sigma = \{\text{req}, \text{grt}\}$ and data $\mathcal{D} = \mathbb{N}$.

→ Large literature on data words:

- Kaminski and Francez, 1994
- Segoufin, 2006
- Bojańczyk, David, Muscholl, Schwentick, and Segoufin, 2011
- Schwentick and Zeume, 2012

How To Model Specifications?

Register Automata

Finite automata equipped with a finite set R of registers

- Store data $\downarrow r$
- Formula φ to compare incoming data d with register content $\uparrow r$

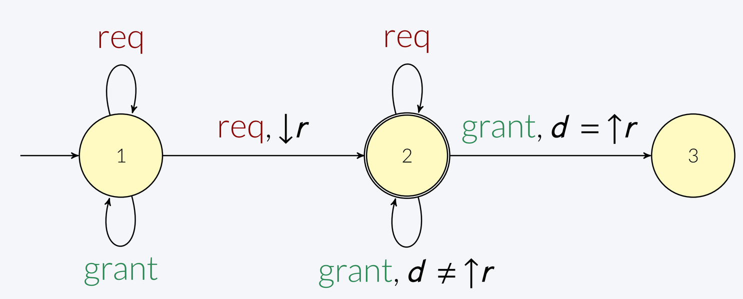


Figure 5. Universal co-Büchi register automaton checking that every request is eventually granted (the $\varphi = \top$ tests are omitted).

Test-Free

- Input transitions do not conduct test on data: φ is always $\top$
- Output transitions output the content of some register: φ is always an equality test $d = \uparrow r$.

→ The register automaton of Figure 5 is not *test-free*.

How To Model Implementations?

Register Transducers

Transitions are $q \xrightarrow{i, \varphi \mid \downarrow r_{in}, o, \uparrow r_{out}} q'$:

- $i \in \Sigma_{in}$, $o \in \Sigma_{out}$ input and output letters
- φ a test over input data
- $r_{in} \in R$ register where the input data is stored
- $r_{out} \in R$ register whose content is output

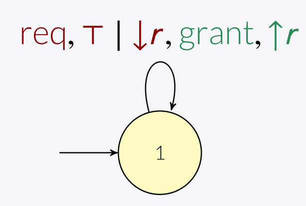


Figure 6. A register transducer immediately granting each request.

Test-Free

- Defined analogously
- Transitions are $q \xrightarrow{i \mid \downarrow r_{in}, o, \uparrow r_{out}} q'$

→ The register transducer of Figure 6 is *test-free*.

Results

Bounded Synthesis

→ The number of registers of the implementation is **fixed**.

Specification (Register Automaton)	Status
Nondeterministic Büchi	Undecidable
Universal co-Büchi	Decidable (Khalimov, Maderbacher, & Bloem, 2018) we provide simpler proof techniques
Nondeterministic Test-Free	Decidable (implementation is also Test-Free)

Proof Techniques

- Reduce to the finite case
- Keep track of equality relations between registers
- Abstract actions by equality types

Future Work

- Examine the case where the number of registers of the implementation is **not fixed**.
- Synthesise from specifications expressed as **logical formulae**.
- Synthesise from **asynchronous** specifications and implementations: no strict alternation between input letters and output letters ⇝ gather information before producing output.
- Relax constraints over the implementation: synthesise an algorithm instead of a transducer (in the spirit of Ehlers, Seshia, & Kress-Gazit, 2014)

References

Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., & Segoufin, L. (2011). Two-variable logic on data words. *ACM Transactions on Computational Logic.*

Büchi, J. R. & Landweber, L. H. (1969). Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society.*

Ehlers, R., Seshia, S. A., & Kress-Gazit, H. (2014). Synthesis with identifiers. In *15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2014).*

Kaminski, M. & Francez, N. (1994). Finite-memory automata. *Theoretical Computer Science, 134*(2), 329–363.

Khalimov, A., Maderbacher, B., & Bloem, R. (2018). Bounded synthesis of register transducers. In *16th International Symposium on Automated Technology for Verification and Analysis (ATVA 2018).*

Pnueli, A. & Rosner, R. (1989). On the synthesis of a reactive module. In *16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1989).*

Schwentick, T. & Zeume, T. (2012). Two-variable logic with two order relations. *Logical Methods in Computer Science.*

Segoufin, L. (2006). Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic.*